2019

Embedded Systems

IOT PROJECT

KAJ WIKMAN, V3

NOVIA | Wolfsskavägen 33

Contents

Introduction to IoT

The IOT is a name for the vast collection of "things" that are being networked together in the home and workplace (up to 20 billion by 2020 according to Gardner, a technology consulting firm). And they may be underestimating it. We all have large numbers of computers in a modern house. Many of them that have IP addresses.

- Networking - these IOT devices talk to one another (M2M communication) or to servers located in the local network or on the Internet. Being on the network allows the device the common ability to consume and produce data.
- Sensing/Sensors - IOT devices sense something about their environment.
- Actuating/Actuators - IOT devices that do something. Lock doors, beep, turn lights on, or turn the TV on.

Of course, not every IOT device will have all three, but these are the characteristics of what we will find out there.

Characterizing an IOT Project

When looking at a new project, the first thing to do to understand an IOT project is to look at the six different aspects for characterizing an IOT project.

- Communications
- Processor Power
- Local Storage
- Power Consumption
- Functionality
- Cost

Communications

Communications are important to IOT projects. In fact, communications are core to the whole genre. There is a trade-off for IOT devices. The more complex the protocols and higher the data rates, the more powerful processor you need and the more electrical power the IOT device will consume.

TCP/IP base communications (think web servers; HTTP-based (like SOAP/REST servers); streams of data; UDP, RPC) provide the most flexibility and functionality at a cost of processor and electrical power.

Low-power BlueTooth and Zigbee types of connections allow much lower power for connections with the corresponding decrease in bandwidth and functionality.

Processor Power

There are several different ways of gauging processor power. Processor speed, processor instruction size, and operating system all play in this calculation. For most IOT sensor and device applications, you will not be limited by processor speed as they are all fast. However, there is <u>one exception</u> to this. If you are using encryption and decryption techniques, then those operations are computationally expensive and require more processor power to run. The trade-off can be that you have to transmit or receive data much more slowly because of the computational requirements of encrypting/decrypting the data. However, for many IOT projects, this is just fine.

Local Storage

Local storage refers to all three of the main types of storage: RAM, EEPROM, and Flash Memory.

- RAM (Random Access Memory) is high-data rate, read/writable memory, generally used for data and stack storage during execution of the IOT program.
- EEPROM (Electrically Erasable Programmable Read Only Memory) is used for writing small amounts of configuration information for the IOT device to be read on power up.
- Flash Memory is generally used for the program code itself. Flash is randomly readable (as the code executes, for example), but can only be written in large blocks and very slowly.

Flash is what you are putting your code into with the Arduino IDE. The amount of local storage (especially RAM) will add cost to your IOT Device. For prototyping, the more the merrier. For deployment, less is better as it will reduce your cost.

Power Consumption

Power consumption is the bane of all IOT devices. If you are not plugging your IOT device in the wall, then you are running off batteries or solar cells and every single milliwatt counts in your design. Reducing power consumption is a complex topic that is well beyond this text. However, the concepts are well understood by the following:

- Put your processor in sleep mode as much as possible.
- Minimize communication outside of your device.
- Try to be interrupt driven and not polling driven.
- Scour your design looking for every unnecessary amount of current.

Functionality

This is kind of a catch-all category that is quite subjective. For example, having additional GPIOs (General Purpose Input Outputs) available is great for flexibility. Having additional serial interfaces are very useful for debugging. Special hardware support for encryption and decryption can make device computer security much simpler. One of the things that is often missing in many IOT prototyping system is software debugging support hardware.

Cost

What is an acceptable cost for your IOT device? That depends on the value of the device and the market for your device. A 2.50e price can be great for prototypes but will be the death of the product in production. You need to size the price to the product and the market.

Real-Time Reference Architecture

To realize the benefits a Real-Time Business solution requires a set of business and technical capabilities that define the process by which you ingest device events, perform advanced analytics, gather insight, and act.

A product development team who is responsible for testing a new business model will need a reference architecture that provides a roadmap for how they can construct a real-time system from connected devices and cloud services.

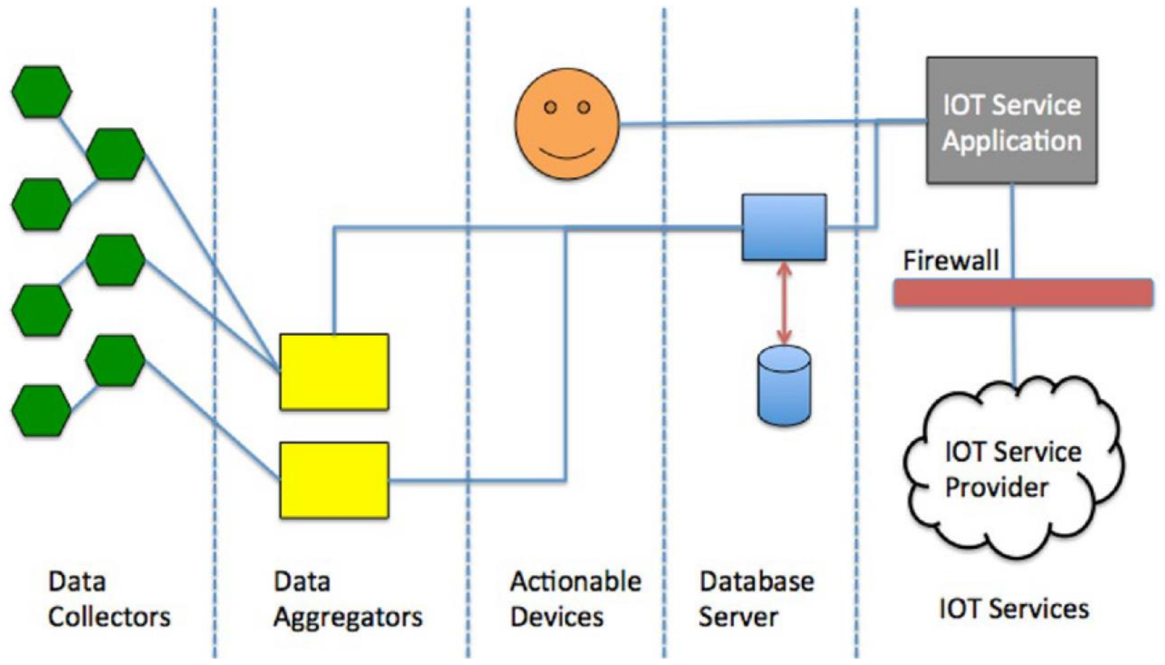


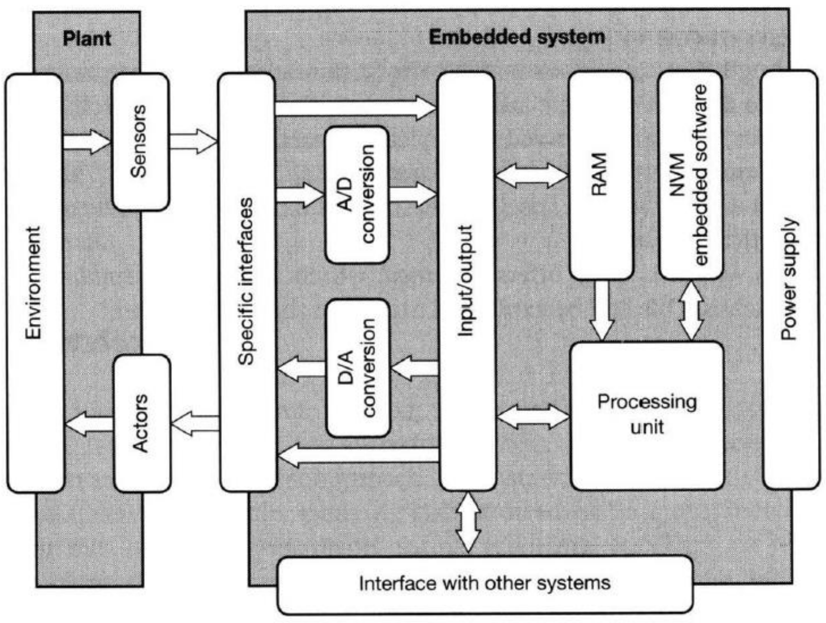
Figure 1 Device architecture

- *Data collector*: A sensor, IOT device, and so on, that produces data from some event or observation.
- *Data aggregator*: A node (embedded controller, microcontroller, small computer, and so on) that receives information from one or more data collectors. Its purpose is to aggregate and augment the data for storage at the next layer.
- *Actionable device*: An IOT device that provides some user-controllable feature such as moving a sensor, operating locks, and so on.
- *Database server*: A node, typically a server that stores the data collected for later retrieval and analysis.
- *IOT services*: A computer system that provides an access layer to the database server and actionable devices. Systems are typically Internet servers or cloud services that allow users to view the data and manipulate the actionable devices.

Embedded system

- Software **embedded** into a **technical system** that interacts with the real physical world, controlling some specific hardware.
- Interaction with the technical system via **actors** and **sensors**

[Broekman&Notenboom2003]



Case Study Problem Statement: Habitat Control Center (HCC) Simplified

HCC-1.0 INTRODUCTION

A Habitat Control Center (HCC) is to contain 48 living quarters and a software system, named Sealed Environment Monitor (SEM), which is to act as a monitor of all the living quarters for the habitat personnel in the HCC[1]

HCC-2.0 SEALED ENVIRONMENT MONITOR (SEM)

The SEM shall monitor all occupied living quarters. There shall be a total of 48 living quarters (in this project <u>one simulated quarters</u> is ok but code so that you code manages many if connected. If you have more than one Arduino, please use it.), each of which may, or may not, be occupied at any one time.

HCC-2.1 MONITORED DATA.

For each occupied living quarter, the SEM shall obtain, as fast as possible, the following environment condition data

- Current temperature (degrees C)
- Current oxygen level (as a percentage).

This information shall be obtained from two sensors that are located inside each living quarter. There shall be one sensor per environmental condition. In the quarter panel there shall be 5 leds per condition: red, yellow, green, yellow, red. (The scaling shall be from min to max so to speak)

HCC-2.2 ALARM CONDITIONS

For each of the items in Paragraph HCC-2.1 the Data Collectors will send sensor values to the Aggregators. The Aggregators shall immediately react to the following situations as indicated and send an alarm message to the SEM: (You can simulate the sensors with potentiometers. There will only be one fan. You may hard code the nominal values.)

- For a changed value that represents a deviation of >=1% but <2% from the nominal values, the yellow led in the panel shall flash at a rate of two times per second.
- For a changed value that represents a deviation of >=2% but less than 3% from the nominal values, the yellow led in the panel shall be lit.
- For a changed value that represents a deviation of >=3% from the nominal values, the red led shall flash at a rate of four times per second.
- For a changed value that represents a deviation of >=5% from the nominal values, the red led shall be lit.
- If the value is about the nominal value, <1%, the green led shall be lit.
- And the same for smaller values but % - signed.

[1] It is important to note that the authors do not have access to actual alarms and sensors, consequently the interfaces are emulated in the software.

HCC-3.0 PANELS and ACTUATORS

Quarter

The panel is located in the quarter. The panel shall have a row of leds for each condition. There shall be a panic button and a fan. When the panic button is pressed it shall light a red button on the Data Collector and send a message to the Aggregator. The Aggregator turn of the electrical feed for the collector and send a message to the control room. The fan shall start when the temperature is more then 10% over the nominal values. It shall stop when the temperature back to nominal value, with a deviation range of 1% of nominal value. This is controlled by the collector it self.

Aggregator

The aggregator panel shall have a stop button that stops the electrical feed for all the collectors connected to the aggregator and send a message to the control room. When the aggregator receives a panic button pressed message it shall lit a red button in its own panel and turn of the electrical feed for the quarter in question, just in case it hasn't already been shutoff, and send a message to SEM.

Control room

The control room shall receive all the messages. The only way to reset an aggregator and a collector when a panic button is pressed shall be with a message from the control room.

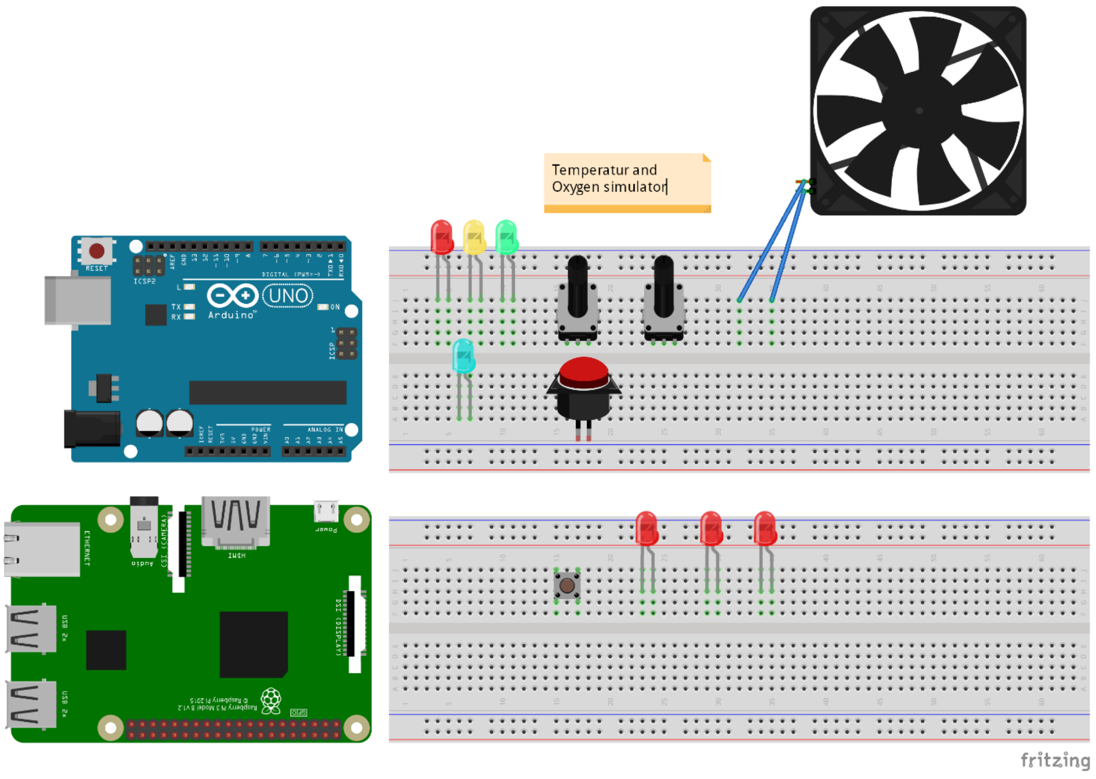
HCC-4.0 NOMINAL VALUES

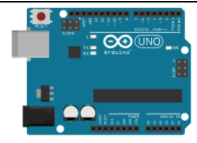
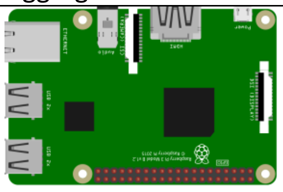
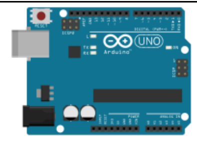
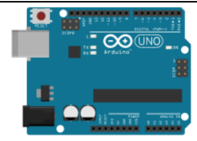
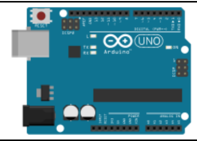
All nominal values shall be found in the database (in our case hard coded values ok). There shall be an option for an Operator in the control room to redefine the values of the environmental conditions maintained in the database. The Operator shall be able to redefine all environmental nominal values to be used for all living quarters and apply this change to SEM processing and sending down the new information.

HCC-5.0 ALARMS and PANIC BUTTONS

The alarm features shall be only turned on/off by the Aggregators and send corresponding messages down to the collector and to the SEM for respective processing. The SEM shall show these warning indicators as long as the alarm condition continues to exist. The SEM shall show the status of the panic buttons and be able to reset the aggregators and the collectors

HCC-7.0 PROTOSCHEMATICS



On-premises	Site	Cloud	Control room	User
Collectors	Aggregators	MQTT Broker	Sem software	Clients
				
				
				
				

- All Collector code shall be done using Arduino C
- All Aggregator code shall be done in c/c++ in Raspberry
- The Mosquitto MQTT program shall be used as broker. On a computer. OR using Azure IoT as the broker.
- The web app shall be on yet an other computer. Free choice for web server and language.
- One client, web browser, shall have a connection to the web app.