LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & SCIENCE, BHOPAL

CS-801

Internet Of Things

Vision of the Department

To be recognized for keeping innovation, research and excellence abreast of learning in the field of computer science & engineering to cater the global society.

Mission of the Department

M1: To provide an exceptional learning environment with academic excellence in the field of computer science and engineering.

M2: To facilitate the students for research and innovation in the field of software, hardware and computer applications and nurturing to cater the global society.

M3: To establish professional relationships with industrial and research organisations to enable the students to be updated of the recent technological advancements.

M4: To groom the learners for being the software professionals catering the needs of modern society with ethics, moral values and full of patriotism.

Program Educational Objectives (PEO's)

PEO1: The graduate will have the knowledge and skills of major domains of computer science and engineering in providing solution to real world problems most efficiently.

PEO2: The graduate will be able to create and use the modern tools and procedures followed in the software industry in the relevant domain.

PEO3: The graduate will be following the ethical practices of the software industry and contributing to the society as a responsible citizen.

PEO4: The graduate will have the innovative mindset of learning and implementing the latest developments and research outcomes in the computer hardware and software to keep pace with the fast changing socio economic world.

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & SCIENCE, BHOPAL

CS-801

Internet Of Things

COURSE OUTCOMES

CO1: Discuss Internet of Things and its hardware and software components

CO2: Design Interface I/O devices, sensors & communication modules.

CO3: Analyze data from various sources in real-time

CO4: Monitor data and devices with remote control.

CO5: Develop real life IoT based projects.

LIST OF EXPERIMENTS

1. To study IOT, their characteristics of components and basic awareness of Arduino/ Raspberry Pi.
2. To study various supporting OS platforms for Raspberry-Pi /Beagle board.
3. Study of Connectivity and Configuration of Raspberry-Pi/ Beagle Board circuit with basic peripherals, LEDs, Understanding GPIO and its use in program.
4. Experiment on connectivity of Rasberry Pi with existing system components.
5. Experiment on application framework and embedded software agents for IoT Toolkit.
6. Experiment on HTTP-to-COAP semantic mapping Proxy in IoT Toolkit.
7. Experiment on Gate way as a service deployment in IoT Toolkit.
8. Experiment on application framework and embedded software agents for IoT Toolkit.
9. Exercise on working principle of Rasberry Pi.
10. Experiment on connectivity of Rasberry Pi with existing system components.

CS-801

Internet Of Things

EXPERIMENT -1

Aim: To study IOT, their characteristics of components and basic awareness of Arduino/ Raspberry Pi.

Theory

The Internet of Things (IoT) is a scenario in which objects, animals or people are provided with single identifiers and the capability to automatically transfer and the capability to automatically transfer data more to a network without requiring human-to-human or human-to-computer communication.

Arduino Board

1. An Arduino is actually a microcontroller based kit
2. It is basically used in communications and in controlling or operating many devices
3. Arduino UNO board is the most popular board in the Arduino board family
4. In addition, it is the best board to get started with electronics and coding
5. Some boards look a bit different from the one given below, but most Arduino's have majority of these components in common
6. It consists of two memories- Program memory and the data memory.
7. The code is stored in the flash program memory, whereas the data is stored in the data memory.
8. Arduino Uno consists of 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

Various components on the Arduino board is shown in fig 1 and its brief description below:

1. Power USB

Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection (1).

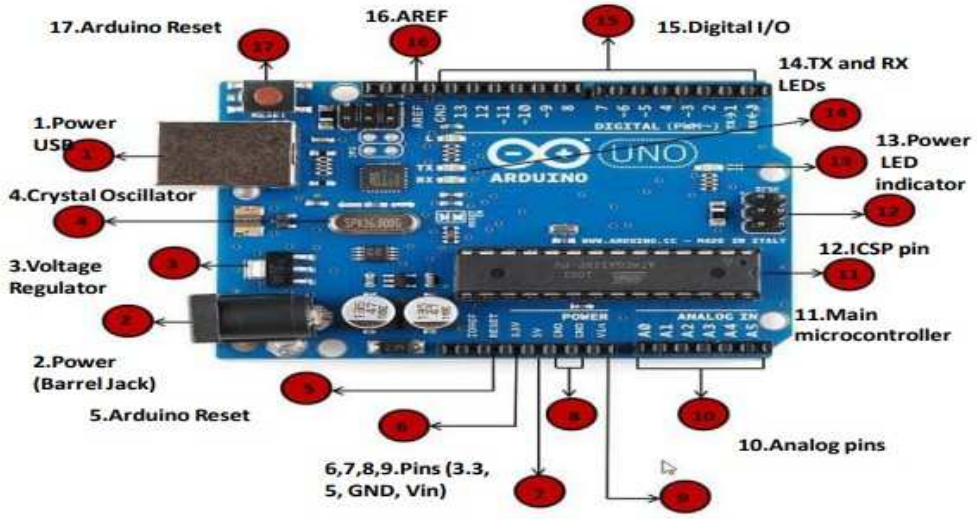


Fig 1 Components used in Arduino

2. Power (Barrel Jack)

Arduino boards can be powered directly from the AC mains power supply by connecting it to the Barrel Jack (2).

3.Voltage Regulator

The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.

4. Crystal Oscillator

The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.

5, 17. Arduino Reset

You can reset your Arduino board, i.e., start your program from the beginning. You can reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).

6, 7, 8, 9.Pins (3.3, 5, GND,Vin)

- 3.3V (6) − Supply 3.3 output volt
- 5V (7) − Supply 5 output volt
- Most of the components used with Arduino board works fine with 3.3 volt and 5 volt.
- GND (8)(Ground) − There are several GND pins on the Arduino, any of which can be used to ground your circuit.
- Vin (9) − This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.

10. Analog pins

The Arduino UNO board has six analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.

11. Main microcontroller

Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.

12. ICSP pin

Mostly, ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the master of the SPI bus.

13. Power LED indicator

This LED should light up when you plug your Arduino into a power source to indicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection.

14. TX and RX LEDs

On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.

15. Digital I/O

The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc. The pins labeled “~” can be used to generate PWM.

16. AREF

AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

Program an Arduino

CS-801

Internet Of Things

1. The most important advantage with Arduino is the programs can be directly loaded to the device without requiring any hardware programmer to burn the program.

2. This is done because of the presence of the 0.5KB of Bootloader which allows the program to be burned into the circuit.

3. All we have to do is to download the Arduino software and writing the code.

4. The Arduino tool window consists of the toolbar with the buttons like verify, upload, new, open, save, serial monitor.

5. It also consists of a text editor to write the code, a message area which displays the feedback like showing the errors, the text console which displays the output and a series of menus like the File, Edit, and Tools menu.

Basic Adruino functions are:

1. digitalRead(pin): Reads the digital value at the given pin.
2. digitalWrite(pin, value): Writes the digital value to the given pin.
3. pinMode(pin, mode): Sets the pin to input or output mode.
4. analogRead(pin): Reads and returns the value.
5. analogWrite(pin, value): Writes the value to that pin.

CS-801

Internet Of Things

EXPERIMENT -2

Aim: To study various supporting OS platforms for Raspberry-Pi /Beagle board.

Theory

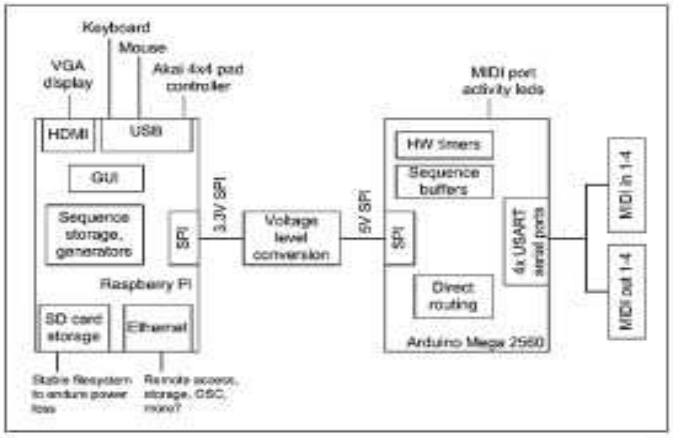
Internet of Things: - The Internet of Things refers to the ever-growing network of physical objects that feature an IP address for internet connectivity, and the communication that occurs between these objects and other Internet-enabled devices and systems. The Internet of Things (IoT) refers to the use of intelligently connected devices and systems to leverage data gathered by embedded sensors and actuators in machines and other physical objects.

Examples of IoT

1) Apple Watch and Home Kit.
2) Smart Refrigerator.
3) Smart Refrigerator.
4) Smart cars.
5) Google Glass.
6) Smart thermostats.

Raspberry-Pi:-The Raspberry Pis a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote teaching of basic computer science in schools and in developing countries. It does not include peripherals (such as keyboards and mice). The Raspberry Pis a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. The Raspberry Pis a credit-card-sized computer that costs between $5 and $35. It's available anywhere in the world, and can function as a proper desktop computer or be used to build smart devices. A Raspberry Pi is a general-purpose computer, usually with a Linux operating system, and the ability to run multiple programs. Raspberry Pi is like the brain. Its

primary advantage comes in processing higher level processing capability. It's a single board computer.



Theory:

1. Raspberry-Pi: -

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the
Raspberry Pi Foundation to promote teaching of basic computer science in schools and in

Theory:

1. Raspberry-Pi: -

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the
Raspberry Pi Foundation to promote teaching of basic computer science in schools and in

Theory:

1. Raspberry-Pi: -

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the
Raspberry Pi Foundation to promote teaching of basic computer science in schools and in

Theory:

1. Raspberry-Pi: -

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the
Raspberry Pi Foundation to promote teaching of basic computer science in schools and in

Theory:

1. Raspberry-Pi: -

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the
Raspberry Pi Foundation to promote teaching of basic computer science in schools and i

Fig 2. Raspberry-Pi Architecture

Various components on the Raspberry Pi board:

ARM CPU/GPU: This is a Broadcom BCM2835 System on a Chip (SoC) that's made up of an ARM central processing unit (CPU) and a Video core 4 graphics processing unit (GPU). The CPU handles all the computations that make a computer work (taking input, doing calculations and producing output), and the GPU handles graphics output.

GPIO: These are exposed general-purpose input/output connection points that will allow the real hardware hobbyists the opportunity to tinker.

RCA: An RCA jack allows connection of analog TVs and other similar output devices.

Audio out: This is a standard 3.55-millimeter jack for connection of audio output devices such as headphones or speakers.

Beagle Board: - The Beagle Board is a low-power open-source single-board computer produced by Texas Instruments in association with Digi-Key and Newark element14. The Beagle Board was also designed with open source software development in mind, and as a way of demonstrating the Texas Instrument's OMAP3530 system-on-a-chip.[8] The board was developed by a small team of engineers as an educational board that could be used in colleges around the world to teach open source hardware and software capabilities. It is also sold to the public under the Creative Commons share-alike license. The board was designed using Cadence OrCAD for schematics and Cadence Allegro for PCB manufacturing; no simulation software was used.

EXPERIMENT -3

Aim: Study of Connectivity and Configuration of Raspberry-Pi/ Beagle Board circuit with basic peripherals, LEDs, Understanding GPIO and its use in program.

Theory

Connecting Hardware Peripherals to Raspberry-Pi board. Raspberry-Pi setup through SSH (Headless Configuration of Raspberry-Pi-3 (VNC, Putti)).

How to Connect a Raspberry-Pi to the Laptop or PC Display

CS-801

Internet Of Things

Required devices:

1. Raspberry Pi
2. Ethernet Cable
3. Laptop/ PC
4. SD Card with Raspbian
5. Micro USB Cable

How does it Work?

To connect a Raspberry Pi to a laptop or PC display, you can simply use an Ethernet cable.

The Raspberry Pi's desktop GUI can be viewed through the laptop or PC display using a 100mbps

Ethernet connection between the two.

We used VNC server software to connect the Raspberry-Pi to our laptop or PC.

Installing the VNC server on your Raspberry-Pi allows you to see the Raspberry Pi's desktop remotely.

1. Setting up your Raspberry Pi
2. Install Raspbian OS on blank SD card.
3. Insert this SD card into Raspberry-Pi board.
4. Connect micro USB cable to power the Raspberry-Pi.
5. Sharing Internet Over Ethernet in Window OS

This step explains how you can share your laptop or PC with the Raspberry Pi via Ethernet cable.

To share internet with multiple users over Ethernet, go to Network and Sharing Center.

Then click on the WiFi network. Double click on Wireless area connection. Click on Properties

Go to "Sharing" tab and click on "Allow other network users to connect".

After this, make sure that the networking connection is changed to "Local Area Connection"

Now, to check the IP assigned to the network established, click on the new local area connection link created:

Now open command prompt.

CS-801

Internet Of Things

Ping the broadcast address of your IP. (Type) E.g. : ping 192.168.137.1

Stop the ping after 5 seconds.

To get the IP address of Raspberry Pi in the established network, use the Software "Advance IP Scanner". It is free software.

Setting up the VNC Server to Connect Your Raspberry Pi to the Laptop or PC Display

First install VNC server and Putty on your laptop/ PC

Open Putty Software, and enter login ID: pi and Password: raspberry.

After that, enter commands into Putty i.e,

You will be prompted to enter and confirm a password.

This will be asked only once, during first time setup.

Enter an 8 digit password. Setting Up the Client Side (Laptop or PC) Download VNC client and install it.

When you first run VNC viewer, enter the IP address of your Raspberry Pi given dynamically by your laptop and append with : 1(denoting port number) and press connect.

You will get a warning message, press 'Continue'. Enter the 8 digit password which was entered in the VNC server installation on your Raspberry Pi. Finally, the Raspberry Pi desktop should appear as a VNC window. You will be able to access the GUI and do everything as if you are using the Pi's keyboard, mouse, and monitor directly.

GPIO

A powerful feature of the Raspberry Pi is the row of GPIO (general-purpose input/output) pins along the top edge of the board. A 40-pin GPIO header is found on all current Raspberry Pi boards (unpopulated on Pi Zero and Pi Zero W). Prior to the Pi 1 Model B+ (2014), boards comprised a shorter 26-pin header

Voltages

Two 5V pins and two 3V3 pins are present on the board, as well as a number of ground pins (0V), which are unconfigurable. The remaining pins are all general purpose 3V3

Pins, meaning outputs are set to 3V3 and inputs are 3V3-tolerant.

EXPERIMENT NO. 4

Aim: Experiment on connectivity of Rasberry Pi with existing system components.

Theory

The raspberry pi comes in two models, they are model A and model B. The main difference between model A and model B is USB port. Model A board will consume less power and that does not include an Ethernet port. But, the model B board includes an Ethernet port and designed

in china. The raspberry pi comes with a set of open source technologies, i.e. communication and multimedia web technologies.In the year 2014, the foundation of the raspberry pi board launched the computer module, that packages a model B raspberry pi board into module for use as a part of embedded systems, to encourage their use.

Raspberry Pi Hardware Specifications

The raspberry pi board comprises a program memory (RAM), processor and graphics chip, CPU, GPU, Ethernet port, GPIO pins, Xbee socket, UART, power source connector. And various interfaces for other external devices. It also requires mass storage, for that we use an SD flash memory card. So that raspberry pi board will boot from this SD card similarly as a PC boots up into windows from its hard disk.

Essential hardware specifications of raspberry pi board mainly include SD card containing Linux OS, US keyboard, monitor, power supply and video cable. Optional hardware specifications include USB mouse, powered USB hub, case, internet connection, the Model A or B: USB WiFi adaptor is used and internet connection to Model B is LAN cable.

GPU (Graphics Processing Unit)

The GPU is a specialized chip in the raspberry pi board and that is designed to speed up the operation of image calculations. This board designed with a Broadcom video core IV and it supports OpenGL

Ethernet Port

The Ethernet port of the raspberry pi is the main gateway for communicating with additional devices. The raspberry pi Ethernet port is used to plug your home router to access the internet.

GPIO Pins

The general purpose input & output pins are used in the raspberry pi to associate with the other electronic boards. These pins can accept input & output commands based on programming raspberry pi. The raspberry pi affords digital GPIO pins. These pins are used to connect other

electronic components. For example, you can connect it to the temperature sensor to transmit digital data.

XBee Socket

The XBee socket is used in raspberry pi board for the wireless communication purpose.

Power Source Connector

The power source cable is a small switch, which is placed on side of the shield. The main purpose of the power source connector is to enable an external power source.

UART

The Universal Asynchronous Receiver/ Transmitter is a serial input & output port. That can be used to transfer the serial data in the form of text and it is useful for converting the debugging code.

Display

The connection options of the raspberry pi board are two types such as HDMI and Composite.Many LCD and HD TV monitors can be attached using an HDMI male cable and with a low-cost adaptor. The versions of HDMI are 1.3 and 1.4 are supported and 1.4 version cable is recommended. The O/Ps of the Raspberry Pi audio and video through HMDI, but does not support HDMI I/p. Older TVs can be connected using composite video. When using a composite video connection, audio is available from the 3.5mm jack socket and can be sent to

your TV. To send audio to your TV, you need a cable which adjusts from 3.5mm to double RCA connectors.

Model a Raspberry Pi Board

The Raspberry Pi board is a Broadcom (BCM2835) SOC (system on chip) board. It comes equipped with an ARM1176JZF-S core CPU, 256 MB of SDRAM and 700 MHz,. The raspberry pi USB 2.0 ports use only external data connectivity options. The board draws its power from a micro USB adapter, with min range of 2. Watts (500 MA). The graphics, specialized chip is

designed to speed up the operation of image calculations. This is in built with Broadcom video core IV cable, that is useful if you want to run a game and video through your raspberry pi.

Features of Raspberry PI Model A

- The Model A raspberry pi features mainly includes
- 256 MB SDRAM memory
- Single 2.0 USB connector
- Dual Core Video Core IV Multimedia coprocessor
- HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC) Video Out
- 3.5 MM Jack, HDMI, Audio Out
- SD, MMC, SDIO Card slot on board storage
- Linux Operating system
- Broadcom BCM2835 SoC full HD multimedia processor
- 8.6cm*5.4cm*1.5cm dimensions

EXPERIMENT -5

Aim: Experiment on application framework and embedded software agents for IoT Toolkit.

An embedded framework is a set of tools used by software developers to build programs for specific applications . The proposed embedded framework aims to assist the developers in configuring and interacting with the microcontrollers' peripherals seamlessly and independently

of the hardware specifics. Embedded systems represent the core of the IoT devices that requires the capability to sense, actuate, and communicate within a network infrastructure. Although IoT devices target different application domains, they share common design challenges to build optimal and cost-effective heterogeneous sensors and actuator networks.

Typically, embedded software professionals deal with multiple microcontroller platforms to deliver cost-effective solutions for final product deployment. Therefore, they spend a relevant amount of time analyzing hardware specifications and configuring device peripherals; also, application software reusability across platforms is limited by hardware differences. This section presents a comparative analysis of development tools based on the common requirements for IoT solutions, then the proposed framework and code generator design and implementation details are presented.

2.1. Internet of Things Design and Development Tools

Broadly speaking, an IoT system architecture is composed of three layers: the perception layer, the network layer, and the application layer. The perception or sensing layer is composed of heterogeneous smart and interconnected devices that interact with physical objects to measure, collect, and process their state information while transmitting data into the IoT network . Different communication technologies in the network layer, such as Bluetooth, Zigbee, Wifi, LoRa, LTE, and 5G, are used to transfer data from the devices to a cloud server for further analysis and decision-making that takes place in the application layer.

An IoT device comprises a microcontroller, sensors, actuators, a communication transceiver, and a power unit, as depicted in **Figure 1**. To implement the interface with the sensor, the microcontroller requires the use of an analog-to-digital converter unit (ADC), while the interface with the actuator requires a digital-to-analog unit (DAC), which may include the use of on/off or pulse width modulated (PWM) signals. On the other hand, the interface with the transceiver is typically implemented through serial communication using protocols such as UART (universal asynchronous receiver transmitter), SPI (serial peripheral interface), and I2C (inter-integrated circuit), among others. Examples of implementations of IoT devices for different applications that include these functional elements can be found in [41,42,43,44,45,46,47].

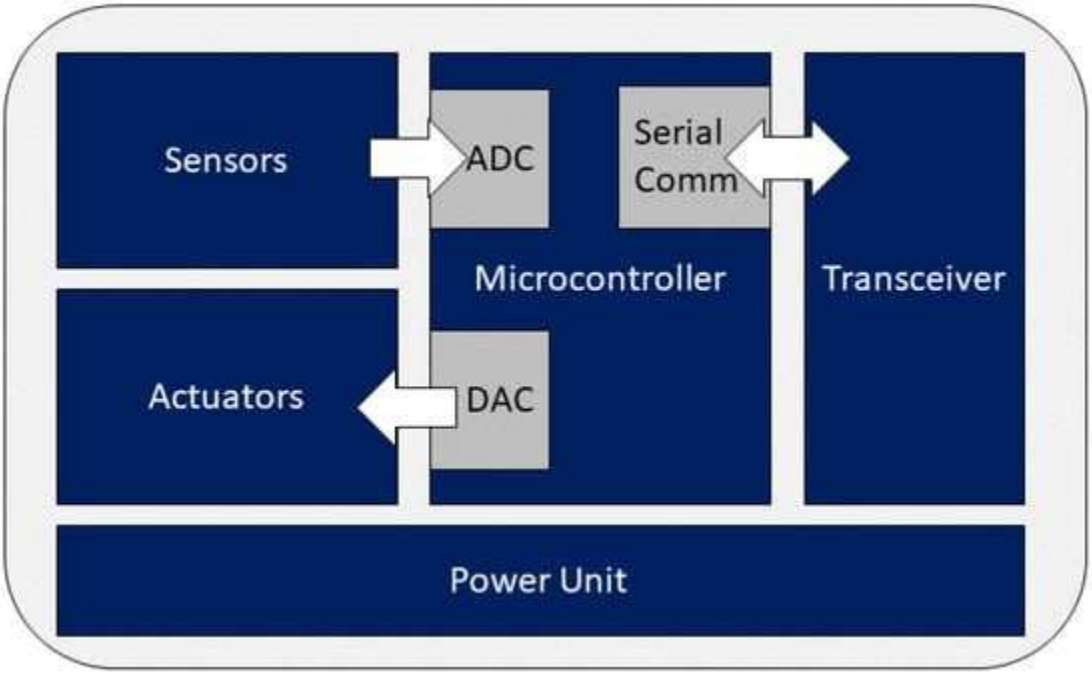


Figure 1. Basic functional elements of an IoT device.

IoT devices typically operate with energy, memory, and computational constraints; therefore, designers must select the adequate microcontroller according to the specific needs of the applications to implement cost-effective solutions. For example, high-performance microcontrollers may be too expensive for basic IoT devices, while low-end microcontrollers may not have enough computational power to support more complex IoT tasks. Additionally, the software developer must consider memory management and energy consumption when implementing the algorithms for IoT devices.

Based on these previous development requirements, **Table 1** summarizes the comparative analysis of different software development frameworks conducted to evaluate the following features:

Software Development Framework Architecture

The proposed software development framework has been designed to allow the migration of an IoT application code into any microcontroller regardless of its family, architecture, and vendor. This allows faster code development, promotes code reusability, and encourages developers to select the adequate microcontroller according to the application requirements.

The framework architecture has been divided into a static element, a dynamic element, and a set of wrappers. **Figure 2** shows the architecture of a typical embedded application that uses the framework, which contains the application code, an optional operating system, the framework,

and vendor drivers. The architecture allows either the use of an embedded operating system-based implementation or the bare-metal approach.

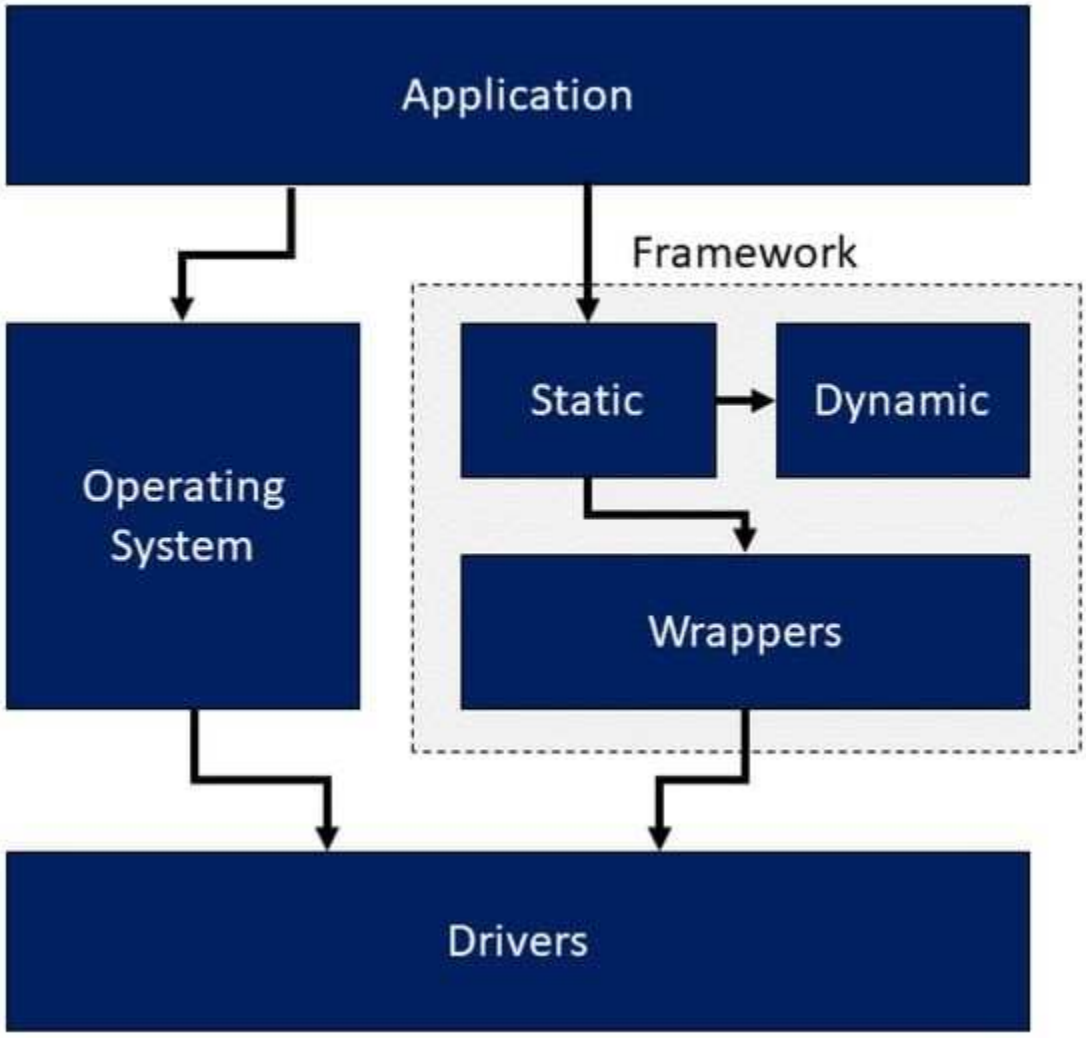


Figure 2. Architecture of an embedded application using the framework as a software development tool.

The application code includes the software modules that do not directly interface with the hardware, but they implement functions and sequential logic required for the system to behave according to the device requirements. Even though this element is represented as a single module, it can contain several modules, each in charge of specific functionality of the IoT device, e.g., network communication, control algorithm, and data acquisition.

The driver element is composed of different code files, which the microcontroller vendors usually provide. These files contain functions, definitions, and data types that can be used to configure and use the different elements in the device, such as general purpose input/output pins

and clock sources, among others. The drivers are specific for each microcontroller, and there might be huge differences among them, even if they come from the same vendor.

The three elements that compose the framework are: static, dynamic, and vendor wrappers. The static element's primary function is to abstract the different operations that can be performed with the microcontroller and its peripherals. The functions defined within this element are generic in order to be able to handle the peripherals without knowing any specific information about the microcontroller being used. This element acts as a bridge between the application module and the other framework elements, and since it contains generic functions, it allows the application code that uses the framework to be used in any microcontroller without requiring any modification. Each peripheral in the microcontroller requires a set of files for the static element (peripheral.h and peripheral.c files), and every microcontroller supported by the framework can reuse this.

The dynamic element contains the files produced by the code generator, which contain the description of how each peripheral used by the application should be configured. The static element consumes these files in order to perform this configuration. The files from this element are application-specific and, therefore, must be one set of files for each peripheral used (peripheral_cfg.c and peripheral_cfg.h files).

The wrappers act as translators between the functions from the static element and the functions contained within the vendor drivers. These files contain generic labels connected with data types and definitions from the vendor drivers. The wrappers also contain functions from the static element that make direct calls to the code contained within the drivers to interact with the peripherals. These labels and functions allow portability in the application module between the different supported microcontrollers because they are defined in the wrappers of every microcontroller. The application module needs to call the functions from the static element using these labels, and the framework will translate and perform the desired operation in the microcontroller. Since the wrappers depend on the microcontroller characteristics, a set of these files (peripheral_wrapper.c and peripheral_wrapper.h files) must be created for each peripheral contained in each microcontroller supported by the framework.

2.3. Supported Peripherals

The peripherals that are currently supported by the framework are the general purpose input/outputs (GPIOs), the analog-to-digital converter (ADC), and the universal asynchronous receiver-transmitter (UART). These represent the essential elements for an IoT device.

The GPIO framework element is formed by the files gpio.c and gpio.h for the static element, gpio_cfg.c and gpio_cfg.h for the dynamic.c and gpio_wrapper.h for the wrappers. This module allows the user to configure the following fields on any GPIO contained in a microcontroller: mode (input, output, or used by a microcontroller peripheral), pull resistor (pull-

up, pull-down, or none), speed (if supported), output type (open drain or push-pull) and the alternate mode, in the case a peripheral is using the GPIO as another function such as UART or ADC. The framework also allows the user to read and write either to individual pins or the complete port in the microcontroller. The currently supported GPIO functions are shown in **Figure 3**.

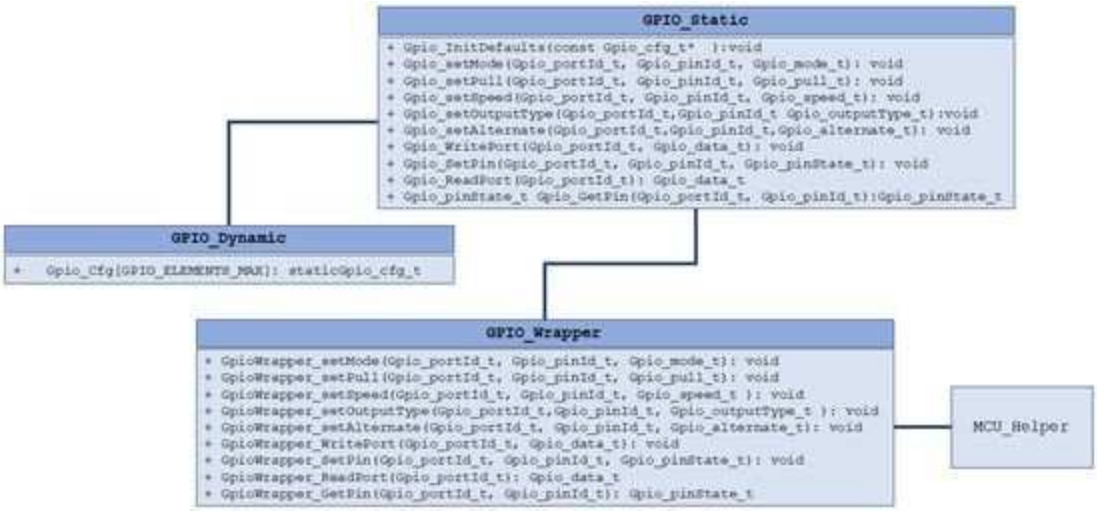


Figure 3. General Purpose Input/Outputs functions diagram.

The ADC framework module is formed by the files adc.c and adc.h for the static element, files adc_cfg.c and adc_cfg.h for the dynamic element, and .c and adc_wrapper.h for the wrappers. This module allows the user to configure the following fields on any ADC contained in the microcontroller: the clock source, the clock pre-scaler, conversion channel, resolution in bits, number of samples, voltage reference, and the justification (right or left) to store the result in a register. The framework also allows the user to enable or disable the ADC, start a conversion, check the conversion status, and get the conversion result for any ADC in the microcontroller. The supported functions are illustrated in **Figure 4**.

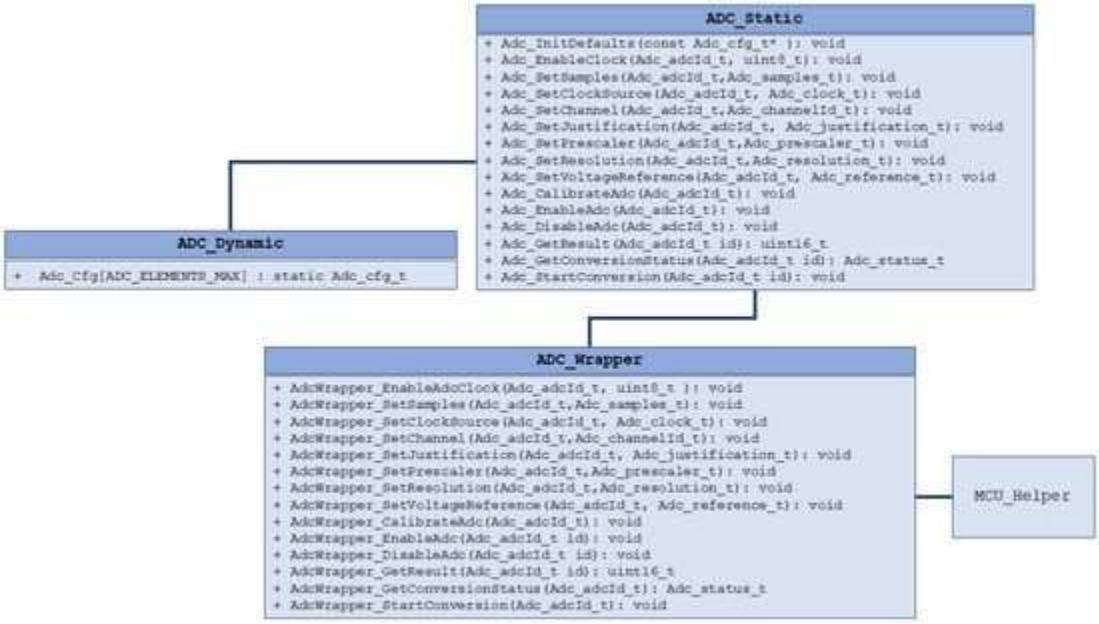


Figure 4. Analog-to-Digital Converter functions diagram.

The UART framework module is formed by the files uart.c and uart.h for the static element, uart_cfg.c and uart_cfg.h for the dynamic element, and uart_wrapper.c and uart_wrapper.h for the wrappers. This module allows the user to configure the following fields on any UART contained within the microcontroller: the clock source, the clock pre-scaler, the baud rate, the number of stop bits, the number of data bits, and the parity (even, odd, or no parity). The framework also allows the user to enable or disable the UART and send or receive data through the polling method. The UART functions are shown in Figure 5.

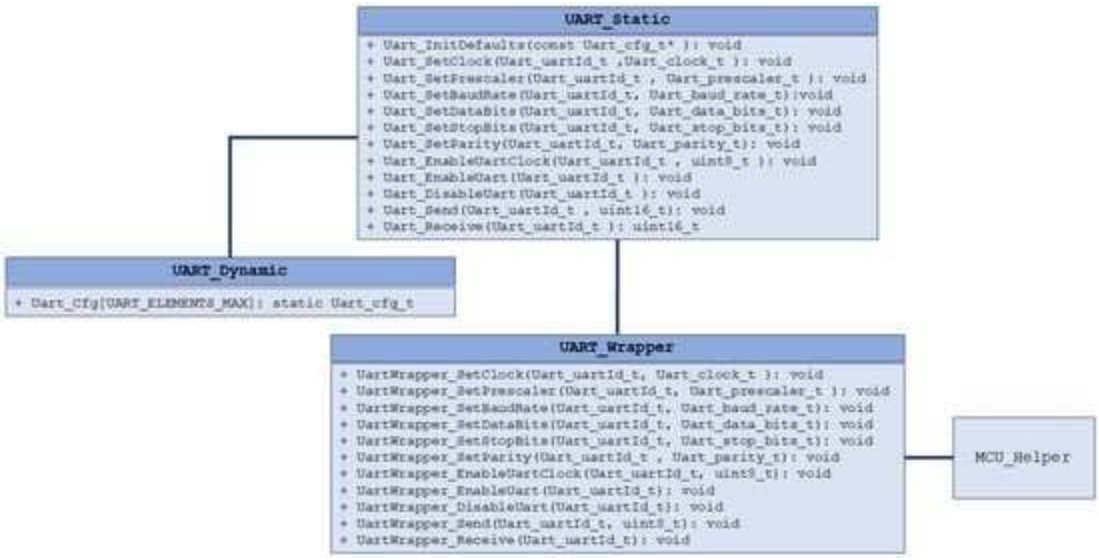


Figure 5. Universal Asynchronous Receiver-Transmitter functions diagram.

2.4. Code Generator Implementation

The code generator consists of a desktop application that allows the user to select the project's microcontroller, configure peripherals, create the dynamics elements, and integrate them into the framework. **Figure 6** shows the software modules that implement the code generator.

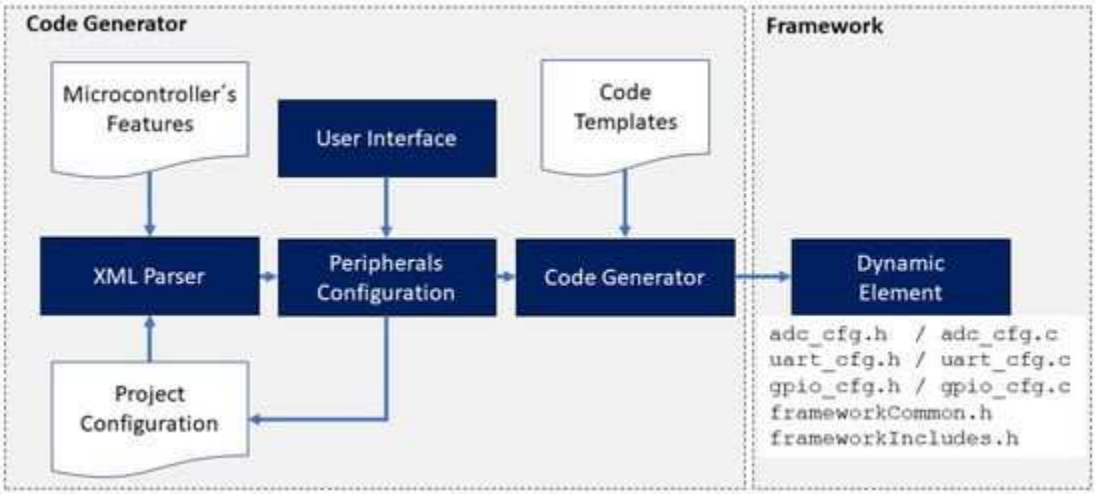


Figure 6. Implementation of the code generator that provides the dynamic elements to the framework.

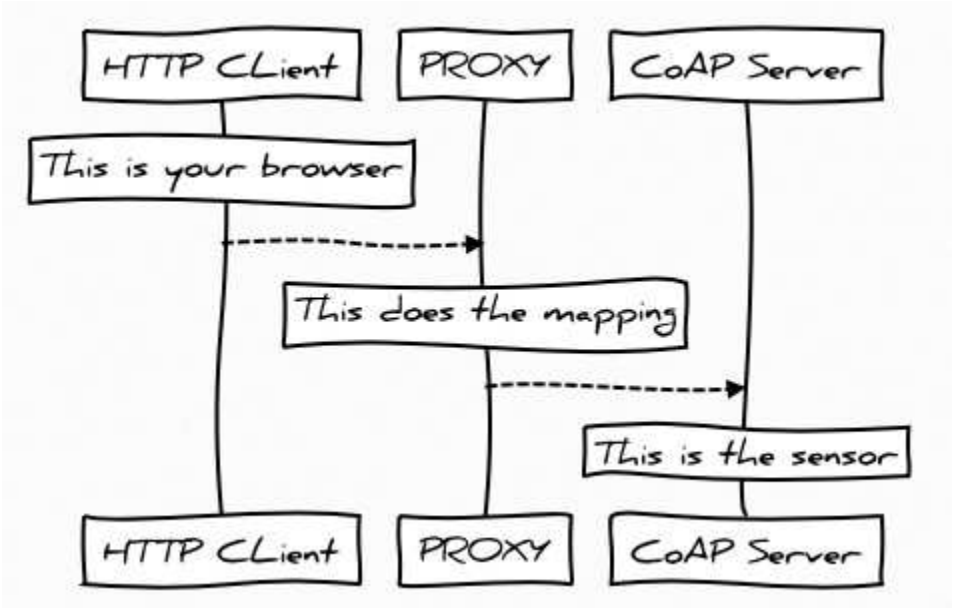
CS-801
Internet Of Things

EXPERIMENT NO. 6

Aim - Experiment on HTTP-to-COAP semantic mapping Proxy in IoT Toolkit.Tcl

When discussing the state of current standardization I often find myself having to think through on how to summarize and simplify what a draft is about. I find that exercise very useful and I thought of providing a series of short summaries about the drafts the WG is currently working on.

The point of the draft (soon RFC) is to describe how to do HTTP-to-CoAP Mapping to allow HTTP clients to access a CoAP Server via a proxy. This works under the assumption that, despite the similarities between CoAP and HTTP, not all browsers will implement support for it and that some legacy devices will need proxying. Another assumption is that users will like to use their smartphones with their home sensors. The set up would look like this:



CS-801

Internet Of Things

HTTP-to_CoAP Mapping Scenario with world-class graphics

The mapping is fairly straightforward as CoAP is designed to mirror HTTP. In its simple form, you simply append one URI after the other, for example:

http://p.example.com/hc/coap://s.example.com/light

If you are using other mappings, you might do a search on ./well-known/core to discover them, they should be using resource type: core.hc and attribute type hct.

In the enhanced form more sophisticated mappings can be expressed. And certain template variables have been created for it:

s = "coap" / "coaps"

hp = host [":" port]

p = path-abempty

q = query

qq = ["?" query}]

The ABNF forms make use of RFC5234, RFC7252 and RFC6690, for simplicity you can check the list of CoAP-related ABNF forms I made. Thanks to all this, we can specify for example that we want to use secure CoAP and query for lights that are on.

Req: GET http://p.example.com/hc?s=coaps&hp=s.example.com&p=/light&q=on

Discovery is also important both on the HTTP and the CoAP side. A sample HTTP discovery on the proxy would look like this:

Req: GET /.well-known/core?rt=core.hc HTTP/1.1

Host: p.example.com

Res: HTTP/1.1 200 OK

Content-Type: application/link-format

Content-Length: 18

// if plain link-format

</hc>;rt="core.hc"

// if JSON link-format

[{"href":"/hc","rt":"core.hc"}]

If it is the CoAP devices that are querying the proxy, they should also get back an anchor URI of the HC proxy as well as the URI mapping.

Req: GET coap://[ff02::1]/.well-known/core?rt=core.hc

Res: 2.05 Content

</hc>;anchor="http://p.example.com";

rt="core.hc";hct="?uri={+tu}"

The HC proxy also performs bidirectional Media Type Mapping of Media Types and content encodings into CoAP Content-Formats

The mapping will depend on whether the application is tightly or loosely coupled with the proxy. For HTTP unsupported media types the HC Proxy should simply answer with a 415 Unsupported Media Type response. When dealing with an unrecognised CoAP "cf" the HC proxy can use the application/coap-payload and append that content format ;cf= whichever is the content format number.

Although possible, content transcoding is generally not advisable as it would tamper the payload and might cause loss of information (and mess other things in general).

At a really high level view, that is what the draft is about, if you are interested I really encourage you to read the draft, and if you want to use it there are several implementations available:

1. Squid HTTP-CoAP mapping module.
2. HTTP-CoAP proxy based on EvCoAP.
3. Luebeck implementation
4. Californium also supports a HTTP-CoAP proxy function but they do not explicitly state that it is based on the draft.

CS-801
Internet Of Things

.

Experiment-7

Aim: Experiment on Gate way as a service deployment in IoT Toolkit

IoT interconnects physical world "Things" by utilizing software and networking technologies.
Due to its roots in traditional sensor networks, connected physical objects are resource-constrained devices, and require competent communication protocol for energy efficiency.
First wave of IoT application in smart city domain emphasized on connecting sensor interfacing with physical-world using lightweight protocols such as CoAP and XMPP [5][6]. In later stages, traditional Internet state transfer protocol such as REST is used for similar applications, where event-centric frameworks had been implemented to reduce number of messages transmitted [7].
The 'Smart-Object' devices with domain specific intelligence are rapidly replacing first wave of IoT devices [8]. Although these devices do not utilize semantic technologies, they provide
higher-level of awareness from the sensor than just plain raw sensor data.
The IoT domain has been started getting congested with heterogeneous applications using different communication protocols and data models [9]. Various organizations such as Open IoT alliance, AllSeen alliance, and IPSO alliance are working on standardization of communication protocols to provide interoperability between various vendors silos [10][11][12]. Organization
such as Internet Engineering Task Force (IETF) and XMPP standards foundation are trying to
scale their messaging protocols, CoAP and XMPP, respectively, to align with other protocols.

These efforts are scattered and largely focus on solving problems around one protocol instead of providing integration solution.

In Web-centric infrastructure, acquisition of contextual information from raw sensor data requires annotation of sensor data with semantic metadata. Key standardization efforts that have sought to establish sensor data models for sensors to be accessible and controllable via the Web include:

- OGC Sensor Web Enablement (SWE)

The SWE efforts established by the Open Geospatial Consortium include following important specifications: Observation & Measurement (O&M), Sensor Model Language (SensorML) and Sensor Observation Service (SOS)[13]. The O&M and SensorML contain standard model and XML schema for observations/measurements and sensors/processes respectively. The SOS is a standard service model, which provides mechanism for querying observation and sensor metadata.

- Semantic Sensor Network (SSN) ontology

The SSN ontology, developed by W3C provides a standard for modeling sensor devices, sensor platforms, knowledge of the environment and observations[14] [15]. The SSN provides a foundation in the direction of achieving interoperability between the interconnected IoT Silos.

- Semantic Sensor Observation Service (SemSOS)

The Semantic Web enabled implementation of SOS, SemSOS, provides a rich semantic backend (knowledge base) while retaining the standard SOS specifications/service interactions. A semantically intelligent client can utilize this capability of SemSOS to derive higher level abstractions from the annotated sensor data [16] by implementing a semantic reasoning service acting on the knowledge base. SemSOS is the principal component of Semantic Sensor Web [17].

Although the utilization of these standards provide integration of Semantic Web with sensor

applications, the interoperability challenges on IoT is far from being solved and a semantic IoT architecture is required to provide interoperability between connected IoT systems. This architecture should support multiple IoT protocols and severe resource and energy constrains.

In standard IoT applications the sink nodes are energy-constrained devices and utilizes minimum resources to conserve the energy. Various proposals seek to optimize the resources and provide translation between application layer protocol via the gateway devices[9][6]. These approaches fail in achieving interoperability at defining sensor annotation model, which is required to provide service level interoperability between IoT systems.

IoT interconnects physical world “Things” by utilizing software and networking technologies.

Due to its roots in traditional sensor networks, connected physical objects are resource-constrained devices, and require competent communication protocol for energy efficiency.

First wave of IoT application in smart city domain emphasized on connecting sensor interfacing with physical-world using lightweight protocols such as CoAP and XMPP [5][6]. In later stages, traditional Internet state transfer protocol such as REST is used for similar applications, where event-centric frameworks had been implemented to reduce number of messages transmitted [7].

The ‘Smart-Object’ devices with domain specific intelligence are rapidly replacing first wave of

IoT devices [8]. Although these devices do not utilize semantic technologies, they provide
higher-level of awareness from the sensor than just plain raw sensor data.
The IoT domain has been started getting congested with heterogeneous applications using different communication protocols and data models [9]. Various organizations such as Open IoT alliance, AllSeen alliance, and IPSO alliance are working on standardization of communication protocols to provide interoperability between various vendors silos [10][11][12]. Organization
such as Internet Engineering Task Force (IETF) and XMPP standards foundation are trying to
scale their messaging protocols, CoAP and XMPP, respectively, to align with other protocols.
These efforts are scattered and largely focus on solving problems around one protocol instead of providing integration solution.
In Web-centric infrastructure, acquisition of contextual information from raw sensor data requires annotation of sensor data with semantic metadata. Key standardization efforts that have sought to establish sensor data models for sensors to be accessible and controllable via the Web include:

- OGC Sensor Web Enablement (SWE)

The SWE efforts established by the Open Geospatial Consortium include following important specifications: Observation & Measurement (O&M), Sensor Model Language (SensorML) and Sensor Observation Service (SOS)[13]. The O&M and SensorML contain standard model and XML schema for observations/measurements and sensors/processes respectively. The SOS is a standard service model, which provides mechanism for querying observation and sensor metadata.

- Semantic Sensor Network (SSN) ontology

The SSN ontology, developed by W3C provides a standard for modeling sensor devices, sensor platforms, knowledge of the environment and observations[14] [15]. The SSN provides a foundation in the direction of achieving interoperability between the interconnected IoT Silos.

- Semantic Sensor Observation Service (SemSOS)

The Semantic Web enabled implementation of SOS, SemSOS, provides a rich semantic backend (knowledge base) while retaining the standard SOS specifications/service interactions. A semantically intelligent client can utilize this capability of SemSOS to derive higher level abstractions from the annotated sensor data [16] by implementing a semantic reasoning service acting on the knowledge base. SemSOS is the principal component of Semantic Sensor Web [17].
Although the utilization of these standards provide integration of Semantic Web with sensor
applications, the interoperability challenges on IoT is far from being solved and a semantic IoT architecture is required to provide interoperability between connected IoT systems. This architecture should support multiple IoT protocols and severe resource and energy constrains.
In standard IoT applications the sink nodes are energy-constrained devices and utilizes minimum resources to conserve the energy. Various proposals seek to optimize the resources and provide translation between application layer protocol via the gateway devices[9][6]. These approaches

fail in achieving interoperability at defining sensor annotation model, which is required to provide service level interoperability between IoT systems.

IoT interconnects physical world "Things" by utilizing software and networking technologies.

Due to its roots in traditional sensor networks, connected physical objects are resource-constrained devices, and require competent communication protocol for energy efficiency.

First wave of IoT application in smart city domain emphasized on connecting sensor interfacing with physical-world using lightweight protocols such as CoAP and XMPP [5][6]. In later stages, traditional Internet state transfer protocol such as REST is used for similar applications, where event-centric frameworks had been implemented to reduce number of messages transmitted [7].

The 'Smart-Object' devices with domain specific intelligence are rapidly replacing first wave of IoT devices [8]. Although these devices do not utilize semantic technologies, they provide

higher-level of awareness from the sensor than just plain raw sensor data.

The IoT domain has been started getting congested with heterogeneous applications using different communication protocols and data models [9]. Various organizations such as Open IoT alliance, AllSeen alliance, and IPSO alliance are working on standardization of communication protocols to provide interoperability between various vendors silos [10][11][12]. Organization

such as Internet Engineering Task Force (IETF) and XMPP standards foundation are trying to

scale their messaging protocols, CoAP and XMPP, respectively, to align with other protocols.

These efforts are scattered and largely focus on solving problems around one protocol instead of providing integration solution.

In Web-centric infrastructure, acquisition of contextual information from raw sensor data requires annotation of sensor data with semantic metadata. Key standardization efforts that have sought to establish sensor data models for sensors to be accessible and controllable via the Web include:

- OGC Sensor Web Enablement (SWE)

The SWE efforts established by the Open Geospatial Consortium include following important specifications: Observation & Measurement (O&M), Sensor Model Language (SensorML) and Sensor Observation Service (SOS)[13]. The O&M and SensorML contain standard model and XML schema for observations/measurements and sensors/processes respectively. The SOS is a standard service model, which provides mechanism for querying observation and sensor metadata.

- Semantic Sensor Network (SSN) ontology

The SSN ontology, developed by W3C provides a standard for modeling sensor devices, sensor platforms, knowledge of the environment and observations[14] [15]. The SSN provides a foundation in the direction of achieving interoperability between the interconnected IoT Silos.

- Semantic Sensor Observation Service (SemSOS)

The Semantic Web enabled implementation of SOS, SemSOS, provides a rich semantic backend (knowledge base) while retaining the standard SOS specifications/service interactions. A semantically intelligent client can utilize this capability of SemSOS to

derive higher level abstractions from the annotated sensor data [16] by implementing a semantic reasoning service acting on the knowledge base. SemSOS is the principal component of Semantic Sensor Web [17].

Although the utilization of these standards provide integration of Semantic Web with sensor applications, the interoperability challenges on IoT is far from being solved and a semantic IoT architecture is required to provide interoperability between connected IoT systems. This architecture should support multiple IoT protocols and severe resource and energy constrains.

In standard IoT applications the sink nodes are energy-constrained devices and utilizes minimum resources to conserve the energy. Various proposals seek to optimize the resources and provide translation between application layer protocol via the gateway devices[9][6]. These approaches fail in achieving interoperability at defining sensor annotation model, which is required to provide service level interoperability between IoT systems.

IoT interconnects physical world "Things" by utilizing software and networking technologies.

Due to its roots in traditional sensor networks, connected physical objects are resource-constrained devices, and require competent communication protocol for energy efficiency.

First wave of IoT application in smart city domain emphasized on connecting sensor interfacing with physical-world using lightweight protocols such as CoAP and XMPP [5][6]. In later stages, traditional Internet state transfer protocol such as REST is used for similar applications, where event-centric frameworks had been implemented to reduce number of messages transmitted [7].

The 'Smart-Object' devices with domain specific intelligence are rapidly replacing first wave of IoT devices [8]. Although these devices do not utilize semantic technologies, they provide higher-level of awareness from the sensor than just plain raw sensor data.

The IoT domain has been started getting congested with heterogeneous applications using different communication protocols and data models [9]. Various organizations such as Open IoT alliance, AllSeen alliance, and IPSO alliance are working on standardization of communication protocols to provide interoperability between various vendors silos [10][11][12]. Organization such as Internet Engineering Task Force (IETF) and XMPP standards foundation are trying to scale their messaging protocols, CoAP and XMPP, respectively, to align with other protocols.

These efforts are scattered and largely focus on solving problems around one protocol instead of providing integration solution.

In Web-centric infrastructure, acquisition of contextual information from raw sensor data requires annotation of sensor data with semantic metadata. Key standardization efforts that have sought to establish sensor data models for sensors to be accessible and controllable via the Web include:

- OGC Sensor Web Enablement (SWE)

The SWE efforts established by the Open Geospatial Consortium include following important specifications: Observation & Measurement (O&M), Sensor Model Language (SensorML) and Sensor Observation Service (SOS)[13]. The O&M and SensorML contain standard model and XML schema for observations/measurements and

sensors/processes respectively. The SOS is a standard service model, which provides mechanism for querying observation and sensor metadata.

- Semantic Sensor Network (SSN) ontology

The SSN ontology, developed by W3C provides a standard for modeling sensor devices, sensor platforms, knowledge of the environment and observations[14] [15]. The SSN provides a foundation in the direction of achieving interoperability between the interconnected IoT Silos.

- Semantic Sensor Observation Service (SemSOS)

The Semantic Web enabled implementation of SOS, SemSOS, provides a rich semantic backend (knowledge base) while retaining the standard SOS specifications/service interactions. A semantically intelligent client can utilize this capability of SemSOS to derive higher level abstractions from the annotated sensor data [16] by implementing a semantic reasoning service acting on the knowledge base. SemSOS is the principal component of Semantic Sensor Web [17].

Although the utilization of these standards provide integration of Semantic Web with sensor
applications, the interoperability challenges on IoT is far from being solved and a semantic IoT architecture is required to provide interoperability between connected IoT systems. This architecture should support multiple IoT protocols and severe resource and energy constrains.

In standard IoT applications the sink nodes are energy-constrained devices and utilizes minimum resources to conserve the energy. Various proposals seek to optimize the resources and provide translation between application layer protocol via the gateway devices[9][6]. These approaches fail in achieving interoperability at defining sensor annotation model, which is required to provide service level interoperability between IoT systems.

IoT interconnects physical world “Things” by utilizing software and networking technologies.

Due to its roots in traditional sensor networks, connected physical objects are resource-constrained devices, and require competent communication protocol for energy efficiency.

First wave of IoT application in smart city domain emphasized on connecting sensor interfacing with physical-world using lightweight protocols such as CoAP and XMPP [5][6]. In later stages, traditional Internet state transfer protocol such as REST is used for similar applications, where event-centric frameworks had been implemented to reduce number of messages transmitted [7].

The ‘Smart-Object’ devices with domain specific intelligence are rapidly replacing first wave of IoT devices [8]. Although these devices do not utilize semantic technologies, they provide
higher-level of awareness from the sensor than just plain raw sensor data.

The IoT domain has been started getting congested with heterogeneous applications using different communication protocols and data models [9]. Various organizations such as Open IoT alliance, AllSeen alliance, and IPSO alliance are working on standardization of communication protocols to provide interoperability between various vendors silos [10][11][12]. Organization
such as Internet Engineering Task Force (IETF) and XMPP standards foundation are trying to
scale their messaging protocols, CoAP and XMPP, respectively, to align with other protocols.

These efforts are scattered and largely focus on solving problems around one protocol instead of providing integration solution.

In Web-centric infrastructure, acquisition of contextual information from raw sensor data requires annotation of sensor data with semantic metadata. Key standardization efforts that have sought to establish sensor data models for sensors to be accessible and controllable via the Web include:

- OGC Sensor Web Enablement (SWE)

The SWE efforts established by the Open Geospatial Consortium include following important specifications: Observation & Measurement (O&M), Sensor Model Language (SensorML) and Sensor Observation Service (SOS)[13]. The O&M and SensorML contain standard model and XML schema for observations/measurements and sensors/processes respectively. The SOS is a standard service model, which provides mechanism for querying observation and sensor metadata.

- Semantic Sensor Network (SSN) ontology

The SSN ontology, developed by W3C provides a standard for modeling sensor devices, sensor platforms, knowledge of the environment and observations[14] [15]. The SSN provides a foundation in the direction of achieving interoperability between the interconnected IoT Silos.

- Semantic Sensor Observation Service (SemSOS)

The Semantic Web enabled implementation of SOS, SemSOS, provides a rich semantic backend (knowledge base) while retaining the standard SOS specifications/service interactions. A semantically intelligent client can utilize this capability of SemSOS to derive higher level abstractions from the annotated sensor data [16] by implementing a semantic reasoning service acting on the knowledge base. SemSOS is the principal component of Semantic Sensor Web [17].

Although the utilization of these standards provide integration of Semantic Web with sensor
applications, the interoperability challenges on IoT is far from being solved and a semantic IoT architecture is required to provide interoperability between connected IoT systems. This architecture should support multiple IoT protocols and severe resource and energy constrains.

In standard IoT applications the sink nodes are energy-constrained devices and utilizes minimum resources to conserve the energy. Various proposals seek to optimize the resources and provide translation between application layer protocol via the gateway devices[9][6]. These approaches fail in achieving interoperability at defining sensor annotation model, which is required to provide service level interoperability between IoT systems.

IoT interconnects physical world “Things” by utilizing software and networking technologies. Due to its roots in traditional sensor networks, connected physical objects are resource-constrained devices, and require competent communication protocol for energy efficiency. First wave of IoT application in smart city domain emphasized on connecting sensor interfacing with physical-world using lightweight protocols such as CoAP and XMPP .In later stages, traditional Internet state transfer protocol such as REST is used for similar applications, where event-centric frameworks had been implemented to reduce number of messages transmitted . The ‘Smart-Object’ devices with domain specific intelligence are rapidly replacing

first wave of IoT devices. Although these devices do not utilize semantic technologies, they provide higher-level of awareness from the sensor than just plain raw sensor data. The IoT domain has been started getting congested with heterogeneous applications using different communication protocols and data models . Various organizations such as Open IoT alliance, All Seen alliance, and IPSO alliance are working on standardization of communication protocols to provide interoperability between various vendors silos .Organization such as Internet Engineering Task Force (IETF) and XMPP standards foundation are trying to scale their messaging protocols, CoAP and XMPP, respectively, to align with other protocols. These efforts are scattered and largely focus on solving problems around one protocol instead of providing integration solution. In Web-centric infrastructure, acquisition of contextual information from raw sensor data requires annotation of sensor data with semantic metadata. Key standardization efforts that have sought to establish sensor data models for sensors to be accessible and controllable via the Web include: • OGC Sensor Web Enablement (SWE) The SWE efforts established by the Open Geospatial Consortium include following important specifications: Observation & Measurement (O&M), Sensor Model Language (SensorML) and Sensor Observation Service (SOS). The O&M and SensorML contain standard model and XML schema for observations/measurements and sensors/processes respectively. The SOS is a standard service model, which provides mechanism for querying observation and sensor metadata.

Semantic Sensor Network (SSN) ontology The SSN ontology, developed by W3C provides a standard for modeling sensor devices, sensor platforms, knowledge of the environment and observations. The SSN provides a foundation in the direction of achieving interoperability between the interconnected IoT Silos. • Semantic Sensor Observation Service (SemSOS) The Semantic Web enabled implementation of SOS, SemSOS, provides a rich semantic backend (knowledge base) while retaining the standard SOS specifications/service interactions. A semantically intelligent client can utilize this capability of SemSOS to derive higher level abstractions from the annotated sensor data by implementing a semantic reasoning service acting on the knowledge base. SemSOS is the principal component of Semantic Sensor Web. Although the utilization of these standards provide integration of Semantic Web with sensor applications, the interoperability challenges on IoT is far from being solved and a semantic IoT architecture is required to provide interoperability between connected IoT systems. This architecture should support multiple IoT protocols and severe resource and energy constrains. In standard IoT applications the sink nodes are energy-constrained devices and utilize minimum resources to conserve the energy. Various proposals seek to optimize the resources and provide translation between application layer protocol via the gateway devices. These approaches fail in achieving interoperability at defining sensor annotation model, which is required to provide service level interoperability between IoT systems.

CS-801

Internet Of Things

Experiment-8

Aim: Experiment on application framework and embedded software agents for IoT Toolkit.

The IoT Toolkit is a collection of libraries that support you in effortless communication with modern IoT devices while concentrating on the important parts of your application.

While IoT devices typically speak simple languages it can become a burden to actually implement these languages conform to their standards. The IoT toolkit provides proven libraries that help you surpass the difficulties when starting to access modern web based services. The

collection of libraries provides an easy to use API split into multiple theme based parts that can be used independent from each other and can be combined anytime necessary.

Using the IoT Toolkit jumpstarts the implementation of simple tasks like retrieving web pages using the HTTP client up to more complex tasks like communicating with REST based APIs by using the HTTP client together with the JSON parser. Popular services like Dropbox that provide their REST API to developers can be interfaced using this toolkit. Depending on the provided service additional components like a TLS stack such as emSSL might be necessary.

The libraries are optimized for use in embedded systems but are not limited to. PC based applications like emDropbox are possible too.

Key features

- Easy to use API to get started with IoT applications
- Supports REST API
- Supports handling plain received data
- Supports handling data encoding like JSON
- Small embedded optimized API
- Small module based libraries for a small footprint
- Memory efficient and resource effective handling of data

EXPERIMENT NO. 9

Aim: Exercise on working principle of Rasberry Pi. Application layer protocols:

CS-801

Internet Of Things

The Raspberry Pi is a Broadcom BCM2835 SOC (system on chip board). It comes equipped with a 700 MHz, 512 MB of SDRAM and ARM1176JZF-S core CPU. The USB 2.0 port of the raspberry pi boars uses only external data connectivity options. The Ethernet in the raspberry pi is the main gateway to interconnect with other devices and the internet in model B. This draws its power from a micro USB adapter, with a minimum range of 2.5 watts (500 MA). The graphics, specialized chip is designed to speed up the manipulation of image calculations. This is in built with Broadcom video core IV cable that is useful if you want to run a game and video through your raspberry pi.

Model B Raspberry pi Board

Features of Raspberry PI Model B

- 512 MB SDRAM memory
- Broadcom BCM2835 SoC full high definition multimedia processor
- Dual Core Video Core IV Multimedia coprocessor
- Single 2.0 USB connector
- HDMI (rev 1.3 and 1.4) Composite RCA (PAL & NTSC) Video Out
- 3.5 MM Jack, HDMI Audio Out
- MMC, SD, SDIO Card slot on board storage
- Linux Operating system
- Dimensions are 8.6cm*5.4cm*1.7cm

- On board 10/100 Ethernet RJ45 jack

To Set Up & Start your Raspberry Pi

The Raspberry Pi board comes equipped with an SD card. This slot permits us to insert an SD card and that can use it as our devices. The SD card is a main storage device for raspberry pi board like a hard disk of a personal computer. The bootable Linux operating system is loaded onto the card, you are planning to use. The raspberry pi supports Linux, Qtonpi, ARM, Mac operating systems. You can select one OS; you will need to write it to an SD card using a Disk manager application. You can also use other storage mechanism, like USB external hard drive or USB drive. There are a numerous brands of SD cards are available in the market in different sizes. The raspberry pi supports max 64 GB SD card.

Before you start your raspberry pi, you are going to need to connect a display, keyboard, mouse like as a PC. It supports three different O/Ps like HDMI video, composite video, and DSI video, where the DSI video needs some specific hardware. When you buy a raspberry pi board it may sold with or without an SD card. It is a very important specification in raspberry pi board. Because, it keeps its operating system, documents and programs. If your raspberry pi did not come with an SD card, then the min size you should get is 4GB. Advantages of the raspberry pi are, it is small in size, and it works as a normal computer at low cost server to handle web traffic.

Applications of Raspberry Pi

The raspberry pi boards are used in many applications like Media streamer, Arcade machine, Tablet computer, Home automation, Carputer, Internet radio, Controlling robots, Cosmic Computer, Hunting for meteorites, Coffee and also in raspberry pi based projects.

Raspberry Pi based Motor Speed Control

The main intention of this project is to control the speed of a DC Motor using Raspberry Pi.

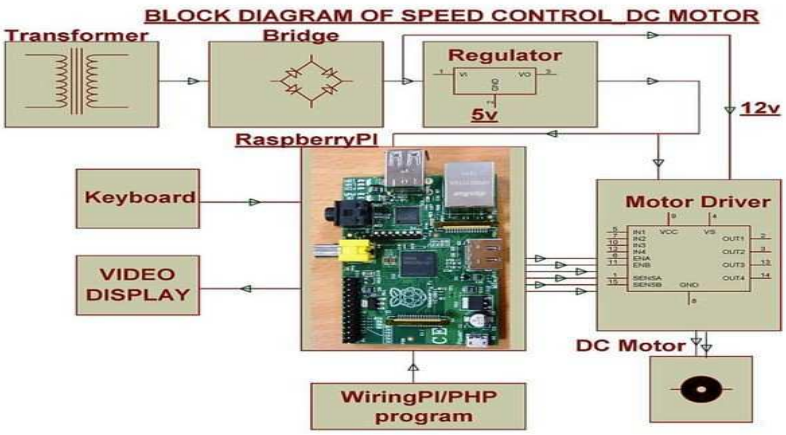
Hardware and software Requirements

DC Motor, Raspberry pi model, TV or PC monitor, Motor Driver IC, LED, Resistors, Capacitors, Diode, Transformer, Voltage Regulator and PHP program/wiring pi

CS-801

Internet Of Things

BLOCK DIAGRAM



Raspberry Pi based Motor Speed Control Block Diagram

This project uses a Raspberry Pi board to control the DC motor speed. The speed of a DC motor is directly proportional to the voltage applied across its terminals, when the voltage across the motor terminal is varied, then the speed also gets varied accordingly. So this is the main principle of this project. A keyboard is connected to the Raspberry pi board to run the motor at different speeds by pressing the key.

CS-801

Internet Of Things

EXPERIMENT NO. 10

Aim: Experiment on connectivity of Rasberry Pi with existing system components.

We designed LabNet as a distributed system (network) where LabNet presents a node running on a RasPi. We had two important requirements for this system: openness and scalability. Openness means that each node can control an experimental chamber on its own or together with a number of other nodes .Scalability means that there can be any number of nodes and thus experimental chambers in the system. However, a node or a chamber has to be removable from the system without adjustments on the other nodes. To ensure this, each node in our system is controlled by a RasPi, each RasPi is configured in the same way and controlled by the same software. However, this also comes with the restriction that at most one experimental system can be connected to each RasPi to be removable without electrical adjustments. But this also means a simplification: LabNet only needs to accept a single connection and does not need resource management for multiple connections, because only one experiment runs on one system and the hardware is not shared.

Thus, the network of LabNet nodes represents the distributed system and offers, as servers, the hardware resources in the network. However, hardware control in the context of the experiments is the responsibility of clients and not a LabNet duty. For example, LabNet does not decide about an output pin state, but LabNet knows how to switch the state and performs it at the client's request. One client could take the control over of the entire LabNet distributed system or divide the nodes among several clients.

It all depends on the situation and requirements: a large number of identical experimental chambers with identical experimental tasks are usually controlled by one client while different experimental tasks may better be controlled by separate clients, also to start and stop experiments independently. For communication between LabNet and client a flexible and fast message protocol using Protobuf was developed (section Message protocol). The clients can be implemented in any language with Protobuf support, for example, Python, C#, C++, etc.

Since the Raspberry Pi is a single-board computer, it runs 'Raspberry Pi OS': a Debian-based Linux distribution. This allows a large freedom in the choice of programming language and software tools. Both interpreted languages, such as Python, and compiled languages are available. LabNet was required to meet two criteria:

1. Time-critical: all operations should be performed as quickly as possible.
2. Flexible: new functionality extensions should be as simple as possible.

Unfortunately, all interpreted programming languages have a disadvantage in execution speed compared to compiled languages. Nevertheless, many of the tools developed recently, such as Autopilot and pyControl, use Python. Python is a simple language and provides many packages for all purposes. However, because execution speed was of primary importance we decided to use C++.

Extensions with new functionalities are generally possible in two ways: (i) software adaption with recompilation in case of compiled languages and (ii) a plug-in system. In the current LabNet version, we use recompilation but our road map also includes a future plug-in system. To simply modifications the software must have a suitable architecture and a high degree of modularization.

Since its version 2 the Raspberry Pi has 4 cores. In addition, most of its hardware controllers, such as USB or Ethernet, operate asynchronously, thus they do not require CPU capacity because of DMA (Direct Memory Access), and they report their work completion via interrupt messages. LabNet needed an architecture that optimally leverages this already available hardware asynchrony for parallel execution. Handling GPIO lines is fast, but accessing a UART may lead to considerable delays in sequentially executed software. This presupposes the use of multiple threads. Since programming with many parallel threads is a very error-prone and time-consuming task, we decided to develop an actor-based software (see sections Actor model and SObjectizer). This also provides higher flexibility and software modularity.

Example

The following example and the corresponding listings (1–3) show how a client can initialize and control the hardware together with a LabNet server on a RasPi with a simple hardware setup. The client could run on a PC and use any language that has support for Protobuf-like Python, C++, C#, etc. Since we use C# in our experiments, the C# notation is also used in the listings. Basically, it shows the use of some of the LabNet messages, but the communication via TCP/IP is omitted for simplicity. We simply assume that a TCP/IP client exists and handles all operations like send, receive, and serialization.

Let us assume an experimental setup with an LED and audio as stimuli, a valve to release a liquid reward and a photo gate as a nose-poke sensor to detect animal behaviour. All these components can be connected directly to the GPIO pins via a simple circuit. The headphone jack can be used for audio output. Then, we need to send five commands to LabNet to initialize all components; see Listing 1. It would usually be necessary for the client to wait for the responses from LabNet and check the initialization results. Here, we skip this step.

CS-801

Internet Of Things

During experiments, animals must usually perform some operant behavior. This can be as simple as nose poking to trigger a photo gate after a certain stimulus has been perceived. In Listing 2, LabNet activates an LED and produces a sine tone. In the case of the tone, it is instructed to automatically generate a pulsed output. On detecting On and Off state changes, LabNet transmits such photo gate state changes to the client. In response to the photo gate state change, a reward can be provided. In Listing 3, a liquid reward valve is opened for 100 ms.

A typical experiment in combination with LabNet comprises several phases:

- establishing a TCP/IP connection;
- initializing all hardware components;
- turning stimuli on or off in a specific order;
- waiting for an animal reaction and potentially providing a reward.

Performance evaluation

Because the neurons in the brain work in the millisecond range, the response times in behavioral experiments are critical and should match that range.

```
// start GPIO interface with WiringPi
var initIo = new GpioWiringPiInit();
// init a digital output on pin 5
var led = new GpioWiringPiInitDigitalOut {
        Pin = 5,
       IsInverted = false
};
// init a digital output on pin 26
var valve = new GpioWiringPiInitDigitalOut {
        Pin = 26,
        IsInverted = false
};
// init a digital input on pin 23
var poke = new GpioWiringPiInitDigitalIn {
        Pin = 23,
        IsInverted = false,
        ResistorState = PullUp
};

// start sound interface
var initSound = new InitSound();
```

```
// create a sine tone
var sine = new DefineSineTone {
        Id = 1,
        Frequenz = 1000,
        Volume = 0.5
};
```

Listing 1

Each generated object represents an initialization message to be serialized with Protobuf and transmitted to LabNet. The first message initializes the digital I/O interface with WiringPi pin notation. The next three initialize an LED, a valve, and a poke sensor on the WiringPi interface. The fifth creates the sound generator on the headphone jack. The last, initializes a sine tone with 1 kH frequency and 50% volume. Object initialization in C# notation. Serialization and TCP/IP communication not listed.

```
// change the state of the pin 5 to true
var setLed = new DigitalOutSet {
        Id = new PinId { Interface = GpioWiringpi, Pin = 5 },
        State = true
};
// turn the sine tone in pulses of 500ms on and off
var pulseSound = new DigitalOutPulse {
        Id = new PinId { Interface = Sound, Pin = 1 },
        HighDuration = 500, // ms
        LowDuration = 500, // ms
        Pulses =10
};
```

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & SCIENCE, BHOPAL

CS-801

Internet Of Things