SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)
COIMBATORE-35

Department of Information Technology

19ITT302 – Internet of Things

UNIT – 5 DESIGN METHODOLOGY & FUTURE TRENDS

Need for IoT Systems Management

IoT can have complex hardware, software and deployment designs including sensors, actuators, software and network resources, data collection and analysis services and interfaces. Managing multiple devices within a single system requires advanced management capabilities.

The need for managing IoT systems in described as follows,

- Automating Configuration
- Monitoring Operational & Statistical Data
- Improved Reliability
- System Wide Configurations
- Multiple System Configurations
- Retrieving & Reusing Configurations

Automating Configuration
System management interfaces provide predictable and easy to use management capability and the ability to automate system configuration. Automation becomes even more important when a system consist of multiple devices or nodes. In such cases automating system configuration ensures that all devices have the same configuration and variations or errors due to manual configurations are avoided.

Monitoring Operational & Statistical Data
Operational data is the data which is related to systems operating parameter and collected by system at runtime. Statistical data is the data which describes systems performance (E.G CPU and Memory usage). Management system can help in monitoring operational and statistical data of a system.

Improved Reliability
A management system that allows validating the system configuration before they are put into effect can help in improving the system reliability.

System Wide Configurations
For IoT system that consist of multiple devices or nodes, ensuring system wide configuration can be crucial for the correct functioning of the system. Management approaches in which each

device is configured separately can result in system faults. This happens when some system uses new configuration whereas others use old configuration. To avoid this system wide configuration is essential to have same configuration across multiple devices using single atomic transactions. This ensures configuration changes are applied to all system or none.

Multiple System Configurations
For system it may be desirable to have multiple valid configurations which are applied at different times or in certain conditions

Retrieving & Reusing Configurations
Management systems which have the capability of retrieving configurations from the devices can help in reusing the configurations for other devices of the same type.

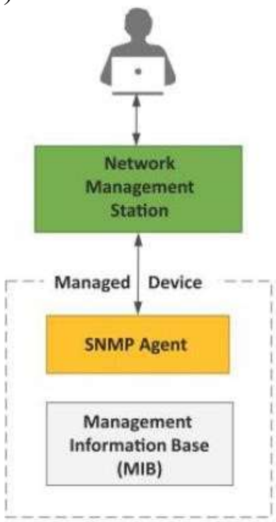
Simple Network Management Protocol (SNMP)

SNMP is a well-known and widely used network management protocol that allows monitoring and configuring network devices such as routers, switches, servers, printers, etc.

SNMP is an application layer protocol that uses User Datagram Protocol as the transport protocol on ports 161/162.

SNMP component include
- Network Management Station (NMS) or SNMP Manager

Managed Devices
- SNMP Agent that runs on the device
- Management Information Base (MIB)



Network Management Station (NMS)

NMS executes SNMP commands to monitor and configure the managed device.

Managed Device

It is the device that is being managed and runs a software called SNMP Agent.

SNMP Agent

It is a management software module installed on a managed device. Managed devices can be network devices like PC, routers, switches, servers, etc. The managed device contains the MIB which has all the information of the device attributes to be managed.

Management Information Base (MIB)

MIBs use the structure of the management information notation for defining the structure of the management data. The structure of the management data is defined in the form of variables which are identified by object identifiers (OIDs) which have a hierarchical structure. Each agent has its own MIB, which is a collection of all the objects that the manager can manage. MIB is categorized into eight groups: system, interface, address translation, ip, icmp, tcp, udp, and egp. These groups are under the mib object.

How SNMP Works?

- The SNMP manager is a host that acts as the client and runs client program, the SNMP agent acts as the server and the MIB acts as the server's database.
- When the SNMP manager asks the agent a question, the agent uses the MIB to supply the answer.
- The agent is used to keep the information in a database while the manager is used to access the values in the database.
- SNMP software agents on network devices and services communicate with a network management system to relay status information and configuration changes.
- The NMS provides a single interface from which administrators can issue batch commands and receive automatic alerts.
- SNMP uses a blend of pull and push communications between network devices and the network management system.
- The SNMP agent, which resides with the MIB on a network device, constantly collects status information but will only push information to the NMS upon request or when some aspect of the network crosses a pre-defined threshold known as a trap.
- Trap messages are typically sent to the management server when something significant, such as a serious error condition, occurs.

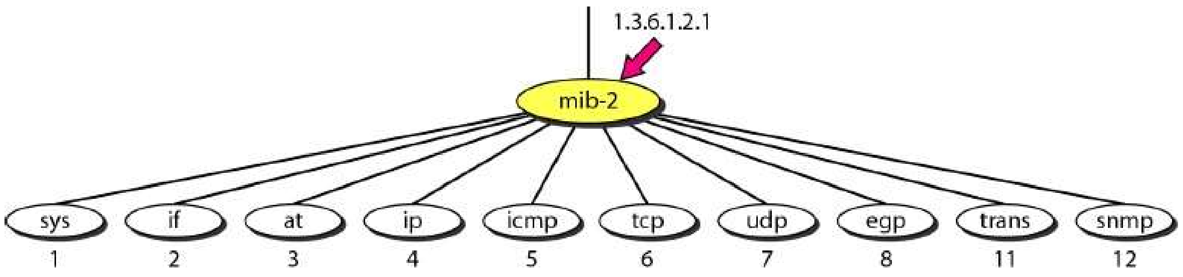
Message Types

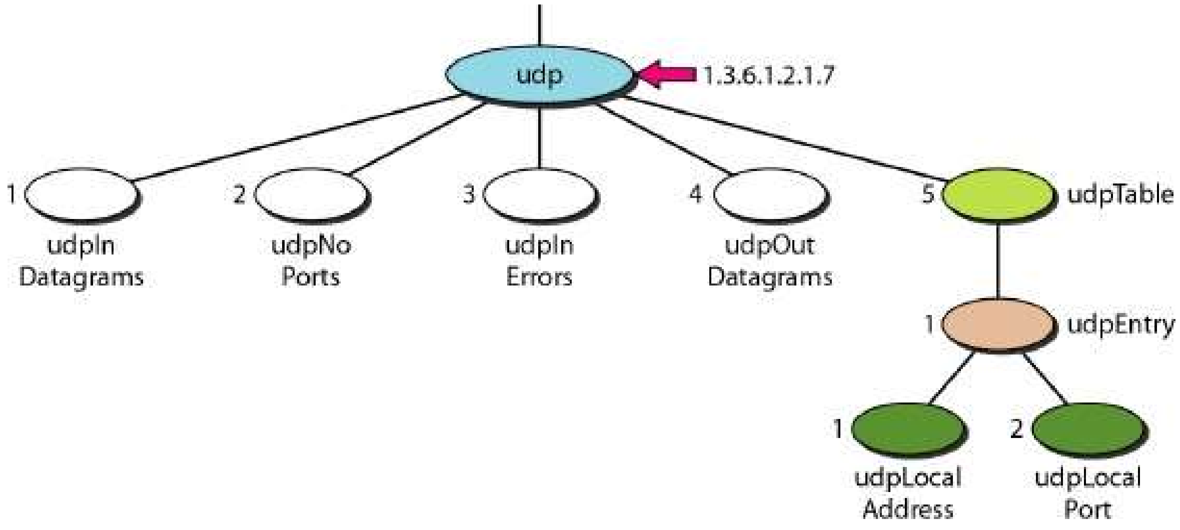
SNMP defines five types of messages: GetRequest, GetNextRequest, SetRequest, GetResponse, and Trap.

GetRequest: The GetRequest message is sent from a manager (client) to the agent (server) to retrieve the value of a variable.

GetNextRequest: The GetNextRequest message is sent from the manager to agent to retrieve

the value of a variable. This type of message is used to retrieve the values of the entries in a table.
GETBULK Request: Sent by the SNMP manager to the agent to efficiently obtain a potentially large amount of data, especially large tables. It is introduced in SNMPv2c.
SetRequest: The SetRequest message is sent from a manager to the agent to set a value in a variable.
RESPONSE: Sent by the agent to the SNMP manager, issued in reply to a GET Request, GETNEXT Request, GETBULK Request and a SET Request. Contains the values of the requested variables.
Trap –
These are the message sent by the agent without being requested by the manager. It is sent when a fault has occurred.
InformRequest –
It was introduced in SNMPv2c, used to identify if the trap message has been received by the manager or not. The agents can be configured to send trap message continuously until it receives an Inform message. It is the same as a trap but adds an acknowledgement that the trap doesn't provide.





Limitations of SNMP Protocol

- SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device.

- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.
- MIBs often lack writable objects without which device configuration is not possible using SNMP. With the absence of writable objects, SNMP can be used only for device monitoring and status polling.
- It is difficult to differentiate between configuration and state data in MIBs.
- Retrieving the current configuration from a device can be difficult with SNMP.
- Earlier versions of SNMP did not have strong security features making the management information vulnerable to network intruders. Though security features were added in the later versions of SNMP, it increased the complexity a lot.

Network Operator Requirements

Ease of use
From the operators point of view ease of use is a key requirement for any network management technology.

Distinction between configuration and state data
Configuration data is set of writable data that is required to transform the system from its initial state to its current state. State data is data which is not configurable. For effective management solution it is important to make clear distinction between configuration data and state data.

Fetch configuration and state data separately
It should be possible to fetch the configuration data and state data separately from the managed device. This is useful when the configuration and state date from different from different devices needs to be compared.

Configuration of the network as a whole
It should be possible for operators to configure the network as a whole rather than individual devices. This is important for systems which have multiple devices and configuring them within one network wide transaction is required to ensure the correct operation of the system.

Configuration transactions across devices
Configuration transactions across multiple devices should be supported.

Configuration deltas
It should be possible to generate the operations necessary for going from one configuration state to another. The device should support configuration deltas with minimal state changes.

Dump and restore configurations
It should be possible to Dump configurations from the devices and restore the configuration to the devices.

Configuration validation
It should be possible to validate the configurations

Configuration database schemas
There is a need for standardized Configuration database schemas or data models across operators.

Comparing configurations
Devices should not arbitrarily recorder data so that it is possible to use text processing tools such as diff to compare configurations.

Role-based access control
Devices should support role based access control model so that the user is given minimum access necessary to perform the required task.

Consistency of access control lists:
It should be possible to do Consistency checks of access control lists across devices.

Multiple configuration sets
There should be support for multiple configurations sets on devices. This way a distinction can be provided between candidate and active configurations.

Support for both data-oriented and task oriented access control
While SNMP access control data is data oriented, CLI access control is usually task oriented. There should be support for both types of access control.

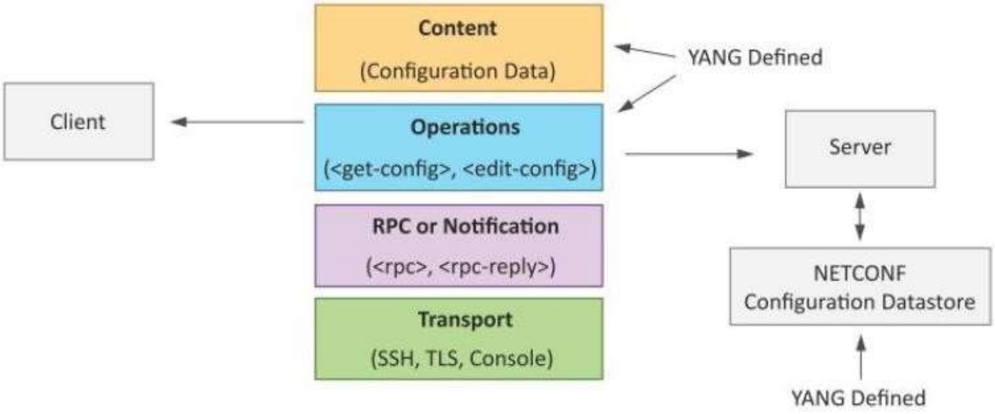
NETCONF

Network Configuration Protocol (NETCONF) is a session-based network management protocol. NETCONF allows retrieving state or configuration data and manipulating configuration data on network devices.

NETCONF works on SSH (Secure Shell) transport protocol. In addition to SSH protocol, NETCONF implementations can support other transport mappings such as Blocks Extensible Exchange Protocol (BEEP).

Transport layer provides end-to-end connectivity and ensure reliable delivery of messages. NETCONF uses XML-encoded Remote Procedure Calls (RPCs) for framing request and response messages.

The RPC layer provides mechanism for encoding of RPC calls and notifications. NETCONF provides various operations to retrieve and edit configuration data from network devices.



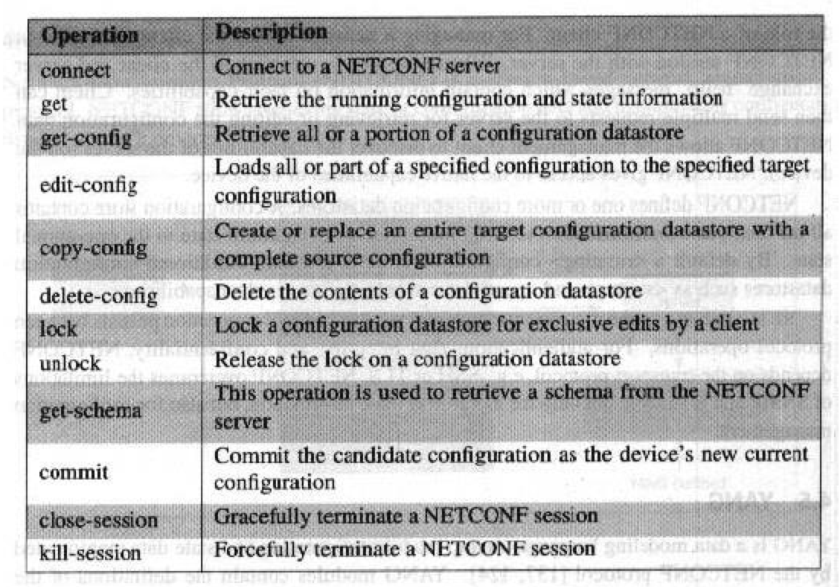
NETCONF Protocol Layers

The Content Layer consists of configuration and state data which is XML-encoded.

The schema of the configuration and state data is defined in a data modeling language called YANG.

NETCONF provides a clear separation of the configuration and state data. For example <get-config> retrieves the configuration data only while the operation <get> retrieves the configuration and state data.

The configuration data resides within a NETCONF configuration datastore on the server. The NETCONF server resides on the network device. The management application plays the role of NETCONF client.



Operation	Description
connect	Connect to a NETCONF server
get	Retrieve the running configuration and state information
get-config	Retrieve all or a portion of a configuration datastore
edit-config	Loads all or part of a specified configuration to the specified target configuration
copy-config	Create or replace an entire target configuration datastore with a complete source configuration
delete-config	Delete the contents of a configuration datastore
lock	Lock a configuration datastore for exclusive edits by a client
unlock	Release the lock on a configuration datastore
get-schema	This operation is used to retrieve a schema from the NETCONF server
commit	Commit the candidate configuration as the device's new current configuration
close-session	Gracefully terminate a NETCONF session
kill-session	Forcefully terminate a NETCONF session

Table 4.1: List of commonly used NETCONF RPC methods

For managing the network devices the client establishes a NETCONF session with the server.

When a session is established the client and server exchanges hello messages which contains information about their capabilities. Client can send multiple requests to the server for retrieving or editing configuration data.

NETCONF defines one or more configuration data stores. A configuration store contains all the configuration information to bring the device from its initial state to the operational state.

NETCONF is a connection oriented protocol and NETCONF connection persists between protocol operations. For authentication, data integrity, and confidentiality NETCONF depends on transport protocol such as SSH or TLS.

NETCONF overcomes the limitations of SNMP and is suitable not only for monitoring state information but also for configuration management.

YANG

YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol

YANG modules contain the definitions of the configuration data, state data, RPC calls that can be issued and the format of the notifications. YANG modules defines the data exchanged between the NETCONF client and server.

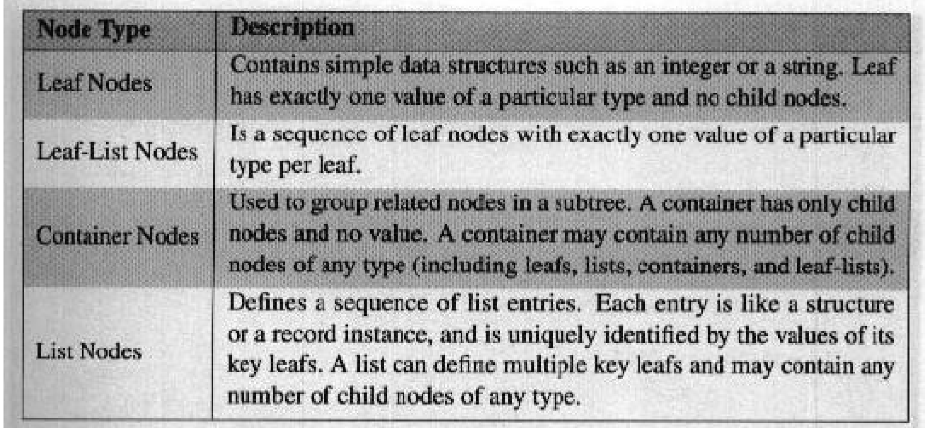
YANG models data using a hierarchical, tree-based structure with nodes. YANG defines four nodes types. Each node has a name, and depending on the node type, the node might either define a value or contain a set of child nodes.

A module comprises of a number of 'leaf' nodes which are organized into a hierarchical tree structure. The 'leaf' nodes are specified using the 'leaf' or 'leaf-list' constructs.

Leaf nodes are organized using 'container' or 'list' constructs. A YANG module can import definitions from other modules.

By default, a node defines configuration data. A node defines state data if it is tagged as config false. Configuration data is returned using the NETCONF <get-config> operation, and state data is returned using the NETCONF <get> operation.

Constraints can be defined on the data nodes, e.g. allowed values. YANG can model both configuration data and state data using the 'config' statement.



Node Type	Description
Leaf Nodes	Contains simple data structures such as an integer or a string. Leaf has exactly one value of a particular type and no child nodes.
Leaf-List Nodes	Is a sequence of leaf nodes with exactly one value of a particular type per leaf.
Container Nodes	Used to group related nodes in a subtree. A container has only child nodes and no value. A container may contain any number of child nodes of any type (including leafs, lists, containers, and leaf-lists).
List Nodes	Defines a sequence of list entries. Each entry is like a structure or a record instance, and is uniquely identified by the values of its key leafs. A list can define multiple key leafs and may contain any number of child nodes of any type.

Table 4.2: YANG Node Types

• Constraints can be defined on the data nodes, e.g. allowed values. YANG can model both configuration data and state data using the 'config' statement.

IoT Systems Management with NETCONF-YANG

This section describes how IoT to manage IoT systems with NETCONG and YANG.
Components and Roles

- Management System
- Management API
- Transaction Manager
- Rollback Manager
- Data Model Manager
- Configuration Validator
- Configuration Database
- Configuration API
- Data Provider API

Management System
The operator uses a Management system to send NETCONF messages to configure the IoT device and receives state information and notifications from the device as NETCONF messages.

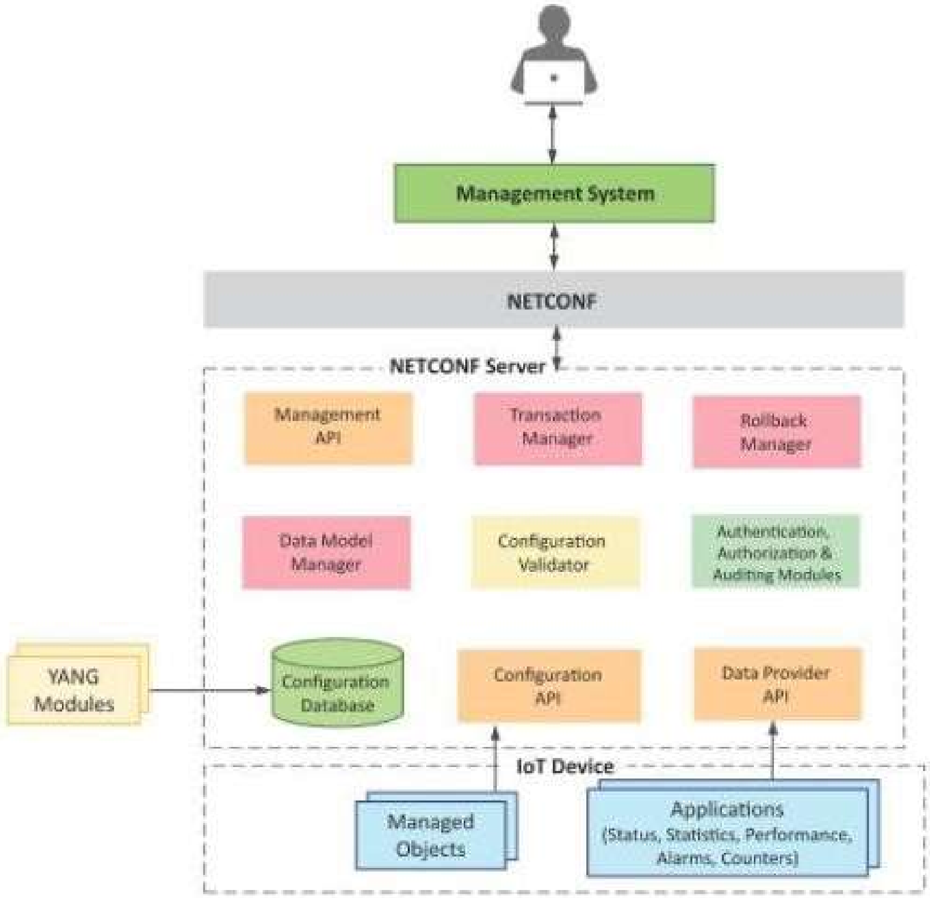
Management API
Management API allows management applications to start NETCONF sessions, read and write configuration data, read state data, retrieve configuration data and invoke RPC, in the same way as operators.

Transaction Manager
Transaction Manager executes all the NETCONF transactions and ensures that the ACID (Atomicity, Consistency, Isolation, Durability properties hold true for the transactions.

Rollback Manager
Rollback manager is responsible for generating all the transactions necessary to roll back a current configuration to its original state.

Data Model Manager
Data Model Manager keeps tracks of all the YANG data models and the corresponding managed objects. The Data model manager also keeps track of the applications which provide data for each part of the data model.

Configuration Validator
Configuration Validator checks if the resulting configuration after applying a transaction would be valid configuration.



Configuration Database
This database contains both the configuration and operational data.

Configuration API
Using the configuration API the applications on the IoT device can read configuration data from the configuration datastore and write operational data to the operational datastore.

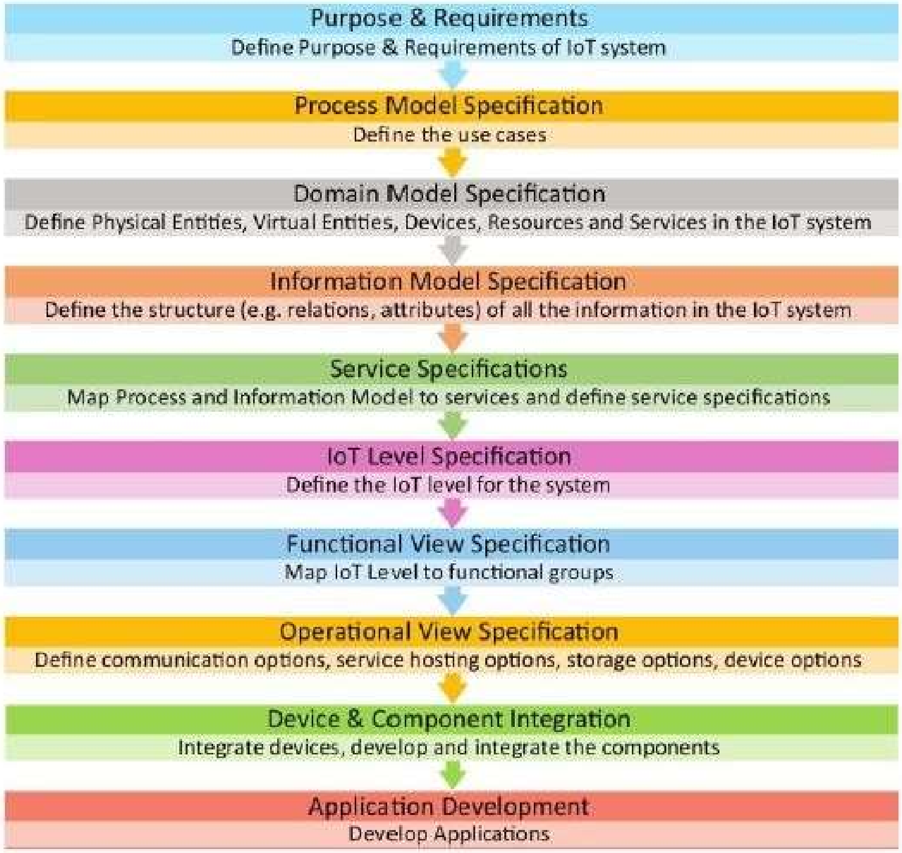
Data Provider API

Applications on the IoT Devices can register for callbacks for various events using the Data provide API. Through the Data provide API, the applications can report statistics and operational data.

IoT Platforms Design Methodology

IoT Design Methodology includes the following steps:

- Purpose & Requirements Specification
- Process Specification
- Domain Model Specification
- Information Model Specification
- Service Specifications
- IoT Level Specification
- Functional View Specification
- Operational View Specification
- Device & Component Integration
- Application Development


Purpose & Requirements
Define Purpose & Requirements of IoT system
Process Model Specification
Define the use cases
Domain Model Specification
Define Physical Entities, Virtual Entities, Devices, Resources and Services in the IoT system
Information Model Specification
Define the structure (e.g. relations, attributes) of all the information in the IoT system
Service Specifications
Map Process and Information Model to services and define service specifications
IoT Level Specification
Define the IoT level for the system
Functional View Specification
Map IoT Level to functional groups
Operational View Specification
Define communication options, service hosting options, storage options, device options
Device & Component Integration
Integrate devices, develop and integrate the components
Application Development
Develop Applications

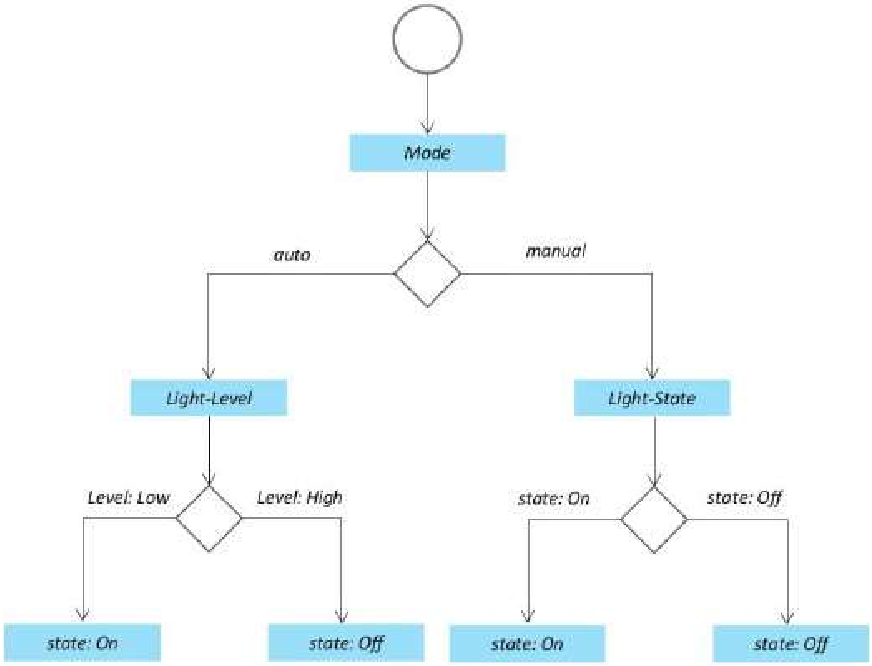
Step 1: Purpose & Requirements Specification

The first step in IoT system design methodology is to define the purpose and requirements of the system. In this step, the system purpose, behavior and requirements (such as data collection requirements, data analysis requirements, system management requirements, data privacy and security requirements, user interface requirements, ...) are captured. In this home automation taken as example

- Purpose : A home automation system that allows controlling of the lights in a home remotely using a web application.
- Behavior : The home automation system should have auto and manual modes. In auto mode, the system measures the light level in the room and switches on the light when it gets dark. In manual mode, the system provides the option of manually and remotely switching on/off the light.
- System Management Requirement : The system should provide remote monitoring and control functions.
- Data Analysis Requirement : The system should perform local analysis of the data.
- Application Deployment Requirement : The application should be deployed locally on the device, but should be accessible remotely.
- Security Requirement : The system should have basic user authentication capability

Step 2: Process Specification

- The second step in the IoT design methodology is to define the process specification. In this step, the use cases of the IoT system are formally described based on and derived from the purpose and requirement specifications.



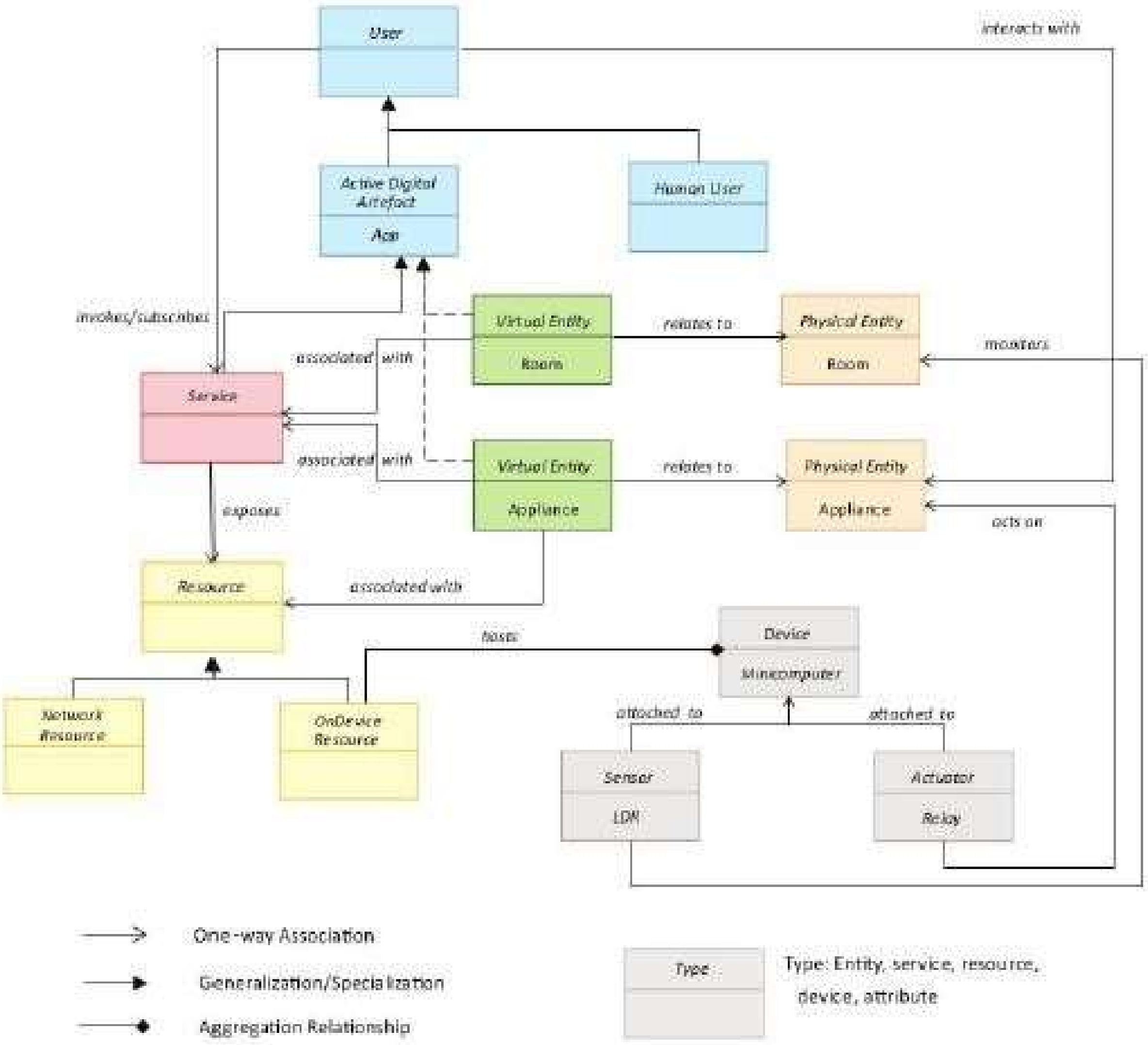
Step 3: Domain Model Specification

- The third step in the IoT design methodology is to define the Domain Model.
- The domain model describes the main concepts, entities and objects in the domain of IoT system to be designed.
- Domain model defines the attributes of the objects and relationships between objects.
- Domain model provides an abstract representation of the concepts, objects and entities in the IoT domain, independent of any specific technology or platform. With the domain

model, the IoT system designers can get an understanding of the IoT domain for which the system is to be designed.

The entities, objects and concepts defined in the domain model include

- Physical Entity
- Virtual Entity
- Device
- Resource
- Service

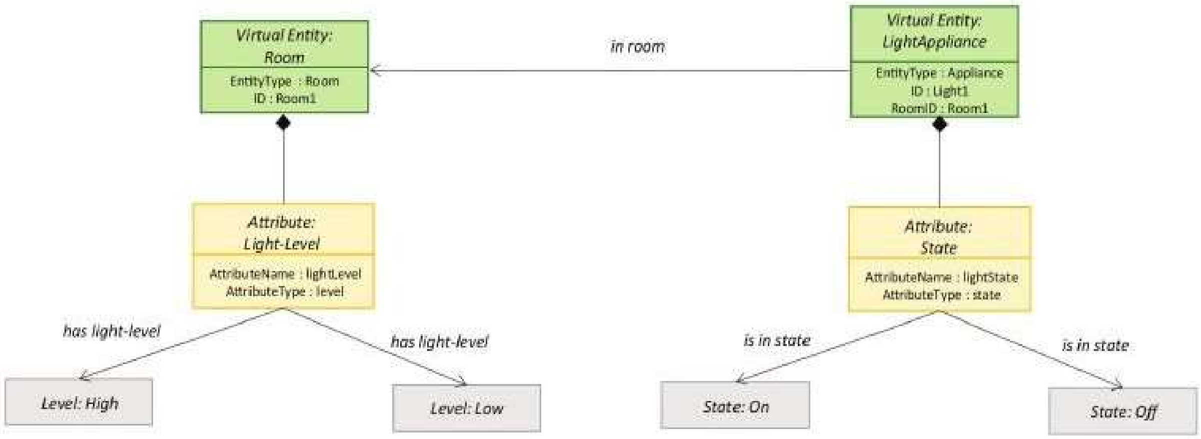


Step 4: Information Model Specification

• The fourth step in the IoT design methodology is to define the Information Model.

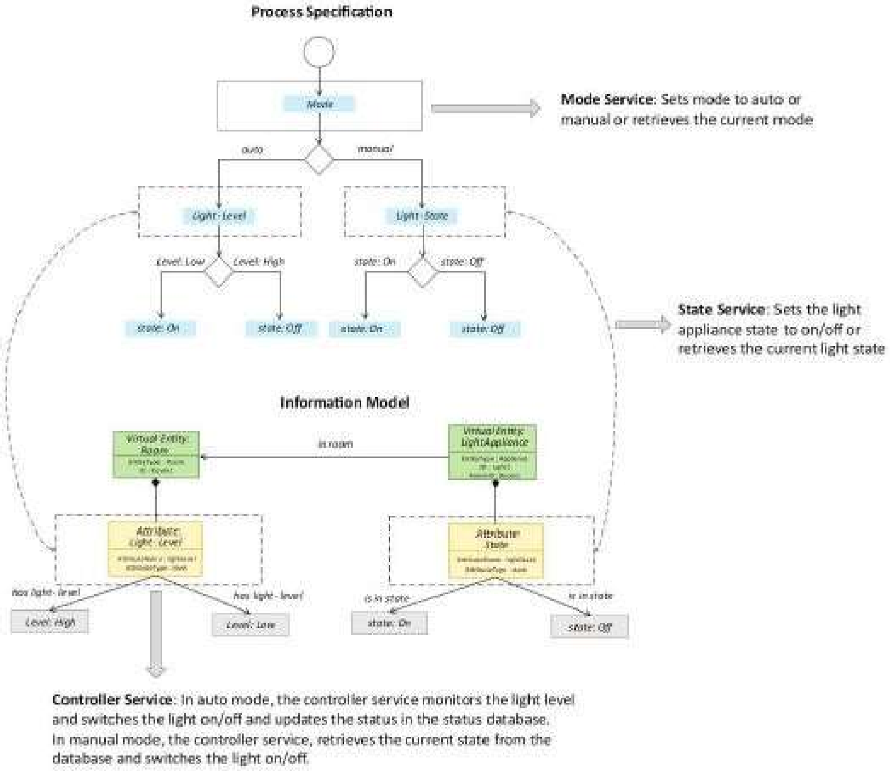
- Information Model defines the structure of all the information in the IoT system, for example, attributes of Virtual Entities, relations, etc. Information model does not describe the specifics of how the information is represented or stored.
- To define the information model, we first list the Virtual Entities defined in the Domain Model.

Information model adds more details to the Virtual Entities by defining their attributes and relations.



Step 5: Service Specifications

The fifth step in the IoT design methodology is to define the service specifications. Service specifications define the services in the IoT system, service types, service inputs/output, service endpoints, service schedules, service preconditions and service effects.



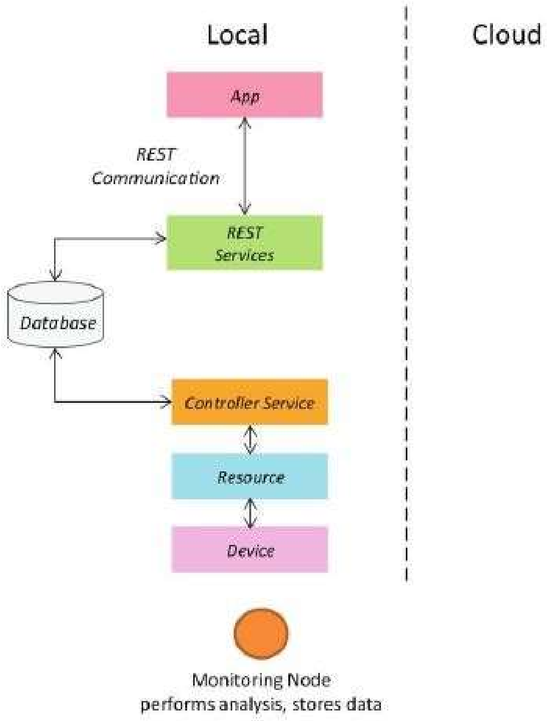
Mode Service: Sets mode to auto or manual or retrieves the current mode

State Service: Sets the light appliance state to on/off or retrieves the current light state

Controller Service: In auto mode, the controller service monitors the light level and switches the light on/off and updates the status in the status database. In manual mode, the controller service, retrieves the current state from the database and switches the light on/off.

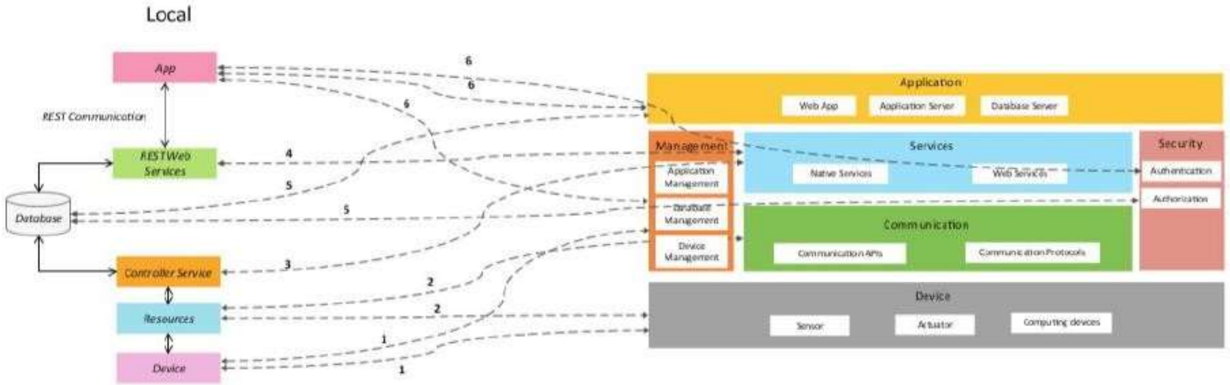
Step 6: IoT Level Specification

- The sixth step in the IoT design methodology is to define the IoT level for the system. In Chapter-1, we defined five IoT deployment levels



Step 7: Functional View Specification

- The seventh step in the IoT design methodology is to define the Functional View. The Functional View (FV) defines the functions of the IoT systems grouped into various Functional Groups (FGs).
- Each Functional Group either provides functionalities for interacting with instances of concepts defined in the Domain Model or provides information related to these concepts.



1. IoT device maps to the Device FG (sensors, actuators devices, computing devices) and the Management FG (device management)

2. Resources map to the Device FG (on-device resource) and Communication FG (communication APIs and protocols)

3. Controller service maps to the Services FG (native service). Web Services map to Services FG (web services)

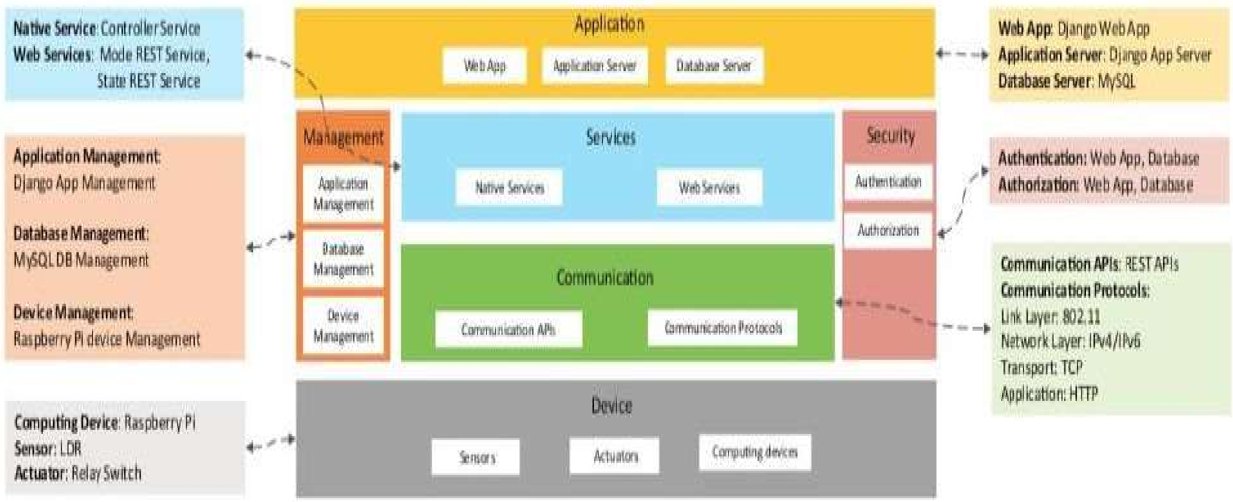
4. Web Services map to Services FG (web services)

5. Database maps to the Management FG (database management) and Security FG (database security)

6. Application maps to the Application FG (web application, application and database servers), Management FG (app management) and Security FG (app security)

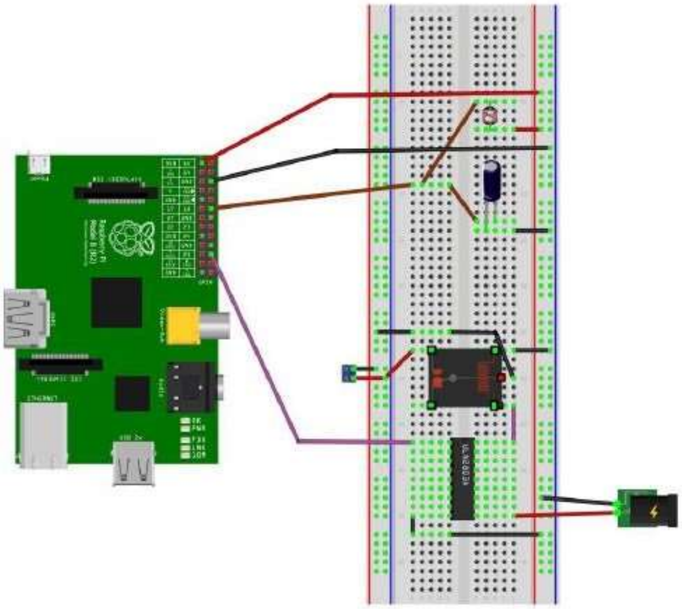
Step 8: Operational View Specification

The eighth step in the IoT design methodology is to define the Operational View Specifications. In this step, various options pertaining to the IoT system deployment and operation are defined, such as, service hosting options, storage options, device options, application hosting options, etc



Step 9: Device & Component Integration

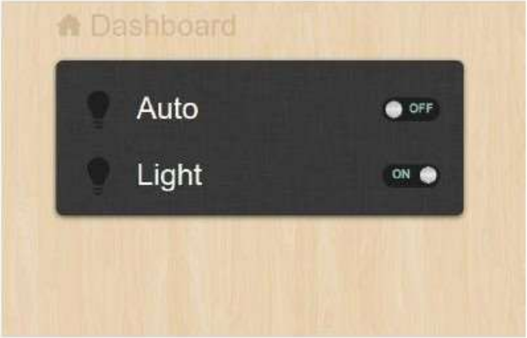
- The ninth step in the IoT design methodology is the integration of the devices and components.



Step 10: Application Development

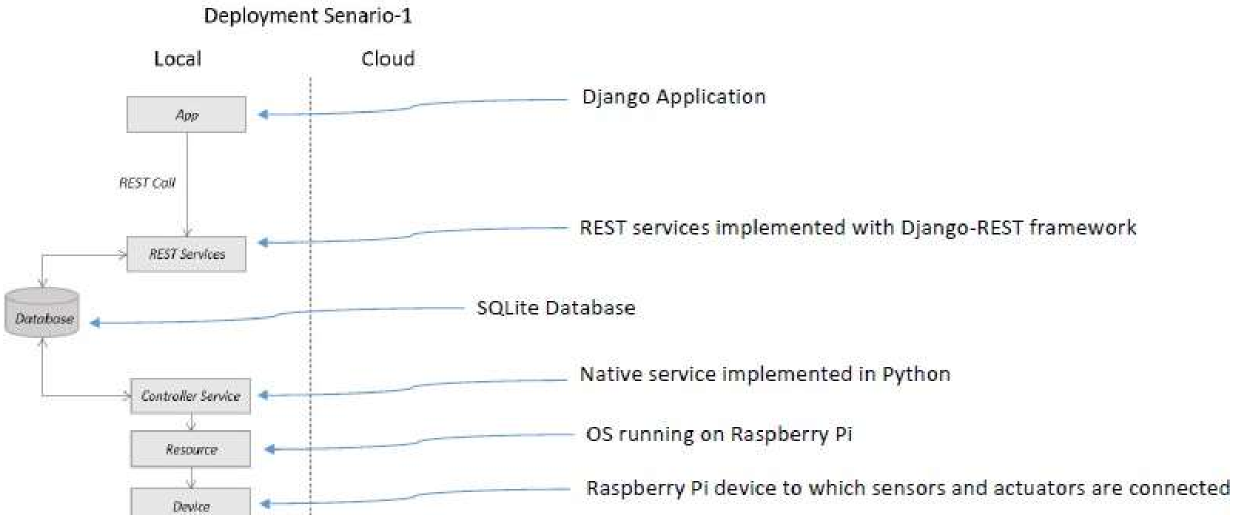
• The final step in the IoT design methodology is to develop the IoT application..

- Auto
  - Controls the light appliance automatically based on the lighting conditions in the room
- Light
  - When Auto mode is off, it is used for manually controlling the light appliance.
  - When Auto mode is on, it reflects the current state of the light appliance.



Finally - Integrate the System

- Setup the device
- Deploy and run the REST and Native services
- Deploy and run the Application
- Setup the database


Deployment Senario-1
Local
Cloud
App
REST Call
REST Services
Database
Controller Service
Resource
Device
Django Application
REST services implemented with Django-REST framework
SQLite Database
Native service implemented in Python
OS running on Raspberry Pi
Raspberry Pi device to which sensors and actuators are connected

IoT Physical Devices & Endpoints

What is an IoT Device

A "Thing" in Internet of Things (IoT) can be any object that has a unique identifier and which can send/receive data (including user data) over a network (e.g., smart phone, smart TV, computer, refrigerator, car, etc.).

IoT devices are connected to the Internet and send information about themselves or about their surroundings (e.g. information sensed by the connected sensors) over a network (to other devices or servers/storage) or allow actuation upon the physical entities/environment around them remotely.

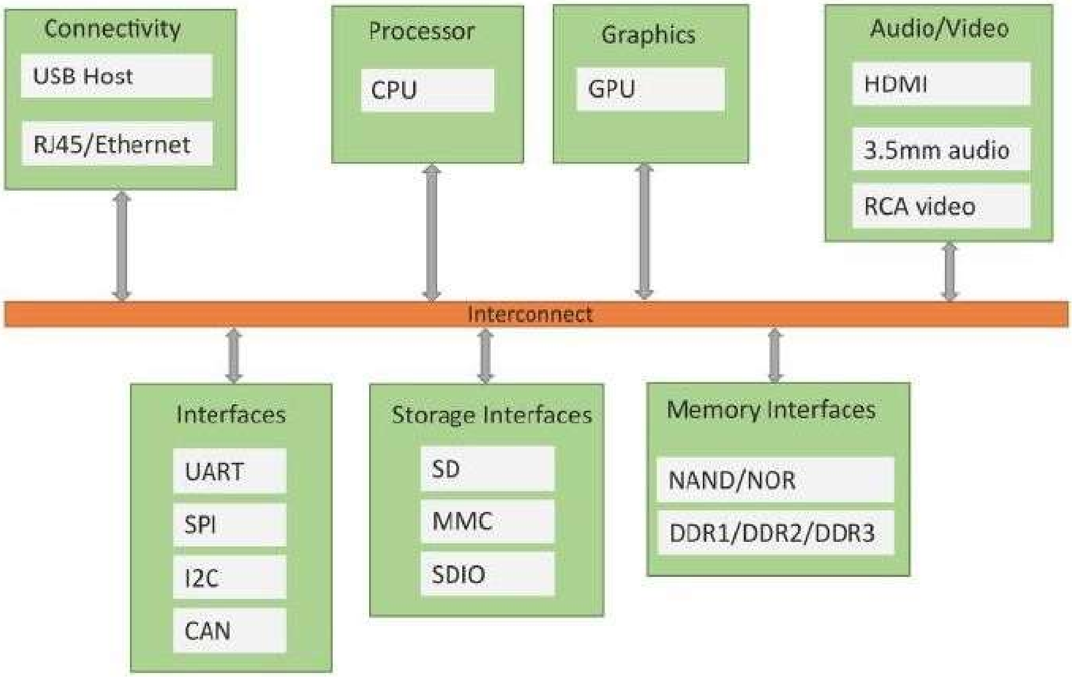
IoT Device Examples

- A home automation device that allows remotely monitoring the status of appliances and controlling the appliances.
- An industrial machine which sends information about its operation and health monitoring data to a server.
- A car which sends information about its location to a cloud-based service.
- A wireless-enabled wearable device that measures data about a person such as the number of steps walked and sends the data to a cloud-based service.

Basic building blocks of an IoT Device

- Sensing : Sensors can be either on-board the IoT device or attached to the device.
- Actuation : IoT devices can have various types of actuators attached that allow taking actions upon the physical entities in the vicinity of the device.
- Communication : Communication modules are responsible for sending collected data to other devices or cloud-based servers/storage and receiving data from other devices and commands from remote applications.
- Analysis & Processing : Analysis and processing modules are responsible for making sense of the collected data.

Block diagram of an IoT Device



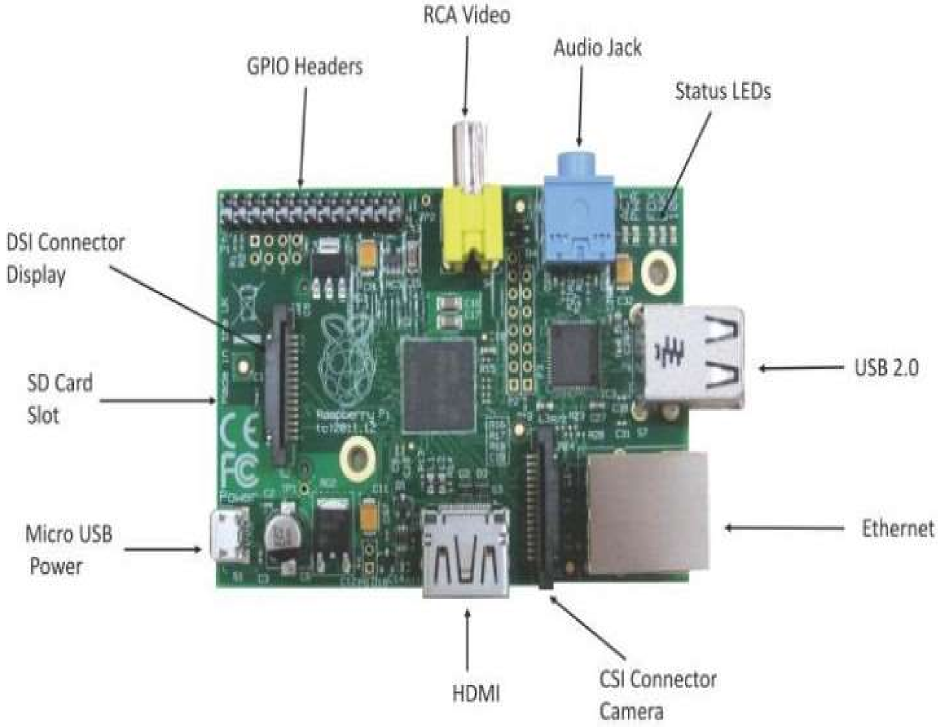
Raspberry Pi

- Raspberry Pi is a low-cost mini-computer with the physical size of a credit card.

- Raspberry Pi runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do.

- Raspberry Pi also allows interfacing sensors and actuators through the general purpose I/O pins.

- Since Raspberry Pi runs Linux operating system, it supports Python "out of the box".



Example of Raspberry Pi board

About the Board

Raspberry Pi board includes the following components/peripherals

Processor and RAM

Most of Raspberry Pi based on ARM processor. The latest version of the board is Raspberry Pi 400 was released in November 2020. It comes with Broadcom BCM2711C0 processor to be clocked at 1.8 GHz, which is slightly higher than the Raspberry Pi 4 Model B which was released in June 2019 with a 1.5 GHz 64-bit quad core ARM Cortex-A72 processor, on-board 802.11ac Wi-Fi, Bluetooth 5, full gigabit Ethernet (throughput not limited), two USB 2.0 ports, two USB 3.0 ports, 2-8 GB of RAM, and dual-monitor support via a pair of micro HDMI (HDMIType D) ports for up to 4K resolution.

USB Ports

Raspberry Pi comes with two USB 2.0 and two USB 3.0 ports. The USB board on Raspberry Pi can provide a current upto 100mA. For connecting devices that draw current more than 100mA, an external USB power hub is required.

Ethernet Ports

Raspberry Pi comes with standard RJ45 ethernet port.

HDMI output

The HDMI port of Raspberry Pi provides both video and audio output. You can connect a Raspberry Pi to a monitor using a HDMI cable.

Composite Video Output

Raspberry Pi comes with a composite video output with an RCA jack that supports both PAL and NTSC video output. The RCA jack can be used to connect older television that have an RCA input only.

Audio output

Raspberry Pi has a 3.5mm audio output jack. This audio jack is used for providing audio output to old television along with the RCA jack for video.

GPIO Pins

Raspberry Pi comes with a number of general purpose input/output pins. There are four types of pins – true GPIO pins, I2C interface pins, SPI interface pins, and serial Rx and Tx pins.

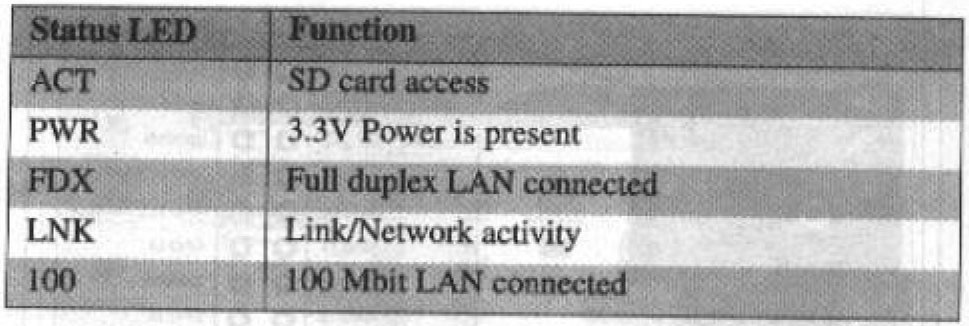
Display Serial Interface (DSI)

The DSI interface can be used to connect an LCD panel to Raspberry Pi.

Camera Serial Interface (CSI)

The CSI interface can be used to connect a camera module to Raspberry Pi.

Status LEDs has 5 status LEDs.



Status LED	Function
ACT	SD card access
PWR	3.3V Power is present
FDX	Full duplex LAN connected
LNK	Link/Network activity
100	100 Mbit LAN connected

Table 7.1: Raspberry Pi Status LEDs

SD Card Slot
Raspberry Pi does not have a built in operating system and storage. You can plug in an SD card loaded with a Linux image to the SD card slot.

Power Output
Raspberry Pi has a micro-USB connector for power output.

Linux on Raspberry Pi

Raspbian
- Raspbian Linux is a Debian Wheezy port optimized for Raspberry Pi.

Arch
- Arch is an Arch Linux port for AMD devices.

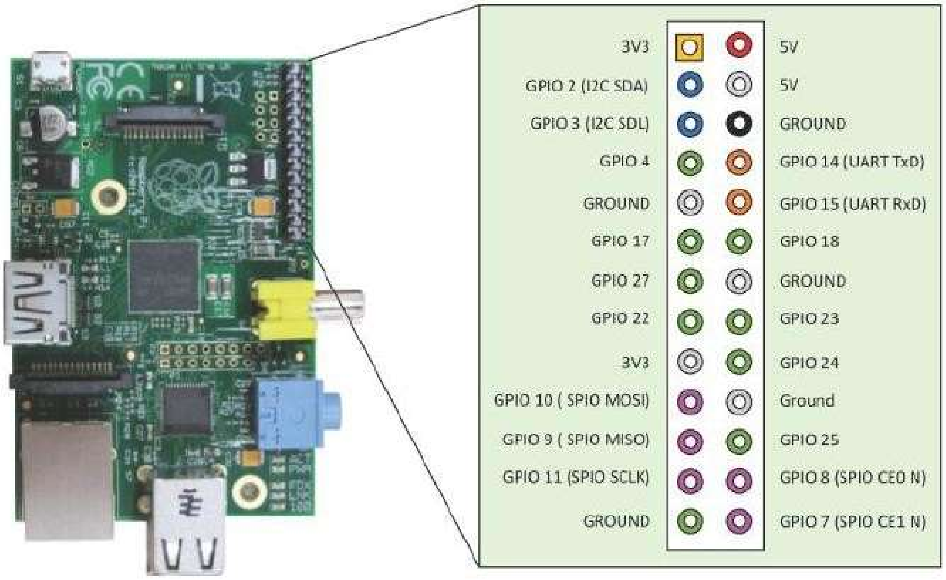
Pidora
- Pidora Linux is a Fedora Linux optimized for Raspberry Pi.

RaspBMC
- RaspBMC is an XBMC media-center distribution for Raspberry Pi.

OpenELEC
- OpenELEC is a fast and user-friendly XBMC media-center distribution.

RISC OS
- RISC OS is a very fast and compact operating system.

Raspberry Pi GPIO



Raspberry Pi Interfaces
Serial
- The serial interface on Raspberry Pi has receive (Rx) and transmit (Tx) pins for communication with serial peripherals.

SPI
- Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices. In SPI connection, there is one master device and one or more peripheral devices. There are five pins on Raspberry Pi for SPI interface.

MISO – (Master In Slave Out) – Master lines for sending data to peripherals
MOSI – (Master Out Slave In) – Slave line for sending data to master.
SCK (Serial Clock) –Clock generated by master to synchronize data transmission
CE0 (Chip Enable 0) – To enable or disable devices
CE0 (Chip Enable 1) – To enable or disable devices

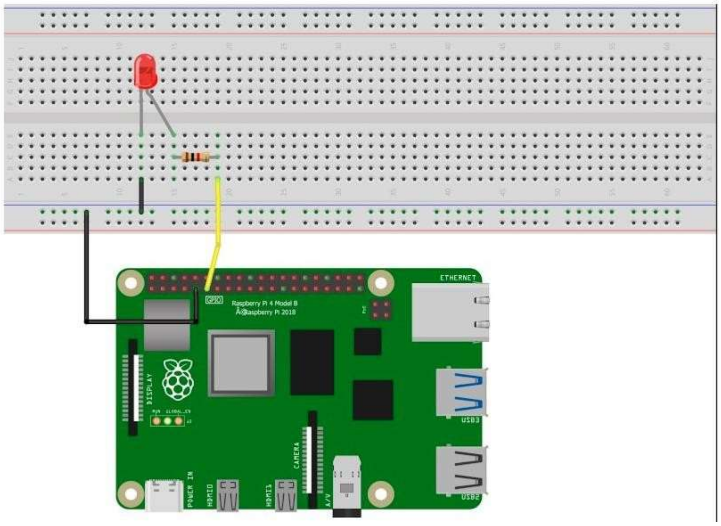
I2C
- The I2C interface pins on Raspberry Pi allow you to connect hardware modules. I2C interface allows synchronous data transfer with just two pins - SDA (data line) and SCL (clock line).

Programming Raspberry Pi with Python

Raspberry Pi runs on Linux and supports python. Therefore you can run any python program that runs on normal computer. However GPIO pins on Raspberry Pi that makes it useful device fro internet of things. You can interface wide variety of sensors and actuators with Raspberry by using GPIO pins and the SPI, I2C and serial interfaces. Input from the sensors connected to Raspberry Pi can be processed and various actions can be taken for instance sending data to server, sending an email, triggering a relay switch.

Controlling LED with Raspberry Pi

In the following example LED is connected to GPIO pin 18. You can connect LED to any other GPIO as well.



The following python program shows blinking an LED connected to Raspberry Pi every second. The program uses RPi GPIO module to control GPIO module on Raspberry Pi. In this program we set pin 18 directions to output and then write True/False alternatively after a delay of one second.

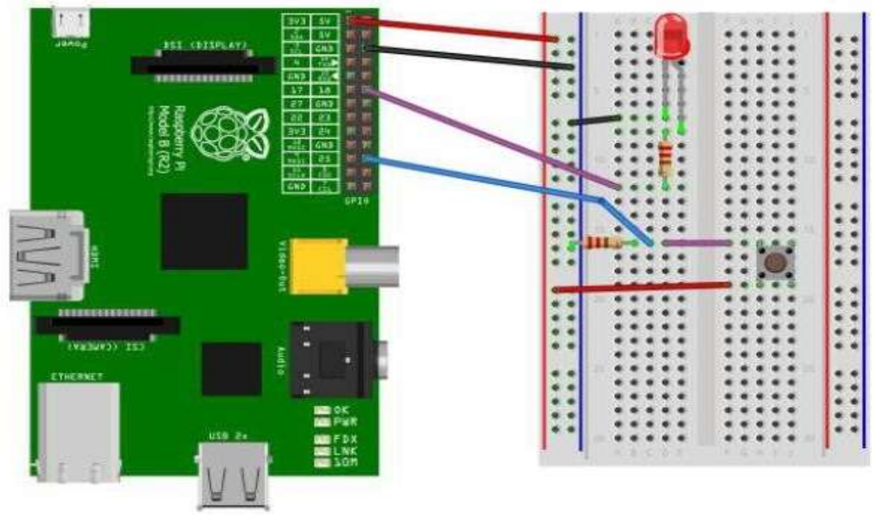
Python Program

```
import RPi.GPIO as GPIO
import time
LED_PIN = 18
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT)

While True:
GPIO.output(LED_PIN, GPIO.HIGH)
time.sleep(1)
GPIO.output(LED_PIN, GPIO.LOW)
Time.sleep(1)
GPIO.cleanup()
```

Interfacing LED and switch with Raspberry Pi

In the below example the LED is connected to GPIO pin 18 and switch is connected to pin 25. In the infinite while loop the value of pin 25 is checked and the state of LED is Toggled if switch is pressed. The above example shows how to get input from GPIO pins and processes the input and take some action. The action in this example is toggling the state of an LED.



Interfacing LED and switch with Raspberry Pi

Python Program

```
from time import sleep
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
#Switch  Pin
GPIO.setup(25, GPIO.IN)
```

```
#LED Pin
GPIO.setup(18, GPIO.OUT)
state=false
def toggleLED(pin):
state = not state
GPIO.output(pin, state)
while True:
try:
if (GPIO.input(25) == True):
toggleLED(pin)
sleep(.01)
except KeyboardInterrupt:
exit()
```

Interfacing a Light Sensor (LDR) with Raspberry Pi

Connect one side of LDR to 3.3V and other side to the 1mF capacitor and also to a GPIO pin. An LED is connected to pin 18 which is controlled based on the light level sensed.

The readLDR function returns a count which is proportional to light level. In this function LDR pin is set to output and low and then ton input. At this point the capacitor starts charging through the resistor until the input reads high. The counter is stopped when the input reads high. The final count is proportional to the light level as greater the amount of light smaller the LDR resistance and greater is the time taken to charge the capacitor.

Python Program

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
ldr_threshold = 1000
LDR_PIN = 18
LIGHT_PIN = 25

def readLDR(PIN):
    reading = 0
    GPIO.setup(LIGHT_PIN, GPIO.OUT)
    GPIO.output(PIN, false)
    time.sleep(0.1)
    GPIO.setup(PIN, GPIO.IN)
    while (GPIO.input (PIN) ==Flase):

        reading=reading+1

    return reading

def switchOnLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, True)

def switchOffLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, False)
while True:
    ldr_reading = readLDR(LDR_PIN)
    if ldr_reading < ldr_threshold:
        switchOnLight (LIGHT_PIN)
    else:
        switchOffLight(LIGHT_PIN)
    time.sleep(1)
```