SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES.
(AUTONOMOUS)
MCA DEPARTMENT

LECTURE NOTES

INTERNET OF THINGS

UNIT-I
INTRODUCTION

IoT comprises of things that have unique identities and are connected to internet. IoT is not limited to just connecting things to the internet but also allow things to communicate and exchange data.

Definition:
A dynamic global network infrastructure with self configuring capabilities based on standard and interoperable communication protocols where physical and virtual things that have identities, physical attributes and virtual personalities and use intelligent interfaces, and are seamlessly integrated into information network, often communicate data associated with users and their environments.

Characteristics:

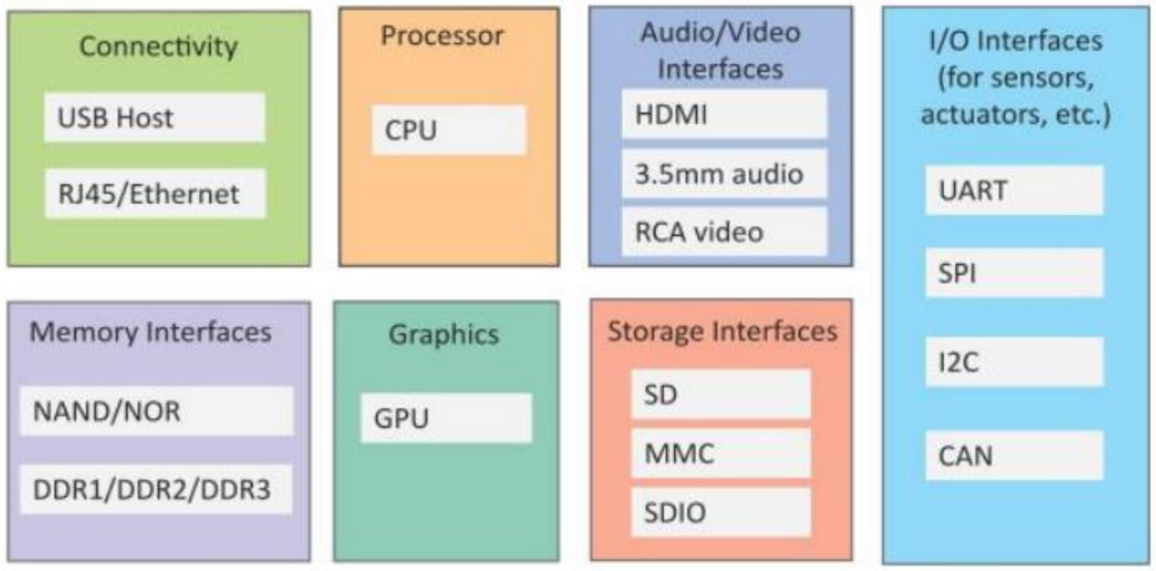
1) **Dynamic & Self Adapting**: IoT devices and systems may have the capability to dynamically adapt with the changing contexts and take actions based on their operating conditions, user‘s context or sensed environment.
   Eg: the surveillance system is adapting itself based on context and changing conditions.
2) **Self Configuring:** allowing a large number of devices to work together to provide certain functionality.
3) **Inter Operable Communication Protocols:** support a number of interoperable communication protocols and can communicate with other devices and also with infrastructure.
4) **Unique Identity:** Each IoT device has a unique identity and a unique identifier (IPaddress).
5) **Integrated into Information Network:** that allow them to communicate and exchange data with other devices and systems.

Applications of IoT:

1) Home
2) Cities
3) Environment
4) Energy
5) Retail
6) Logistics
7) Agriculture
8) Industry
9) Health & LifeStyle

Physical Design Of IoT

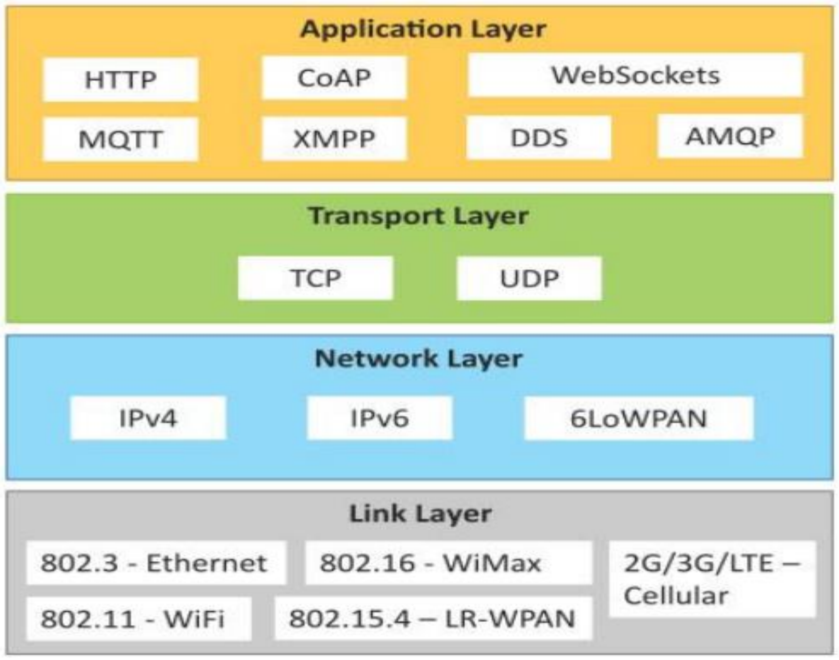
1) **Things in IoT:**



The things in IoT refers to IoT devices which have unique identities and perform remote sensing, actuating and monitoring capabilities. IoT devices can exchange data with other connected devices applications. It collects data from other devices and process data either locally or remotely.
An IoT device may consist of several interfaces for communication to other devices both wired and wireless. These includes (i) I/O interfaces for sensors, (ii) Interfaces for internet connectivity (iii) memory and storage interfaces and (iv) audio/video interfaces.

2) **IoT Protocols:**

a) **Link Layer :** Protocols determine how data is physically sent over the network‘s physical layer or medium. Local network connect to which host is attached. Hosts on the same link exchange data packets over the link layer using link layer protocols. Link layer determines how packets are coded and signaled by the h/w device over the medium to which the host is attached.



Protocols:

- **802.3-Ethernet**: IEEE802.3 is collection of wired Ethernet standards for the link layer. Eg: 802.3 uses co-axial cable; 802.3i uses copper twisted pair connection; 802.3j uses fiber optic connection; 802.3ae uses Ethernet overfiber.

- **802.11-WiFi**: IEEE802.11 is a collection of wireless LAN(WLAN) communication standards including extensive description of link layer. Eg: 802.11a operates in 5GHz band, 802.11b and 802.11g operates in 2.4GHz band, 802.11n operates in 2.4/5GHz band, 802.11ac operates in 5GHz band, 802.11ad operates in 60Ghzband.

- **802.16 - WiMax**: IEEE802.16 is a collection of wireless broadband standards including exclusive description of link layer. WiMax provide data rates from 1.5 Mb/s to 1Gb/s.

- **802.15.4-LR-WPAN**: IEEE802.15.4 is a collection of standards for low rate wireless personal area network(LR-WPAN). Basis for high level communication protocols such as ZigBee. Provides data rate from 40kb/s to250kb/s.

- **2G/3G/4G-Mobile Communication**: Data rates from 9.6kb/s(2G) to up to100Mb/s(4G).

B) **Network/Internet Layer:** Responsible for sending IP datagrams from source n/w to destination n/w. Performs the host addressing and packet routing. Datagrams contains source and destinationaddress.

Protocols:

- **IPv4:** Internet Protocol version4 is used to identify the devices on a n/w using a hierarchical addressing scheme. 32 bit address. Allows total of 2**32addresses.
- **IPv6:** Internet Protocol version6 uses 128 bit address scheme and allows 2**128 addresses.
- **6LOWPAN:**(IPv6overLowpowerWirelessPersonalAreaNetwork) operates in 2.4 GHz frequency range and data transfer 250 kb/s.

C) Transport Layer: Provides end-to-end message transfer capability independent of the underlying n/w. Set up on connection with ACK as in TCP and without ACK as in UDP. Provides functions such as error control, segmentation, flow control and congestion control.
Protocols:

- **TCP:** Transmission Control Protocol used by web browsers(along with HTTP and HTTPS), email(along with SMTP, FTP). Connection oriented and stateless protocol. IP Protocol deals with sending packets, TCP ensures reliable transmission of protocols in order. Avoids n/w congestion and congestion collapse.
- **UDP:** User Datagram Protocol is connectionless protocol. Useful in time sensitive applications, very small data units to exchange. Transaction oriented and stateless protocol. Does not provide guaranteed delivery.

D) **Application Layer:** Defines how the applications interface with lower layer protocols to send data over the n/w. Enables process-to-process communication using ports.
Protocols:

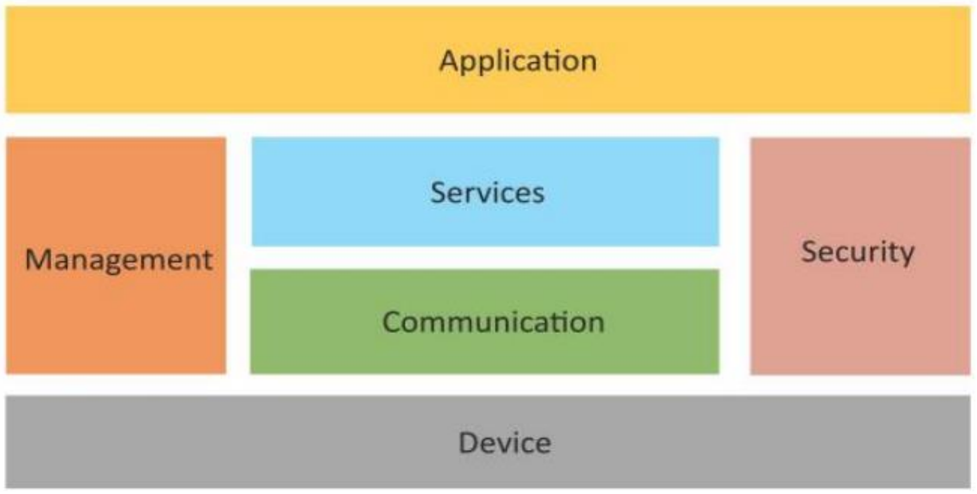
- **HTTP:** Hyper Text Transfer Protocol that forms foundation of WWW. Follow request-response model Stateless protocol.
- **CoAP:** Constrained Application Protocol for machine-to-machine (M2M) applications with constrained devices, constrained environment and constrained n/w. Uses client-server architecture.
- **WebSocket:** allows full duplex communication over a single socket connection.
- **MQTT:** Message Queue Telemetry Transport is light weight messaging protocol based on publish-subscribe model. Uses client server architecture. Well suited for constrained environment.
- **XMPP:** Extensible Message and Presence Protocol for real time communication and streaming XML data between network entities. Support client-server and server-server communication.
- **DDS:** Data Distribution Service is data centric middleware standards for device-to-device or machine-to-machine communication. Uses publish-subscribe model.
- **AMQP:** Advanced Message Queuing Protocol is open application layer protocol for business messaging. Supports both point-to-point and publish-subscribe model.

LOGICAL DESIGN of IoT

Refers to an abstract represent of entities and processes without going into the low level specifies of implementation.

1) IoT Functional Blocks 2) IoT Communication Models 3) IoT Comm. APIs

1) **IoT Functional Blocks:** Provide the system the capabilities for identification, sensing, actuation, communication and management.

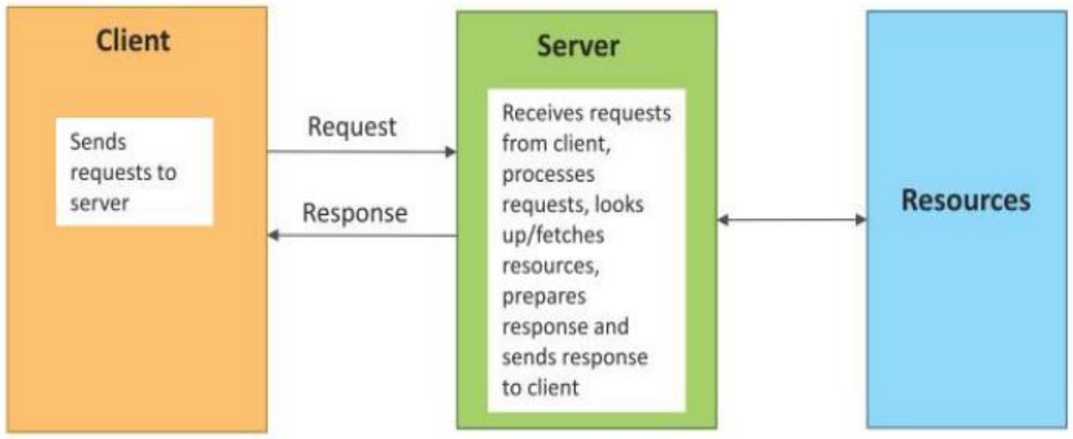


- **Device:** An IoT system comprises of devices that provide sensing, actuation, monitoring and control functions.
- **Communication:** handles the communication for IoT system.
- **Services:** for device monitoring, device control services, data publishing services and services for device discovery.
- **Management:** Provides various functions to govern the IoT system.
- **Security:** Secures IoT system and priority functions such as authentication, authorization,message and context integrity and data security.
- **Application:** IoT application provide an interface that the users can use to control and monitor various aspects of IoT system.

2) IoT CommunicationModels:

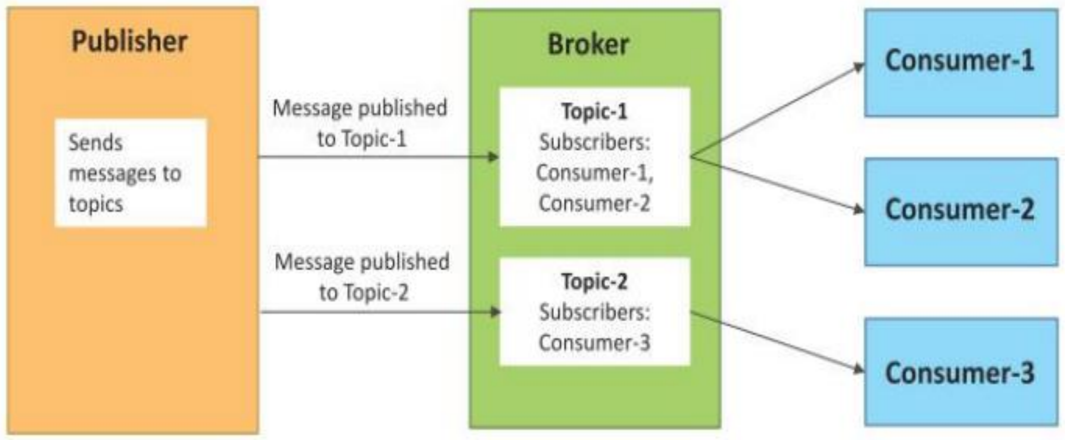
1) Request-Response 2) Publish-Subscribe 3) Push-Pull 4) Exclusive Pair

1) Request-ResponseModel:


Client
Sends requests to server
Request
Response
Server
Receives requests from client, processes requests, looks up/fetches resources, prepares response and sends response to client
Resources

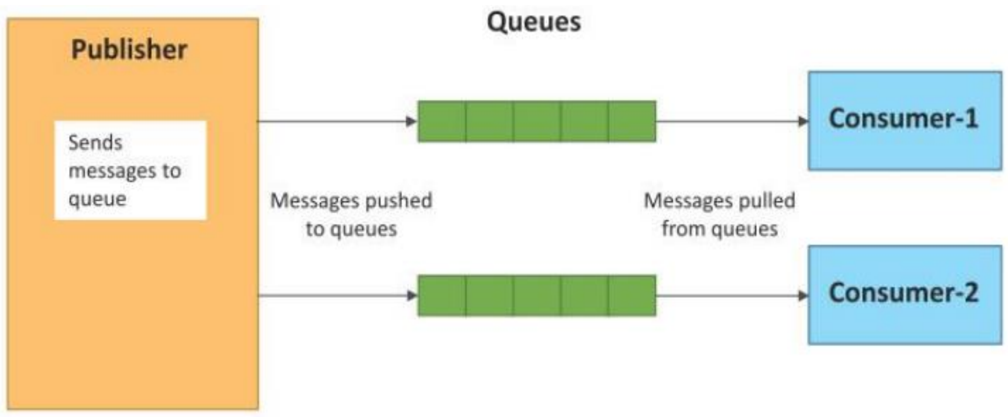
In which the client sends request to the server and the server replies to requests. Is a stateless communication model and each request-response pair is independent of others.

2) Publish-Subscribe Model:

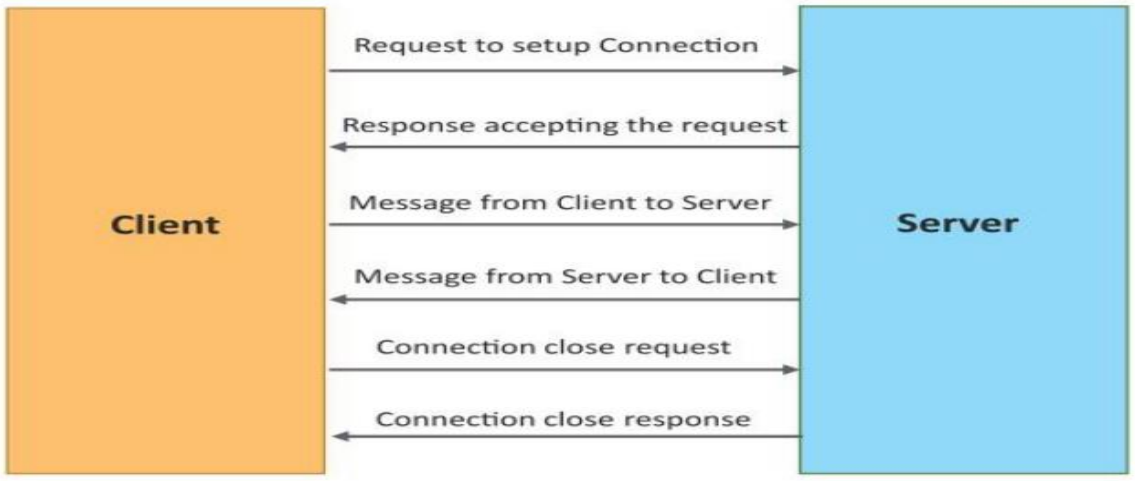

Publisher
Sends messages to topics
Message published to Topic-1
Message published to Topic-2
Broker
Topic-1 Subscribers: Consumer-1, Consumer-2
Topic-2 Subscribers: Consumer-3
Consumer-1
Consumer-2
Consumer-3

It involves publishers, brokers and consumers. Publishers are source of data. Publishers send datato the topics which are managed by the broker. Publishers are not aware of the consumers. Consumers subscribe to the topics which are managed by the broker. When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers.

3) **Push-Pull Model:** in which data producers push data to queues and consumers pull data from the queues. Producers do not need to aware of the consumers. Queues help in decoupling the message between the producers and consumers.



4) **Exclusive Pair:** is bi-directional, fully duplex communication model that uses a persistent connection between the client and server. Once connection is set up it remains open until the client send a request to close the connection. Is a stateful communication model and server is aware of all the open connections.

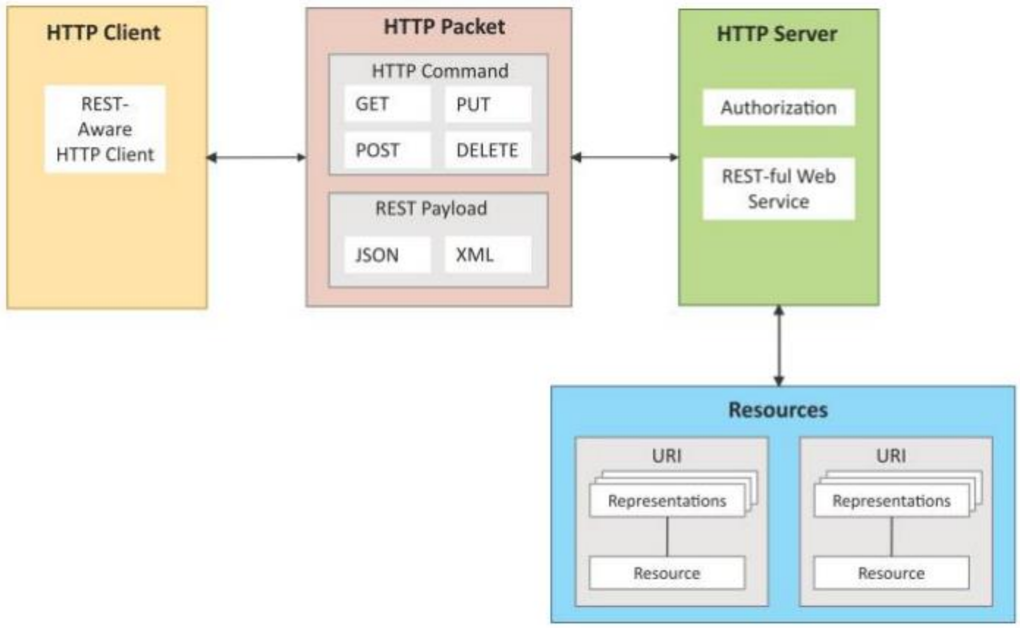

Client
Request to setup Connection
Response accepting the request
Message from Client to Server
Message from Server to Client
Connection close request
Connection close response
Server

3) **IoT Communication APIs:**

- **REST based communication APIs(Request-Response Based Model)**
- **WebSocket based Communication APIs(Exclusive Pair Based Model)**

a) REST based communication APIs: Representational State Transfer(REST) is a set of architectural principles by which we can design web services and web APIs that focus on a system‘s resources and have resource states are addressed and transferred.

The REST architectural constraints: Fig. shows communication between client server with REST APIs.



Client-Server: The principle behind client-server constraint is the separation of concerns. Separation allows client and server to be independently developed and updated.

Stateless: Each request from client to server must contain all the info. Necessary to understand the request, and cannot take advantage of any stored context on the server.

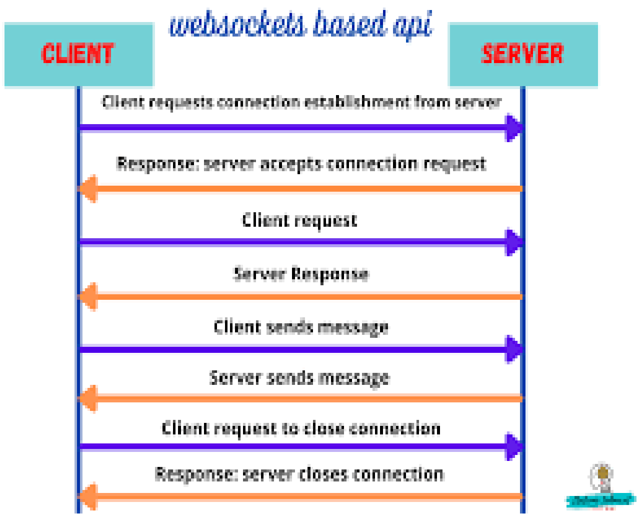
Cache-able: Cache constraint requires that the data within a response to a request be implicitly or explicitly labeled as cache-able or non-cacheable. If a response is cache-able, then a client cache is given the right to reuse that response data for later, equivalent requests.

Layered System: constraints the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting.

User Interface: constraint requires that the method of communication between a client and a server must be uniform.

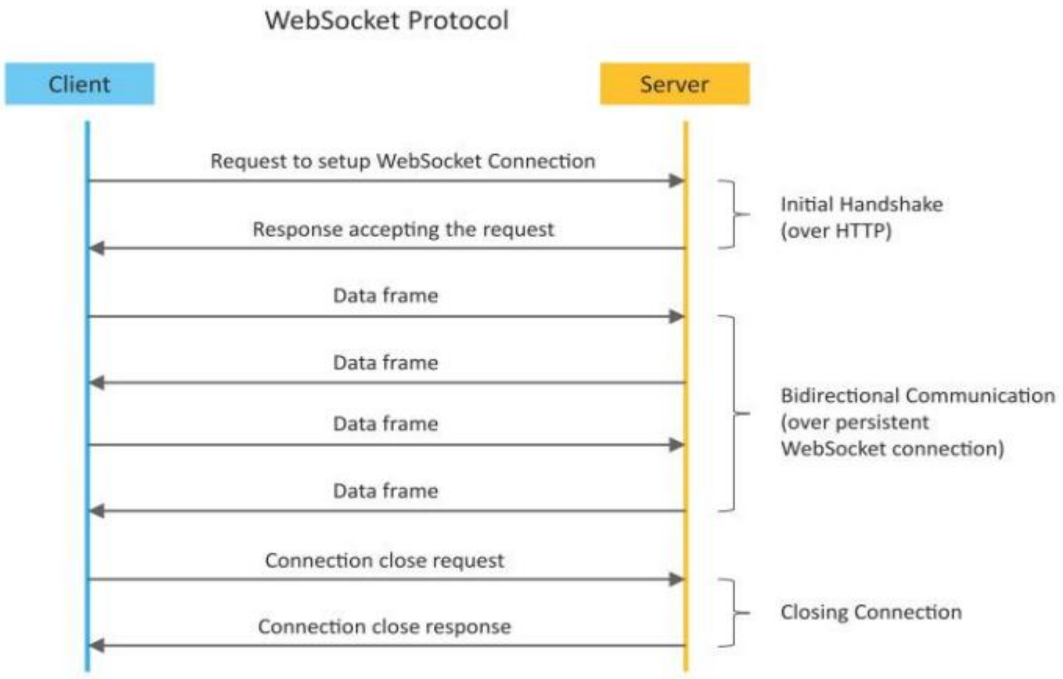
Code on Demand: Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

Request-Response model used by REST:


websockets based api
CLIENT
SERVER
Client requests connection establishment from server
Response: server accepts connection request
Client request
Server Response
Client sends message
Server sends message
Client request to close connection
Response: server closes connection

RESTful webservice is a collection of resources which are represented by URIs. RESTful web API has a base URI(e.g: http://example.com/api/tasks/). The clients and requests to these URIs using the methods defined by the HTTP protocol(e.g: GET, PUT, POST or DELETE). A RESTful web service can support various internet media types.

b) WebSocket Based Communication APIs: WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.


WebSocket Protocol
Client
Server
Request to setup WebSocket Connection
Response accepting the request
Initial Handshake (over HTTP)
Data frame
Data frame
Data frame
Data frame
Bidirectional Communication (over persistent WebSocket connection)
Connection close request
Connection close response
Closing Connection

IoT Enabling Technologies

IoT is enabled by several technologies including Wireless Sensor Networks, Cloud Computing, Big Data Analytics, Embedded Systems, Security Protocols and architectures, Communication Protocols, Web Services, Mobile internet and semantic search engines.

1) **Wireless Sensor Network(WSN):** Comprises of distributed devices with sensors which are used to monitor the environmental and physical conditions. Zig Bee is one of the most popular wireless technologies used by WSNs.
   WSNs used in IoT systems are described as follows:
   - Weather Monitoring System: in which nodes collect temp, humidity and other data, which is aggregated and analyzed.
   - Indoor air quality monitoring systems: to collect data on the indoor air quality and concentration of various gases.
   - Soil Moisture Monitoring Systems: to monitor soil moisture at various locations.
   - Surveillance Systems: use WSNs for collecting surveillance data(motion datadetection).
   - Smart Grids : use WSNs for monitoring grids at various points.
   - Structural Health Monitoring Systems: Use WSNs to monitor the health of structures(building, bridges) by collecting vibrations from sensor nodes deployed at various points in the structure.

2) **Cloud Computing:** Services are offered to users in different forms.
   - Infrastructure-as-a-service (IaaS): provides users the ability to provision computing and storage resources. These resources are provided to the users as a virtual machine instances and virtual storage.
   - Platform-as-a-Service (PaaS): provides users the ability to develop and deploy application in cloud using the development tools, APIs, software libraries and services provided by the cloud service provider.
   - Software-as-a-Service (SaaS): provides the user a complete software application orthe user interface to the application itself.

3) **Big Data Analytics:** Some examples of big data generated by IoT are
   - Sensor data generated by IoT systems.
   - Machine sensor data collected from sensors established in industrial and energy systems.
   - Health and fitness data generated IoT devices.
   - Data generated by IoT systems for location and tracking vehicles.
   - Data generated by retail inventory monitoring systems.

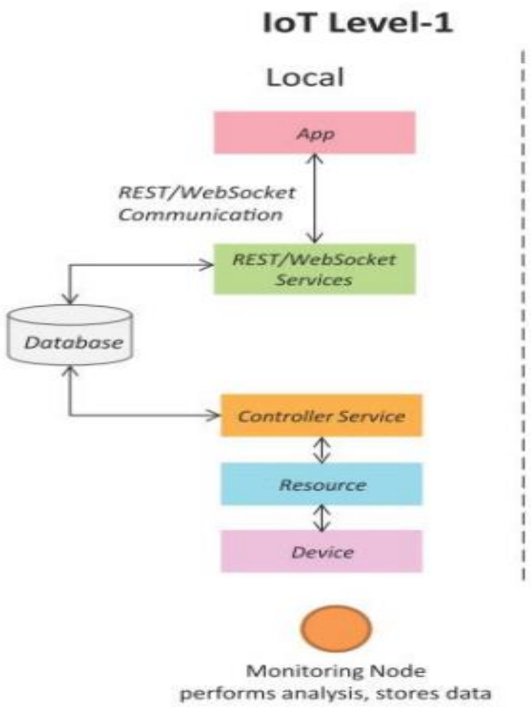
4) **Communication Protocols:** form the back-bone of IoT systems and enable network connectivity and coupling to applications.

- Allow devices to exchange data over network.
- Define the exchange formats, data encoding addressing schemes for device and routing of packets from source to destination.
- It includes sequence control, flow control and retransmission of lost packets.

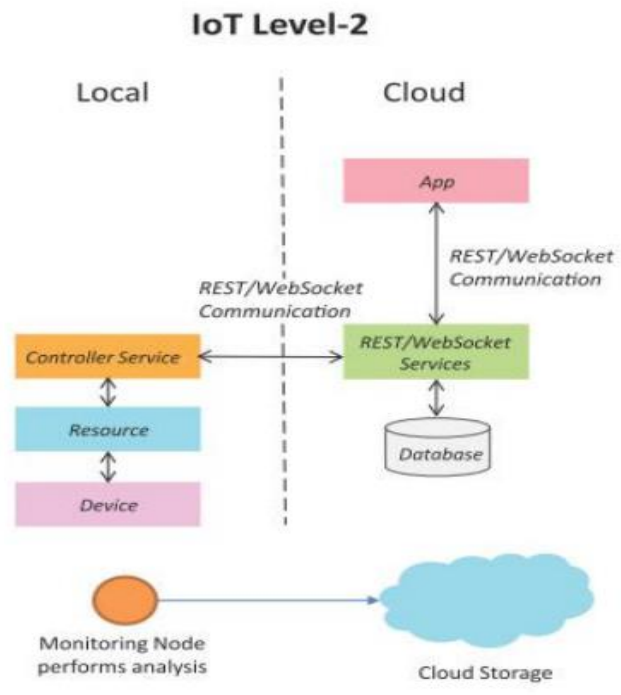
5) **Embedded Systems:** is a computer system that has computer hardware and software embedded to perform specific tasks. Embedded System range from low cost miniaturized devices such as digital watches to devices such as digital cameras, POS terminals, vending machines, appliances etc.,

IoT Levels and Deployment Templates

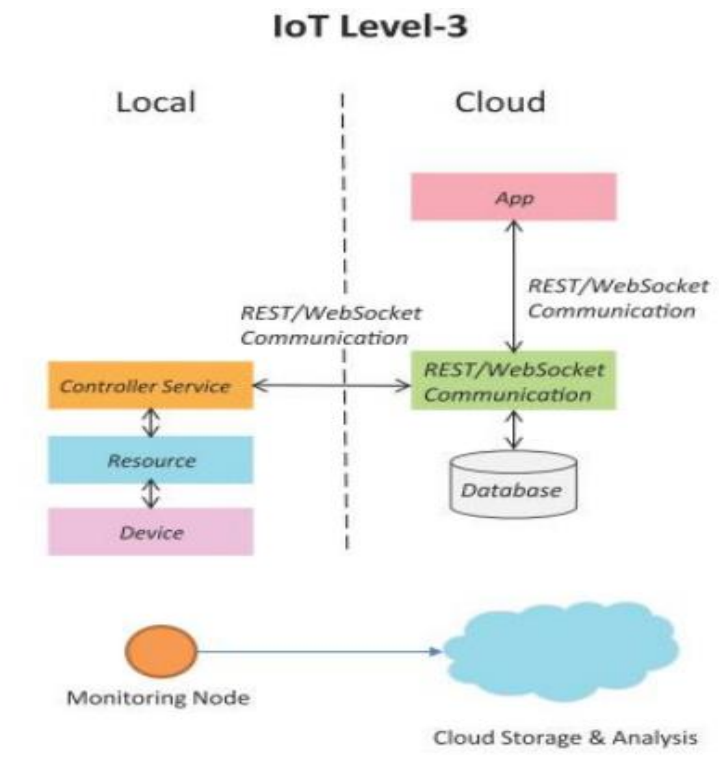
1) **IoT Level1:** System has a single node that performs sensing and/or actuation, stores data, performs analysis and host the application as shown in fig. Suitable for modeling low cost and low complexity solutions where the data involved is not big and analysis requirement are not computationally intensive. An e.g., of IoT Level1 is Home automation.



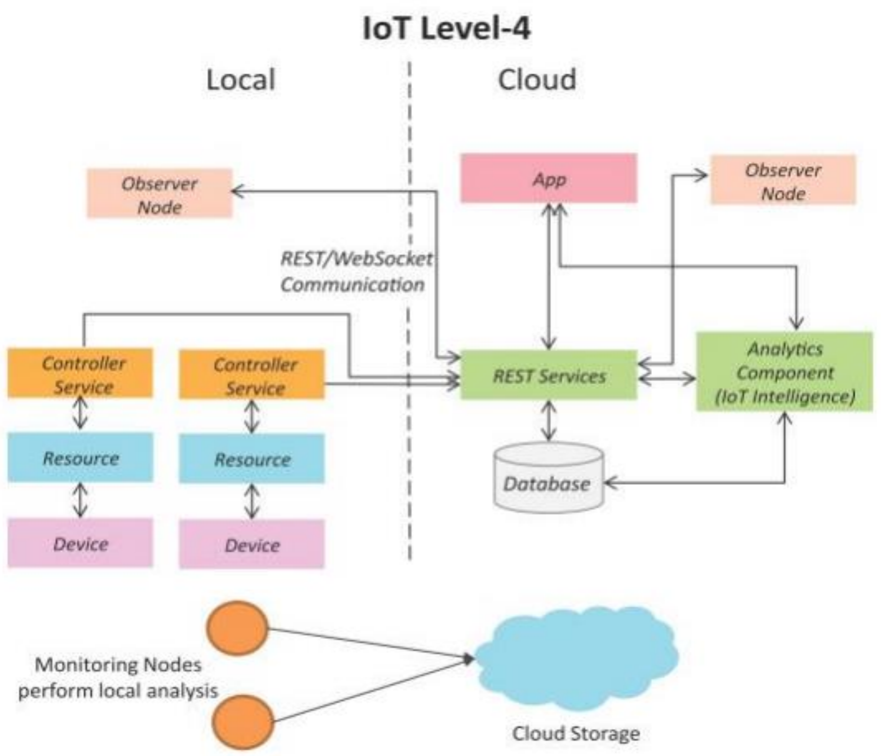
2) **IoT Level2:** has a single node that performs sensing and/or actuating and local analysis as shown in fig. Data is stored in cloud and application is usually cloud based. Level2 IoT systems are suitable for solutions where data are involved is big, however, the primary analysis requirement is not computationally intensive and can be done locally itself. An e,g., of Level2 IoT system for SmartIrrigation.



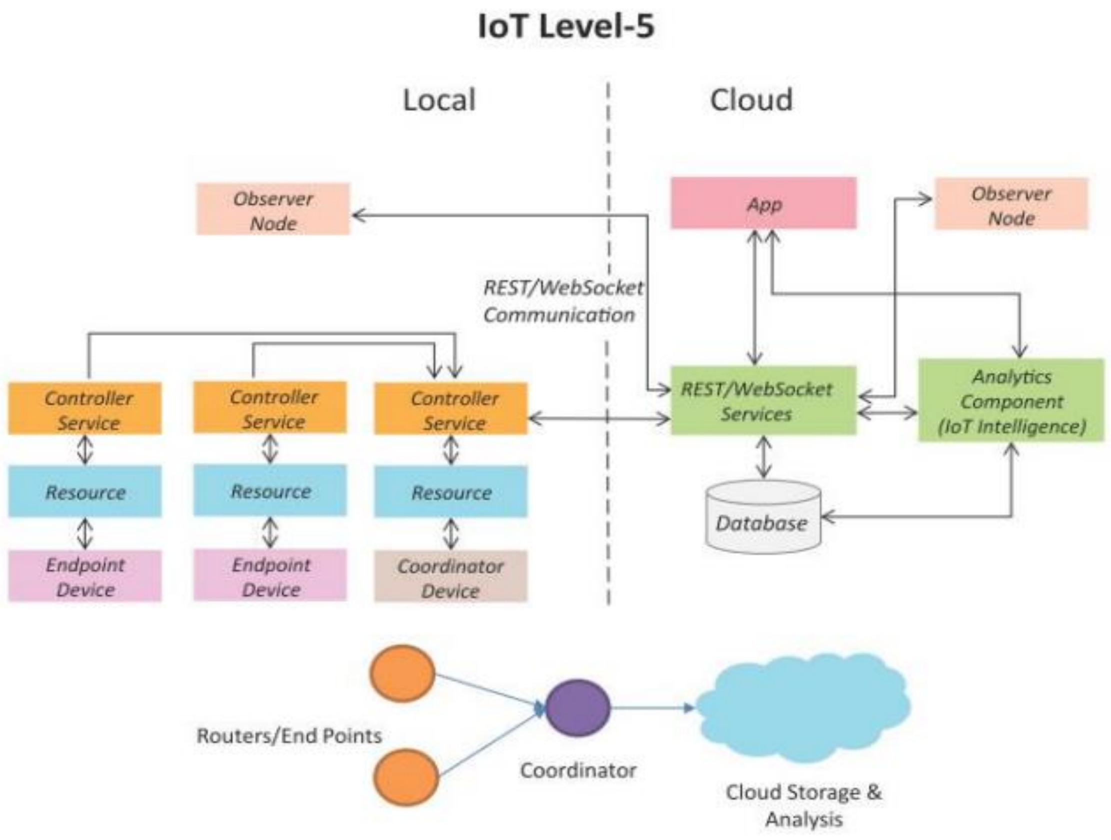
3) **IoT Level3:** system has a single node. Data is stored and analyzed in the cloud application is cloud based as shown in fig. Level3 IoT systems are suitable for solutions where the data involved is big and analysis requirements are computationally intensive. An example of IoT level3 system for tracking package handling.



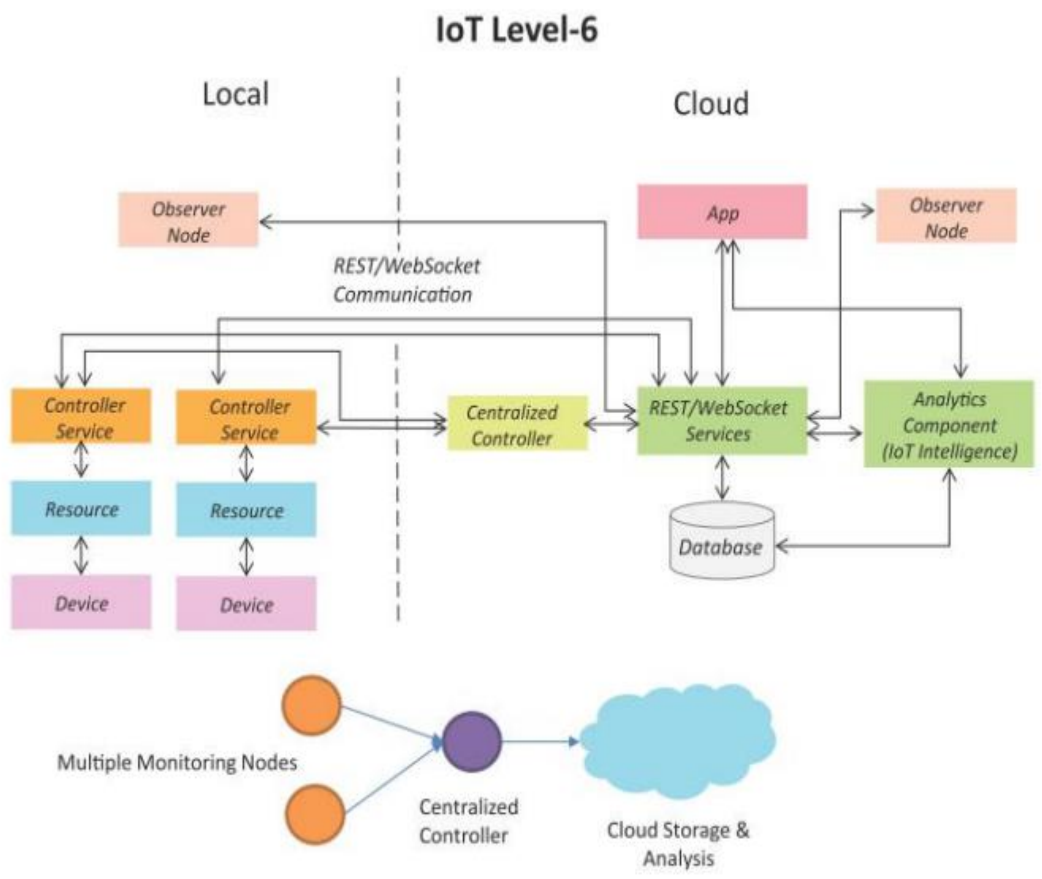
4) **IoT Level4:** System has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud based as shown in fig. Level4 contains local and cloud based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices. An example of a Level4 IoT system for Noise Monitoring.



5) **IoT Level5:** System has multiple end nodes and one coordinator node as shown in fig. The end nodes that perform sensing and/or actuation. Coordinator node collects data from the end nodes and sends to the cloud. Data is stored and analyzed in the cloud and the application is cloud based. Level5 IoT systems are suitable for solution based on wireless sensor network, in which data involved is big and analysis requirements are computationally intensive. An example of Level5 system for Forest Fire Detection.

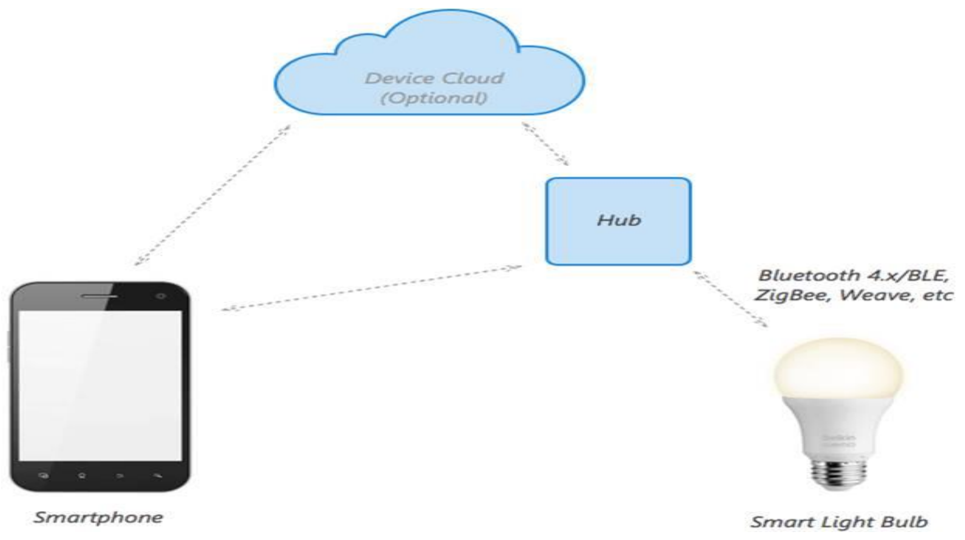


6) **IoT Level6:** System has multiple independent end nodes that perform sensing and/or actuation and sensed data to the cloud. Data is stored in the cloud and application is cloud based as shown in fig. The analytics component analyses the data and stores the result in the cloud data base. The results are visualized with cloud based application. The centralized controller is aware of the status of all the end nodes and sends control commands to nodes. An example of a Level6 IoT system for Weather Monitoring System.



DOMAIN SPECIFIC IoTs

1) **Home Automation:**
   a) **Smart Lighting:** helps in saving energy by adapting the lighting to the ambient conditions and switching on/off or diming the light when needed.



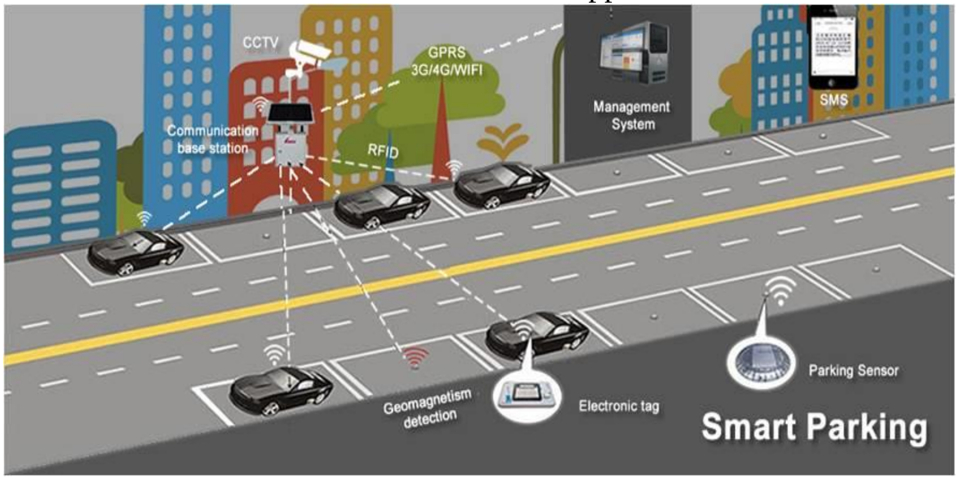
b) **Smart Appliances:** make the management easier and also provide status information to the users remotely.

c) **Intrusion Detection:** use security cameras and sensors(PIR sensors and door sensors) to detect intrusion and raise alerts. Alerts can be in the form of SMS or email sent to the user.

d) **Smoke/Gas Detectors:** Smoke detectors are installed in homes and buildings to detect smoke that is typically an early sign of fire. Alerts raised by smoke detectors can be in the form of signals to a fire alarm system. Gas detectors can detect the presence of harmful gases such as CO, LPG etc.,

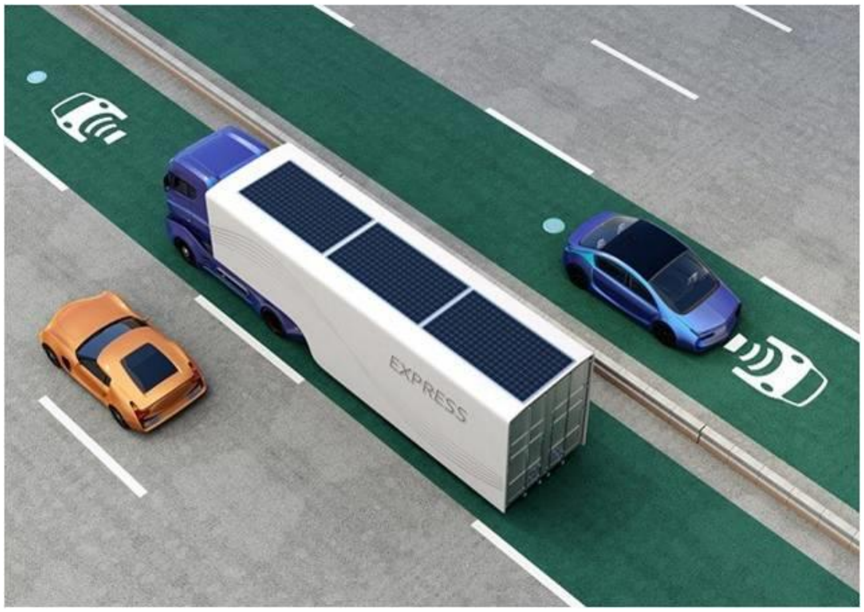
2) Cities:

a) **Smart Parking:** make the search for parking space easier and convenient for drivers. Smart parking are powered by IoT systems that detect the no. of empty parking slots and send information over internet to smart application back ends.

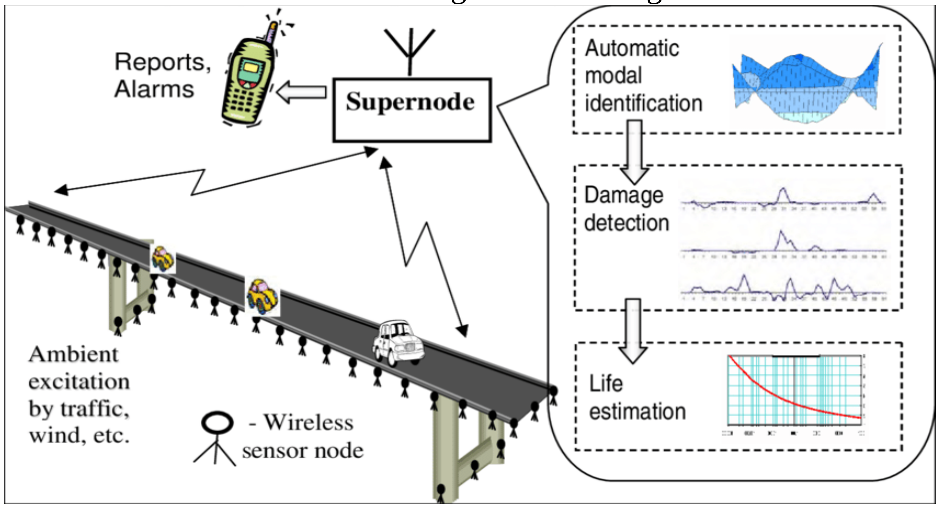


b) **Smart Lighting:** for roads, parks and buildings can help in saving energy.

c) **Smart Roads:** Equipped with sensors can provide information on driving condition, travel time estimating and alert in case of poor driving conditions, traffic condition and accidents.



d) **Structural Health Monitoring:** uses a network of sensors to monitor the vibration levels in the structures such as bridges and buildings.



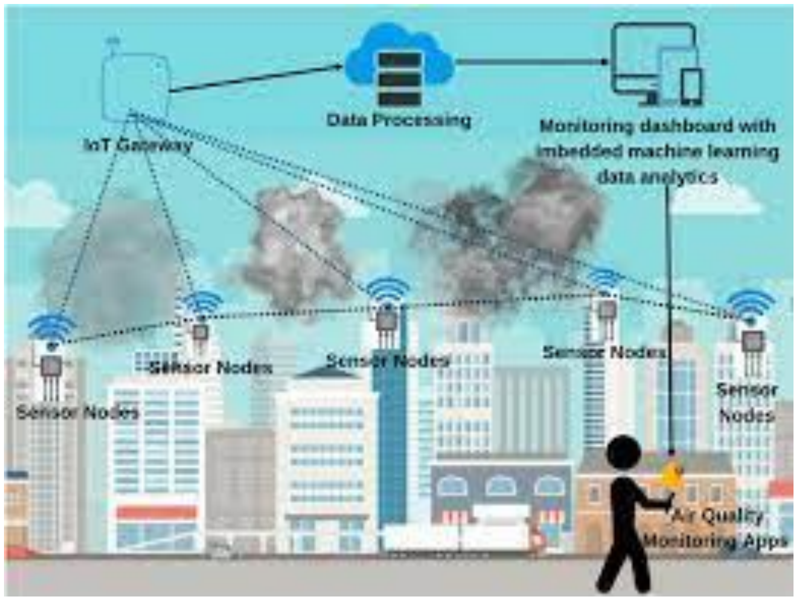
e) **Surveillance:** The video feeds from surveillance cameras can be aggregated in cloud based scalable storage solution.

f) **Emergency Response:** IoT systems for fire detection, gas and water leakage detection can help in generating alerts and minimizing their effects on the critical infrastructures.

3) Environment:

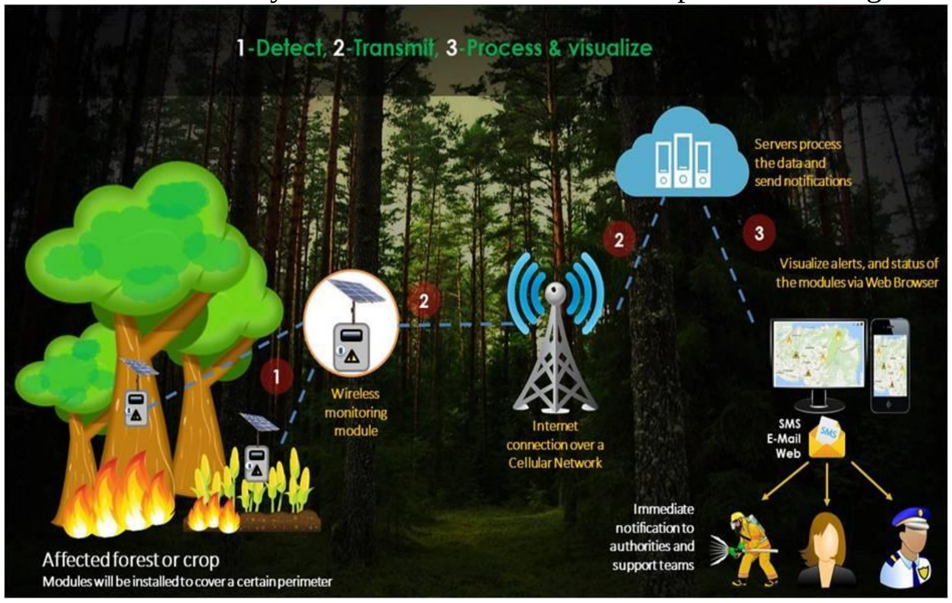
a) **Weather Monitoring:** Systems collect data from a no. of sensors attached and send the data to cloud based applications and storage back ends. The data collected in cloud can then be analyzed and visualized by cloud based applications.

b) **Air Pollution Monitoring:** System can monitor emission of harmful gases(CO2, CO, NO, NO2 etc.,) by factories and automobiles using gaseous and meteorological sensors. The collected data can be analyzed to make informed decisions on pollutions control approaches.


IoT Gateway
Data Processing
Monitoring dashboard with imbedded machine learning data analytics
Sensor Nodes
Sensor Nodes
Sensor Nodes
Sensor Nodes
Sensor Nodes
Air Quality Monitoring Apps

c) **Noise Pollution Monitoring:** Due to growing urban development, noise levels in cities have increased and even become alarmingly high in some cities. IoT based noise pollution monitoring systems use a no. of noise monitoring systems that are deployed at different places in a city. The data on noise levels from the station is collected on servers or in the cloud. The collected data is then aggregated to generate noise maps.

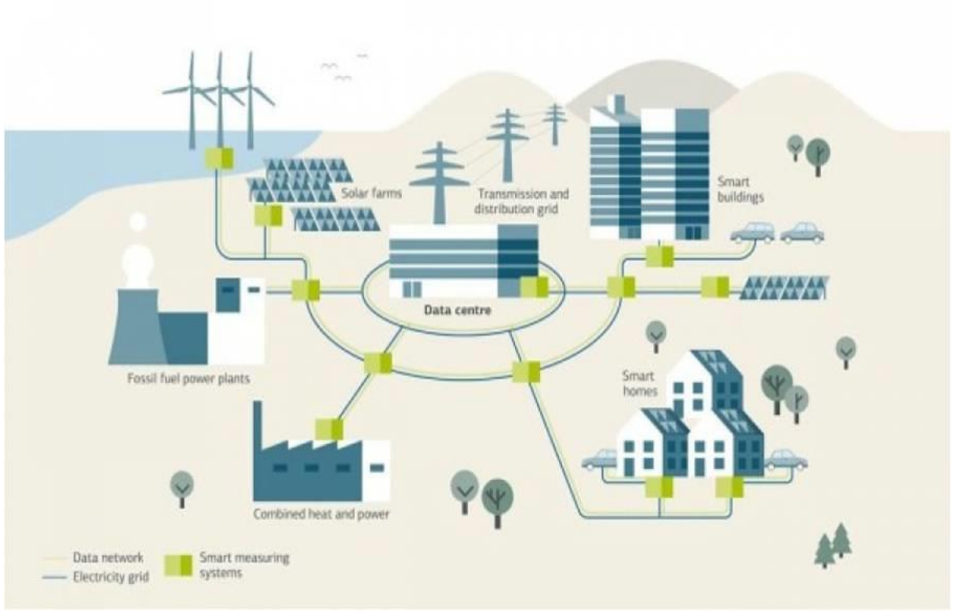
d) **Forest Fire Detection:** Forest fire can cause damage to natural resources, property and human life. Early detection of forest fire can help in minimizing damage.


1-Detect, 2-Transmit, 3-Process & visualize
Servers process the data and send notifications
2
3
Visualize alerts, and status of the modules via Web Browser
2
1
Wireless monitoring module
Internet connection over a Cellular Network
SMS
E-Mail
Web
Immediate notification to authorities and support teams
Affected forest or crop
Modules will be installed to cover a certain perimeter

e) **River Flood Detection:** River floods can cause damage to natural and human resources and human life. Early warnings of floods can be given by monitoring the water level and flow rate. IoT based river flood monitoring system uses a no. of sensor nodes that monitor the water level and flow rate sensors.

4) **Energy:**

a) **Smart Grids:** is a data communication network integrated with the electrical grids that collects and analyze data captured in near-real-time about power transmission, distribution and consumption. Smart grid technology provides predictive information and recommendations to utilities, their suppliers, and their customers on how best to manage power. By using IoT based sensing and measurement technologies, the health of equipment and integrity of the grid can be evaluated.

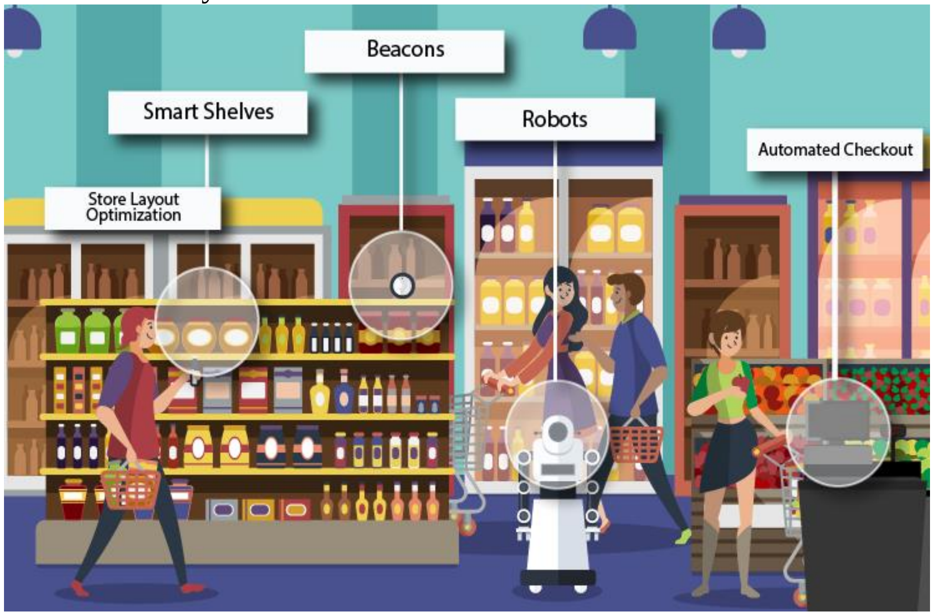


b) **Renewable Energy Systems:** IoT based systems integrated with the transformers at the point of interconnection measure the electrical variables and how much power is fed into the grid. For wind energy systems, closed-loop controls can be used to regulate the voltage at point of interconnection which coordinate wind turbine outputs and provides power support.

c) **Prognostics:** In systems such as power grids, real-time information is collected using specialized electrical sensors called Phasor Measurment Units(PMUs) at the substations. The information received from PMUs must be monitored in real-time for estimating the state of the system and for predicting failures.

5) **Retail:**

a) **Inventory Management:** IoT systems enable remote monitoring of inventory using data collected by RFID readers.

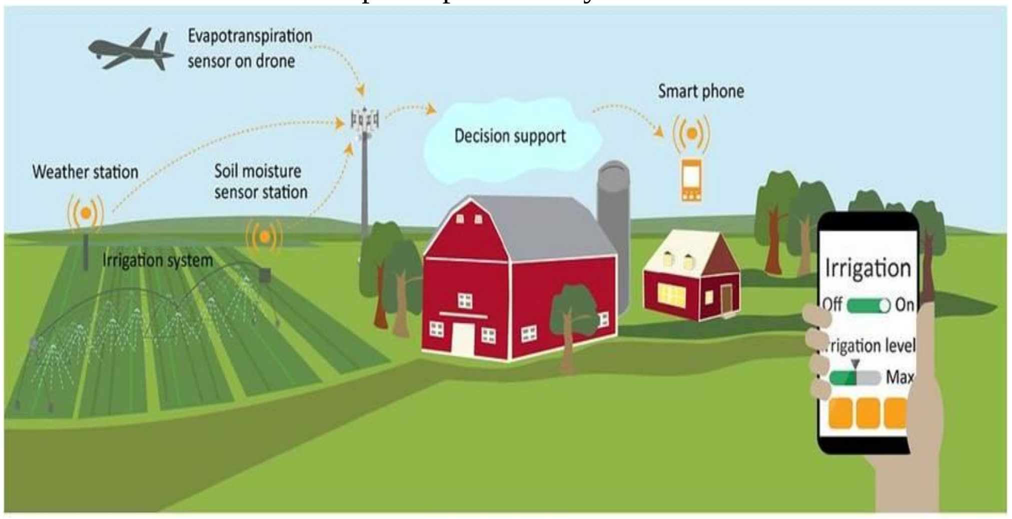


b) **Smart Payments:** Solutions such as contact-less payments powered by technologies such as Near Field Communication (NFC) and Bluetooth.

c) **Smart Vending Machines:** Sensors in a smart vending machines monitors its operations and send the data to cloud which can be used for predictive maintenance.

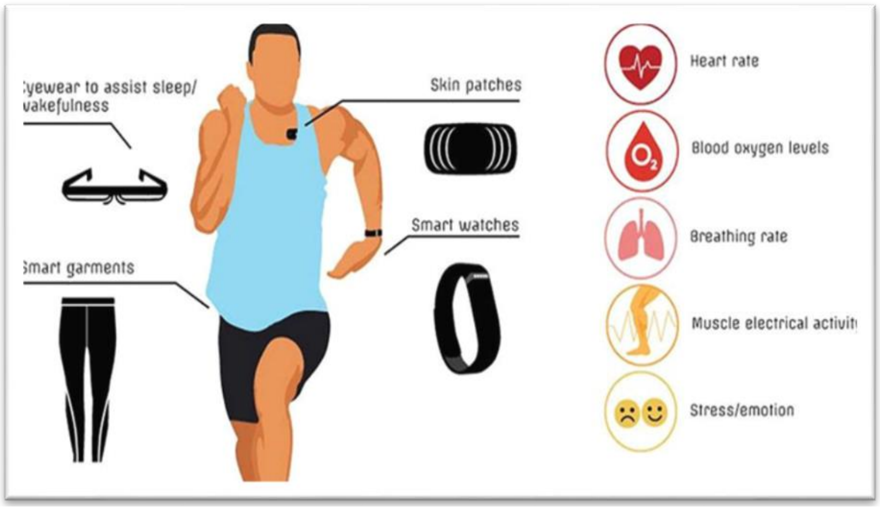
6) **Logistics:**
   a) **Route generation & scheduling:** IoT based system backed by cloud can provide first response to the route generation queries and can be scaled upto serve a large transportation network.
   b) **Fleet Tracking:** Use GPS to track locations of vehicles in real-time.
   c) **Shipment Monitoring:** IoT based shipment monitoring systems use sensors such as temp, humidity, to monitor the conditions and send data to cloud, where it can be analyzed to detect food spoilage.
   d) **Remote Vehicle Diagnostics:** Systems use on-board IoT devices for collecting data on Vehicle operations (speed, RPMetc.,) and status of various vehicle subsystems.

7) **Agriculture:**
   a) **Smart Irrigation:** to determine moisture amount in soil.
   b) **Green House Control:** to improve productivity.



1) **Industry:**
   a) Machine diagnosis and prognosis
   b) Indoor Air Quality Monitoring

2) **Health and LifeStyle:**
   a) Health & Fitness Monitoring
   b) Wearable Electronics


yewear to assist sleep/
vakefulness
Skin patches
Smart watches
Smart garments
Heart rate
Blood oxygen levels
Breathing rate
Muscle electrical activit
Stress/emotion

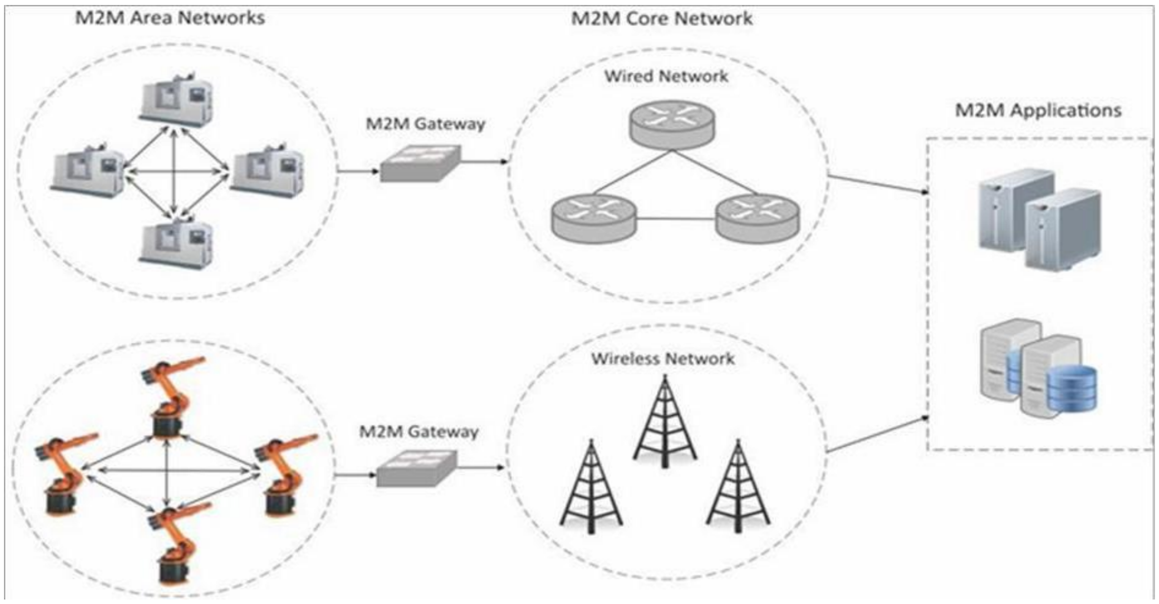
UNIT-II

IoT and M2M

M2M:
Machine-to-Machine (M2M) refers to networking of machines (or devices) for the purposeof remote monitoring and control and data exchange.

- Term which is often synonymous with IoT is Machine-to-Machine (M2M).
- IoT and M2M are often used interchangeably.

Fig. Shows the end-to-end architecture of M2M systems comprises of M2M area networks, communication networks and M2M applications.



- An M2M area network comprises of machines(or M2M nodes) whiach have embedded network modules for sensing, actuation and communicating various communiction protocols can be used for M2M LAN such as ZigBee, Bluetooth, M-bus, Wireless M-Bus etc., These protocols provide connectivity between M2M nodes within an M2M area network.
- The communication network provides connectivity to remote M2M area networks. The communication network provides connectivity to remote M2M area network. The communication network can use either wired or wireless network(IP based). While the M2M are networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based network. Since non-IP based protocols are used within M2M area network, the M2M nodes within one network cannot communicate with nodes in an external network.
- To enable the communication between remote M2M are network, M2M gateways are used.

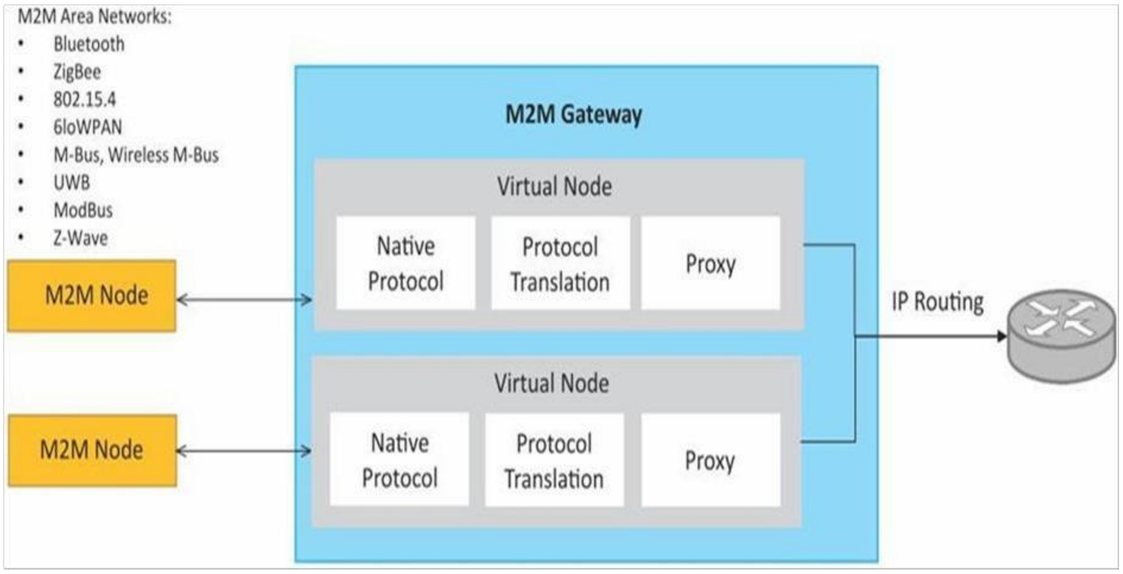


Fig. Shows a block diagram of an M2M gateway. The communication between M2M nodes and the M2M gateway is based on the communication protocols which are naive to the M2M are network. M2M gateway performs protocol translations to enable Ip-connectivity for M2M are networks. M2M gateway acts as a proxy performing translations from/to native protocols to/from Internet Protocol(IP). With an M2M gateway, each mode in an M2M area network appears as a virtualized node for external M2M area networks.

Differences between IoT and M2M

1) **Communication Protocols:**
   - □ Commonly uses M2M protocols include ZigBee, Bluetooth, ModBus, M-Bus, Wireless M-Bustec.,
   - □ In IoT uses HTTP, CoAP, WebSocket, MQTT, XMPP, DDS, AMQPetc.,
2) **Machines in M2M Vs Things inIoT:**
   - □ Machines in M2M will be homogenous whereas Things in IoT will be heterogeneous.
3) **Hardware Vs Software Emphasis:**
   - □ the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.
4) **Data Collection &Analysis**
   - □ M2M data is collected in point solutions and often in on-premises storage infrastructure.
   - □ The data in IoT is collected in the cloud (can be public, private orhybrid cloud).

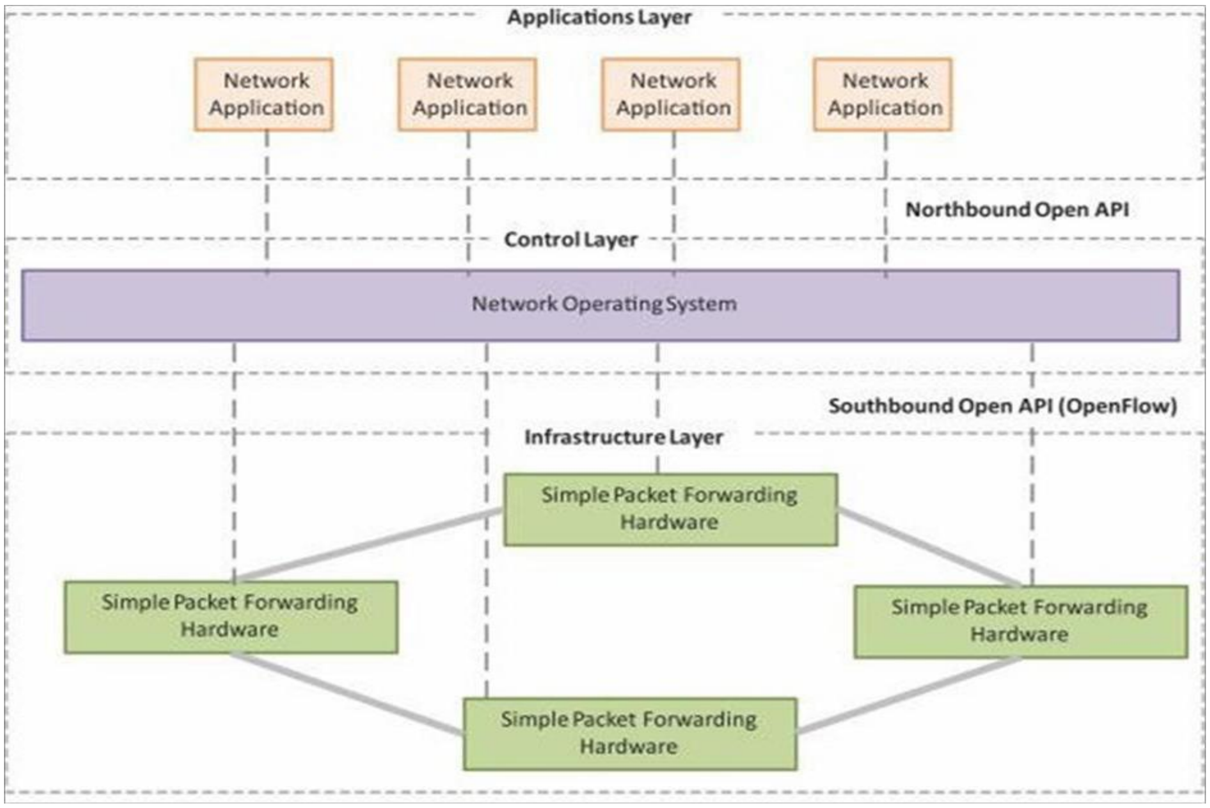
5) **Applications**

- □ M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and on- premises enterprise applications.
- □ IoT data is collected in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc.

SDN and NVF for IoT

Software Defined Networking(SDN):

- Software-Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane and centralizes the network controller.
- Software-based SDN controllers maintain a unified view of the network
- The underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional networks.



SDN Architecture

Key elements of SDN:

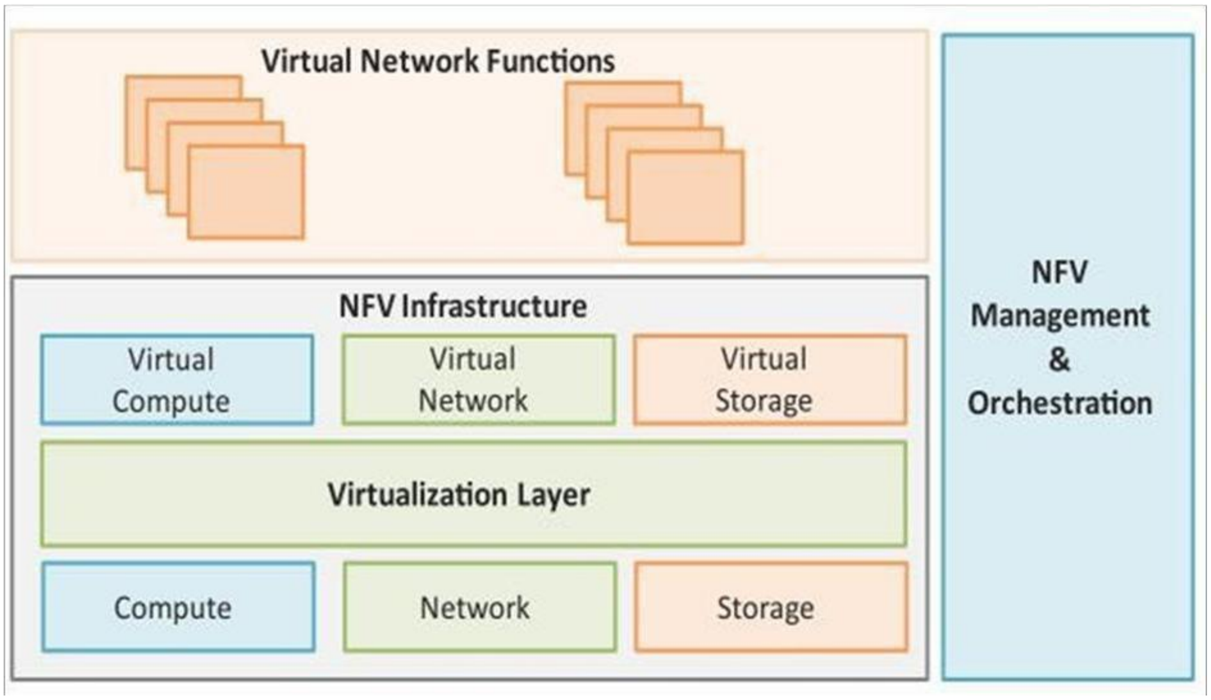
1) **Centralized Network Controller**
   With decoupled control and data planes and centralized network controller, the network administrators can rapidly configure the network.

2) **Programmable Open APIs**
   SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface).

3) **Standard Communication Interface (Open Flow)**
   SDN architecture uses a standard communication interface between the control and infrastructure layers (Southbound interface). Open Flow, which is defined by the Open Networking Foundation (ONF) is the broadly accepted SDN protocol for the South bound interface.

Network Function Virtualization(NFV)

- Network Function Virtualization (NFV) is a technology that leverages virtualization to consolidate the heterogeneous network devices onto industry standard high volume servers, switches and storage.
- NFV is complementary to SDN as NFV can provide the infrastructure on which SDN can run.



Key elements of NFV:

NFV Architecture

1) **Virtualized Network Function(VNF):**
   VNF is a software implementation of a network function which is capable of running over the NFV Infrastructure (NFVI).

2) **NFV Infrastructure(NFVI):**
   NFVI includes compute, network and storage resources that are virtualized.

3) **NFV Management and Orchestration:**
   NFV Management and Orchestration focuses on all virtualization-specific management tasks and covers the orchestration and life-cycle management of physical and/or software resources that support the infrastructure virtualization, and the life-cycle management of VNFs.

Need for IoT Systems Management

Managing multiple devices within a single system requires advanced management capabilities.

1) **Automating Configuration :** IoT system management capabilities can help inautomating the system configuration.
2) **Monitoring Operational & Statistical Data :** Management systems can help in monitoring operational and statistical data of a system. This data can be used for fault diagnosis or prognosis.
3) **Improved Reliability:** A management system that allows validating the system configurations before they are put into effect can help in improving the system reliability.
4) **System Wide Configurations :** For IoT systems that consists of multiple devices or nodes, ensuring system wide configuration can be critical for the correct functioning of the system.
5) **Multiple System Configurations :** For some systems it may be desirable to have multiple valid configurations which are applied at different times or in certain conditions.
6) **Retrieving & Reusing Configurations :** Management systems which have the capability of retrieving configurations from devices can help in reusing the configurations for other devices of the same type.

Simple Network Management Protocol (SNMP)

SNMP is an application layer protocol that uses UDP port number 161/162.SNMP is used to monitor the network, detect network faults, and sometimes even used to configure remote devices.

SNMP components –

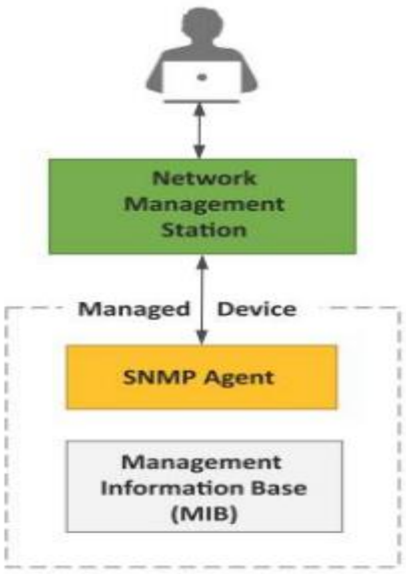
There are 3 components of SNMP:

1. **SNMP Manager –**
   It is a centralized system used to monitor network. It is also known as Network Management Station

(NMS)

2. **SNMP agent –**
   It is a software management software module installed on a managed device. Managed devices can be network devices like PC, routers, switches, servers, etc.

3. **Management Information Base –**
   MIB consists of information on resources that are to be managed. This information is organized hierarchically. It consists of objects instances which are essentially variables.



Strength of SNMP:

1. It is simple to implement.
2. Agents are widely implemented.
3. Agent level overhead is minimal.
4. It is robust and extensible.
5. Polling approach is good for LAN based managed object.
6. It offers the best direct manager agent interface.
7. SNMP meet a critical need.

Limitations of SNMP Management

One of the chief limitations of SNMP network management comes from its focus on device-specific metrics. While these are essential to understanding device status, they are siloed from other infrastructure data sets such as traffic flow records. In addition, SNMP monitoring doesn't provide any insight into user experience or digital experience.

See how ThousandEyes' top rated network monitoring can help you to complement your current internal network SNMP monitoring practices with a top-down approach to Digital Experience Monitoring that incorporates app experience, end-to-end network metrics, network paths, Internet routing and outage insights, plus SNMP device layer status and context.

Limitation of SNMP:

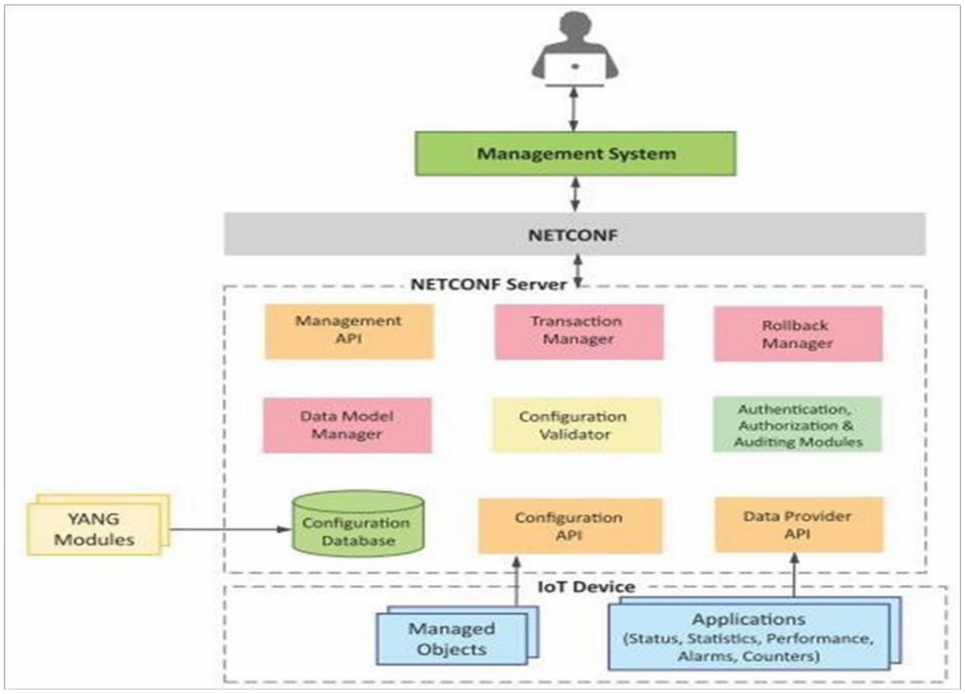
1. It is too simple and does not scale well.
2. There is no object oriented data view.
3. It has no standard control definition.
4. It has many implementation specific (private MIB) extensions.
5. It has high communication overhead due to polling

IoT Systems Management with NETCONF-YANG

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

The generic approach of IoT device management weith NETCONF-YANG. Roles of various componentsare:

1) ManagementSystem
2) ManagementAPI
3) TransactionManager
4) RollbackManager
5) Data ModelManager
6) ConfigurationValidator
7) ConfigurationDatabase
8) ConfigurationAPI
9) Data ProviderAPI

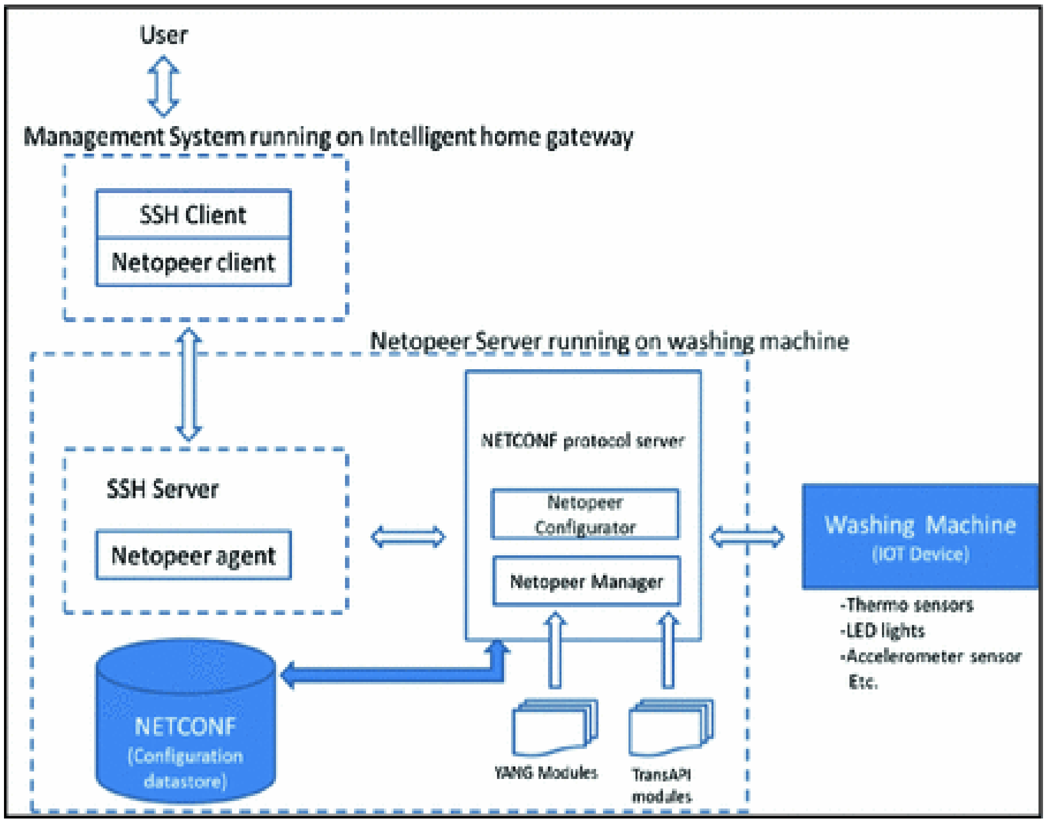


1) **Management System :** The operator uses a management system to send NETCONF messages to configure the IoT device and receives state information and notifications from the device as NETCONF messages.
2) **Management API :** allows management application to start NETCONF sessions.
3) **Transaction Manager:** executes all the NETCONF transactions and ensures that ACID properties hold true for the transactions.
4) **Rollback Manager:** is responsible for generating all the transactions necessary to rollback a current configuration to its original state.
5) **Data Model Manager :** Keeps track of all the YANG data models and the corresponding managed objects. Also keeps track of the applications which provide data for each part of a data model.
6) **Configuration Validator:** checks if the resulting configuration after applying a transaction would be a valid configuration.
7) **Configuration Database :** contains both configuration and operastional data.

8) **Configuration API :** Using the configuration API the application on the IoT device can be read configuration data from the configuration datastore and write operational data to the operational datastore.
9) **Data Provider API:** Applications on the IoT device can register for callbacks for various events using the Data Provider API. Through the Data Provider API, the applications can report statistics and operational data.

NETOPEER

Netopeer is set of opensource NETCONF tools built on the Libnetconf library



- **Netopeer-server**: is a NETCONF protocol server that runs on the managed device, this server provides an environment for configuring the device using NETCONF RPC operations and also retrieving the state data from the device.
- **Netopeer-agent**: is a NETCONF protocol agent running as a SSH/TLS subsystem. This agent accepts incoming NETCONF connection and passes the NETCONF operations received from the NETCONF-client to the NETCONF-server.
- **Netopeer-cli**: is a NETCONF client that provides the command line interface for interacting with the Netopeer-server. The operator uses Netopeer-cli to from the management system to send NETCONF RPC operations for configuring the device and retrieving the state information.
- **Netopeer-manager**: allows managing the YANG and Libnetconf Transaction API modules on the

Netopeer-server.

- **Netopeer- configurator**: is a tool that can be used to configure the Netopeer-server.

Steps for IoT device Management with NETCONF-YANG

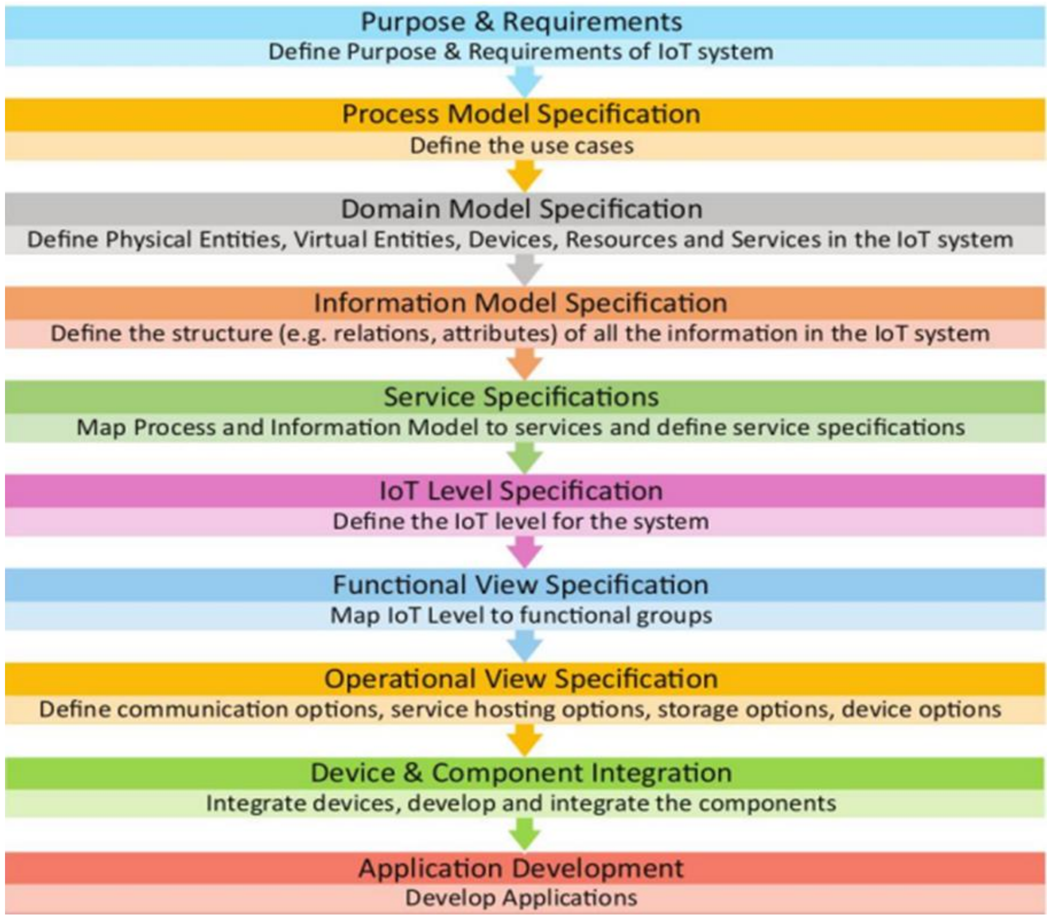
1) Create a YANG model of the system that defines the configuration and state data of the system.
2) Complete the YANG model with the _Inctool' which comes with Libnetconf.
3) Fill in the IoT device mangement code in the TransAPI module.
4) Build the callbacks C file to generate the library file.
5) Load the YANG module and the TransAPI module into the Netopeer server using Netopeer manager tool.
6) The operator can now connect from the management system to the Netopeer server using the NetopeerCLI.
7) Operator can issue NETCONF commands from the Netopeer CLI. Command can be issued to change the configuration data, get operational data or execute an RPC on the IoT device.

UNIT- III
IOT Platform Design Methodology

Designing IoT systems can be a complex and challenging task as these systems involve interactions between various components. A wide range of choices are available for each component. IoT designers often tend to design the system keeping specific products in mind.

We will look at a generic design methodology which is independent of specific product, service or programming language. IoT systems designed with this methodology will have reduced design time, testing time, maintenance time, complexity and better interoperability.

The steps involved in the designing of an IoT system or application can be summarized as shown in the below figure:


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
First step is to define the purpose and requirements of the system. In this step, the system purpose, behavior

and requirements are captured. Requirements can be:

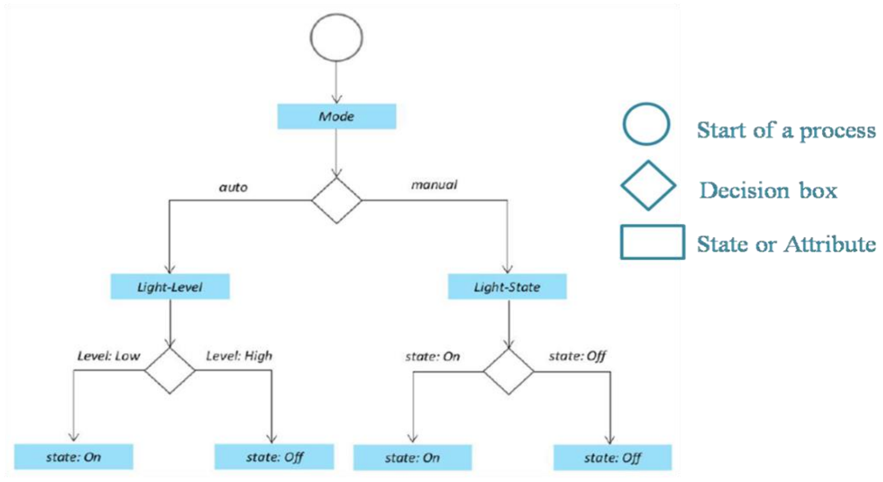
- Data collection requirements
- Data analysis requirements
- System management requirements
- Security requirements
- User interface requirements

For home automation system the purpose and requirements specification is as follows:

Purpose	A home automation system that allows controlling the lights remotely using a web application
Behavior	Home automation system should support two modes: auto and manual **Auto:** System measures the light level in the room and switches on the light when it is dark **Manual:** Allows remotely switching lights on and off
System Management	System should provide remote monitoring and control functions
Data Analysis	System should perform local analysis of the data
Application Deployment	Application should be deployed locally, but should be accessible remotely
Security	Should provide basic security like user authentication

Step 2: Process Specification

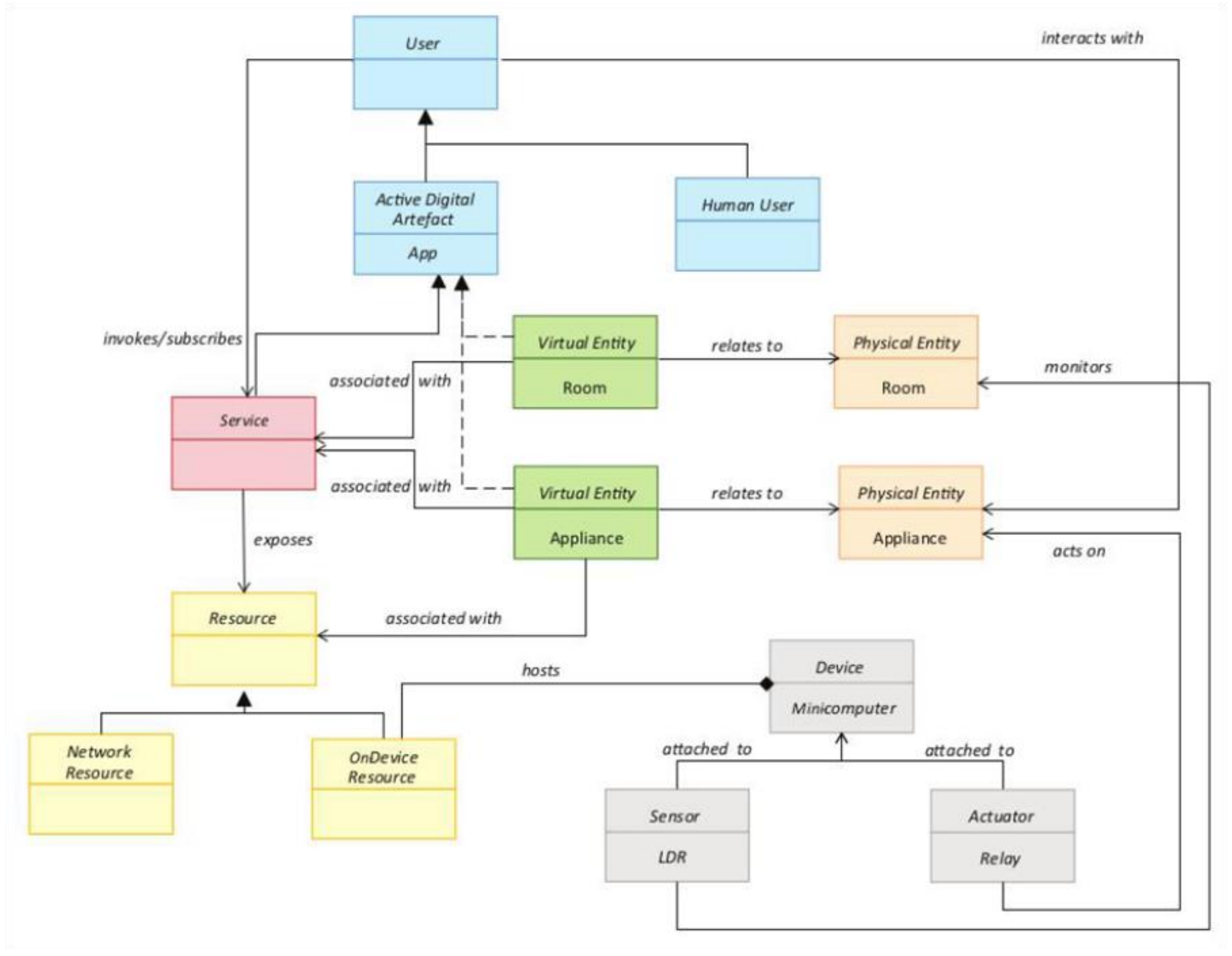
This step, formally described the purpose and requirement specifications of the IoT system. In a process diagram, the circle denotes the start of a process, the diamond denotes a decision box and the rectangle denotes a state or attribute (design flow chart).

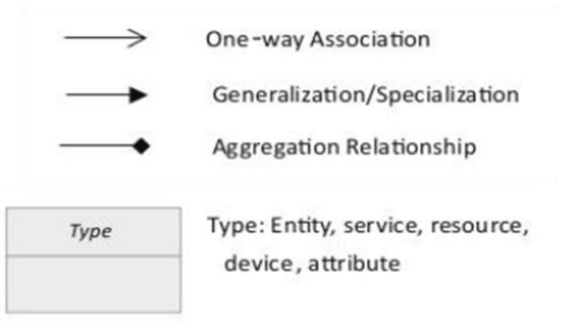


Step 3: Domain Model Specification

The domain model describes the main concepts, entities and objects in the domain of the IoT system to be designed. Domain model defines the attributes of the objects and relationships between objects. The domain model is independent of any specific technology or platform.

Using domain model, 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 of home automation system include the following:



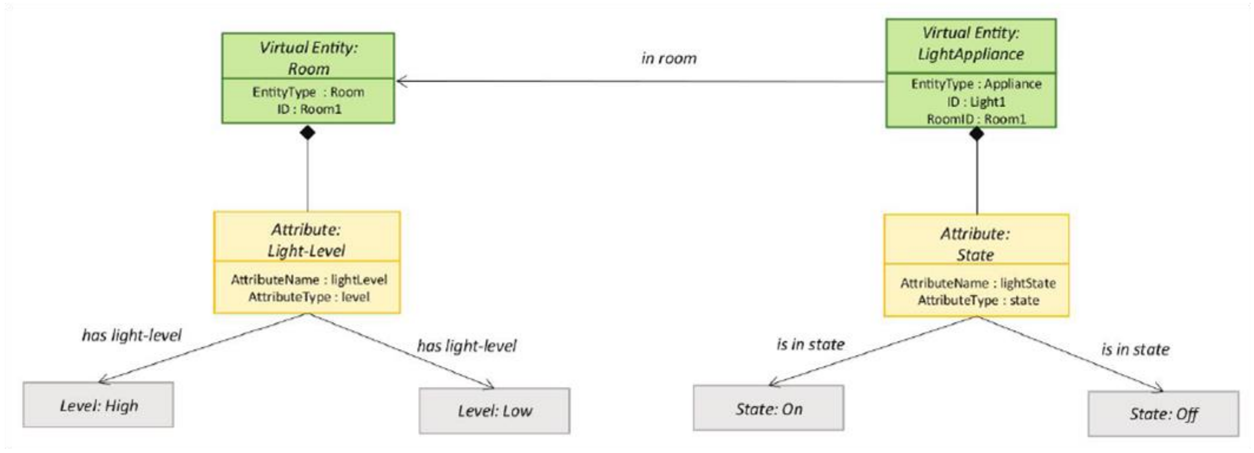

One-way Association
Generalization/Specialization
Aggregation Relationship
Type
Type: Entity, service, resource, device, attribute

Physical Entity	• The physical identifiable objects in the environment • IoT system provides information about the physical entity (using sensors) or performs actuation upon the physical entity
Virtual Entity	• Virtual entity is a representation of the physical entity in the digital world

	• For every physical entity there is a virtual entity
Device	• Devices provide a medium for interaction between physical and virtual entities • Devices are used to gather information from or perform actuation on physical entities
Resource	• Resources are software components which can be either on-device or network-resources • On-device resources are hosted on the device and provide sensing or actuation (eg: operating system) • Network-resources include software components that are available on the network (eg: database)
Service	• Services provide an interface for interacting with the physical entity • Services access resources to perform operations on physical entities

Step 4: Information Model Specification

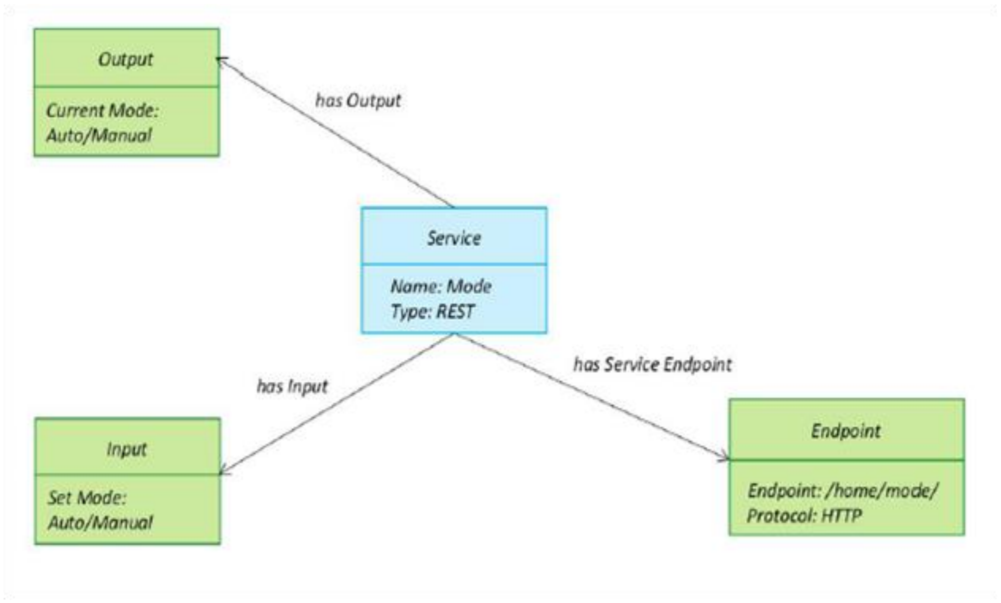
This step defines Information Model. The Information Model defines the structure of all the information in the IoT system, example, attributes of Virtual Entities, relations, etc. The information model does not describe "how the information is represented or stored". The information model defines the list of the Virtual Entities, their attributes, and the relations of the domain model.

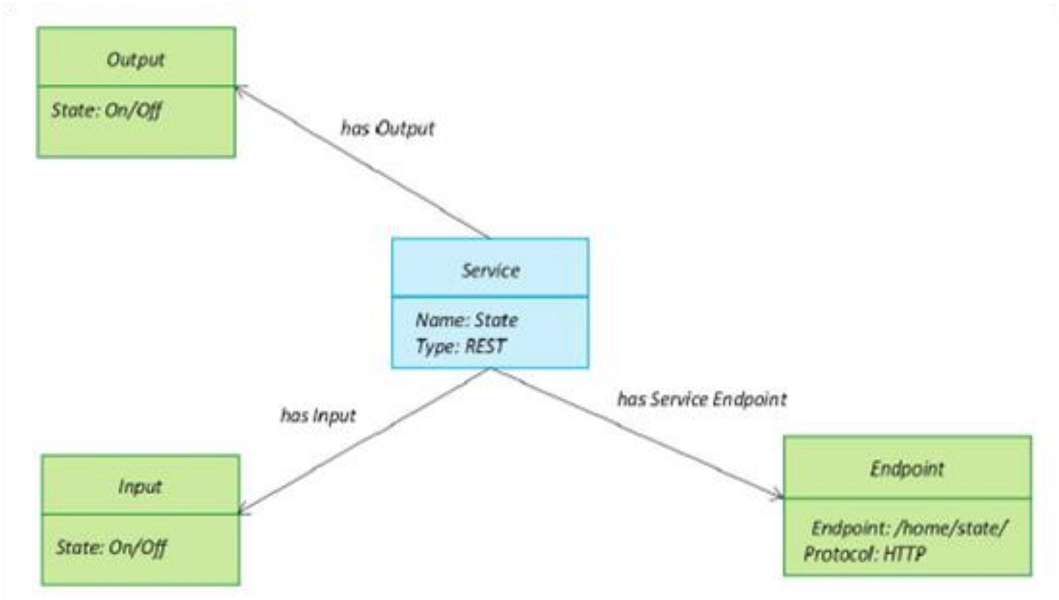


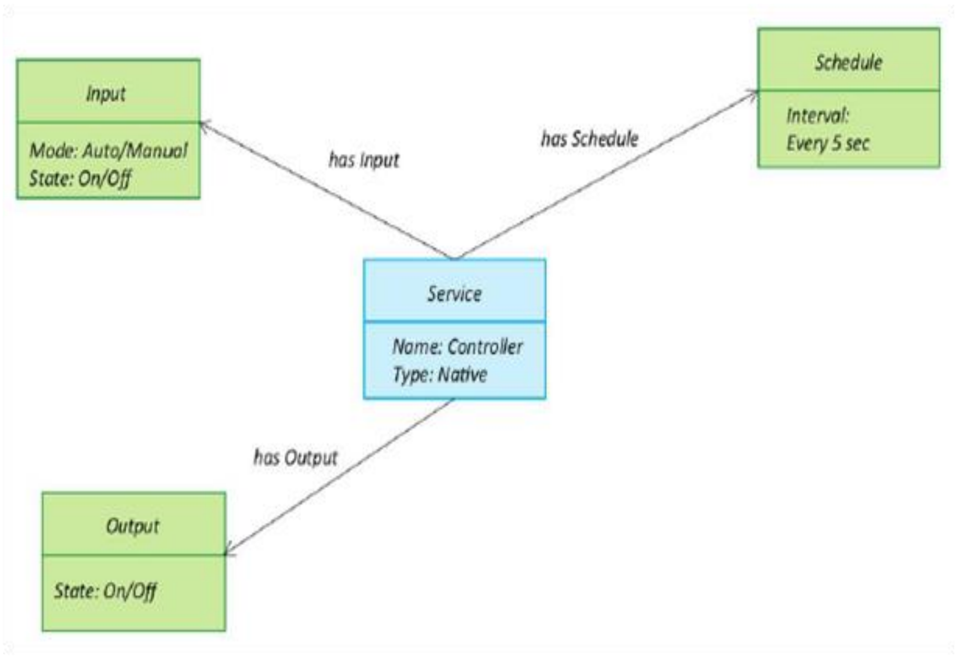
Step 5: Service Specifications

In this step, Service specifications define the services in the IoT system such as service types, service inputs/output, service endpoints, service schedules, service preconditions, and service effects. These services either change the state or attribute values or retrieve the current values. The Mode service is a RESTful web service that sets the mode to auto or manual (PUT request), or retrieves the current mode

(GET request). The mode is updated to/retrieved from the database. The State service is a RESTful web service that sets the light appliance state to on/off (PUT request) or retrieves the current light state (GET request). The state is updated to/retrieved from the status database. The Controller service runs as a native service on the device.

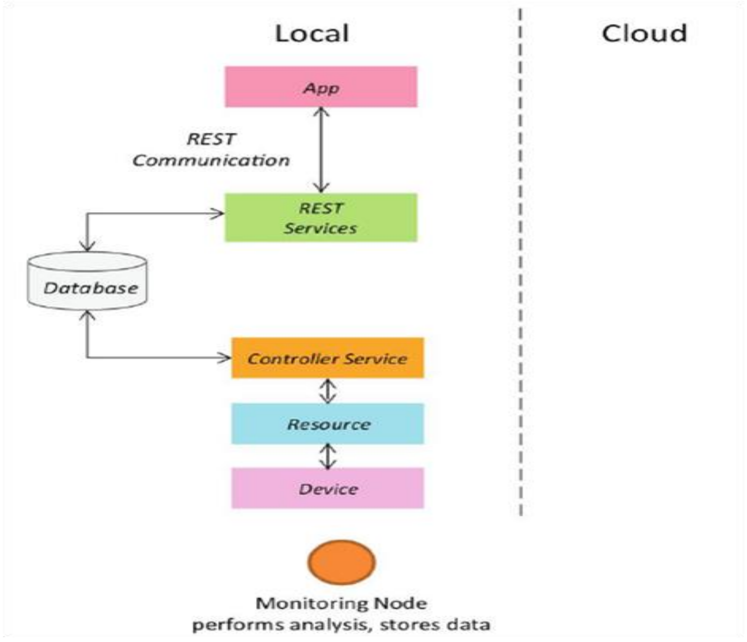






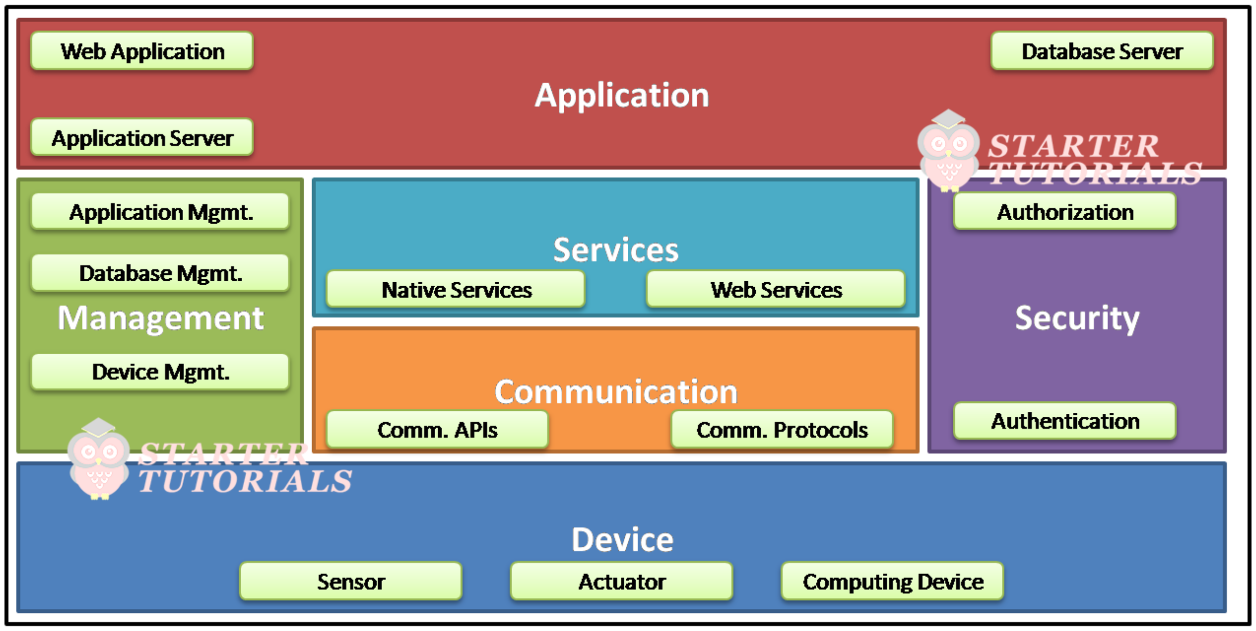
Step 6: IoT Level Specification

In this step define the IoT level for the system. Five types of IoT deployment levels are used according to different conditions.



Step 7: Functional View Specification

In the seventh step define the Functional View. The Functional View (FV) defines the functions of the loT systems grouped into various Functional Groups (FGs). Each Functional Group either provides functionalities for interacting with instances of the Domain Model. A Functional View includes:



· **Device:**

The device FG contains devices for monitoring and control.

· **Communication:**

The communication FG handles the communication protocols and APIs (such as REST and WebSocket) that are used by the services and applications to exchange data over the network. for the IoT system.

These are the backbone of IoT systems and enable network connectivity.

· **Services:**

The service FG includes various services involved in the IoT system such as services for device monitoring, device control services, data publishing services, and services for device discovery.

· **Management:**

The management FG includes all functionalities that are needed to configure and manage the loT system.

· **Security:**

The security FG includes security mechanisms for the loT system such as authentication, authorization, data security, etc.

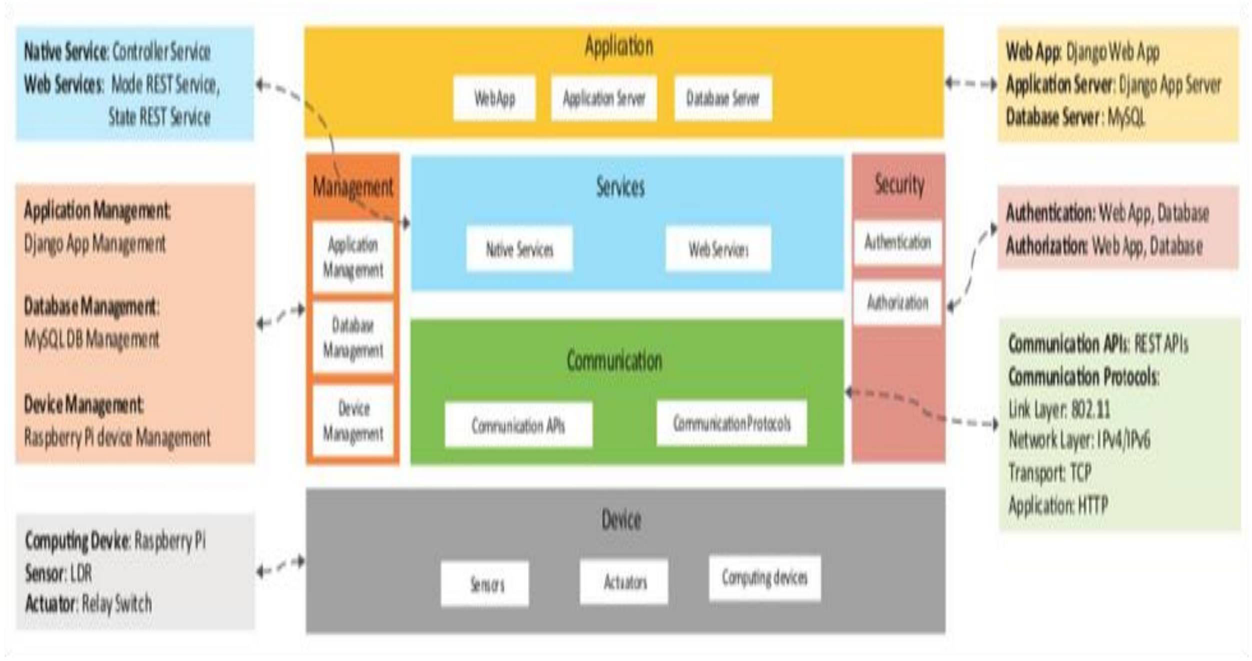
· **Application:**

The application FG includes applications that provide an interface for the users to control and monitor various aspects of the loT system.

Applications also allow users to view the system status and the processed data.

Step 8: Operational View Specification

In this step define the Operational View Specifications. IoT system deployment and operation are defined, such as service hosting options, storage options, device options, application hosting options, etc.



• Devices:

The computing device (Raspberry Pi), light-dependent resistor (sensor), relay switch (actuator). . Communication APIS: REST APIs

• Communication Protocols:

Link Layer - 802.11. Network Layer-IPv4/IPv6, Transport -TCP, Application - HTTP.

• Services:

1. Controller Service - Hosted on the device, implemented in Python, and run as a native service.

2. Mode service - REST-ful web service, hosted on a device, implemented with Django-REST Framework.

3. State service - REST-ful web service, hosted on a device, implemented with Django-REST Framework.

Application:

Web Application - Django Web Application, Application Server - Django App Server, Database Server - MySQL.

• **Security:**

Authentication: Web App, Database

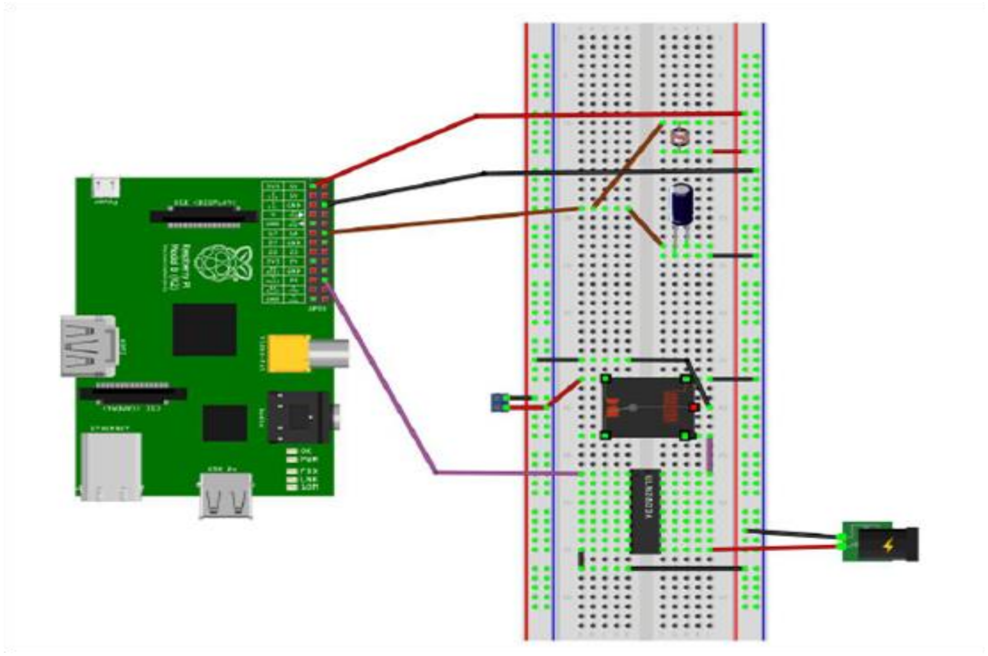
• **Management:**

Application Management - Django App Management

Database Management - MySQL DB Management, Device Management - Raspberry Pi device Management.

Step 9: Device & Component Integration

In this step integration of the devices and components design such as minicomputer, LDR sensor, and relay switch actuator.

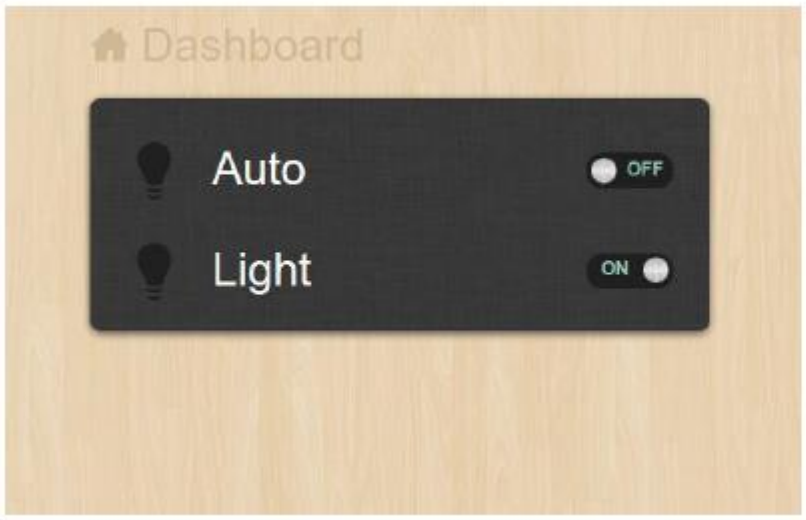


Step 10: Application Development

The final step in the IoT design methodology.

It is to develop the IoT application.

The application has controls for the mode (auto-on or auto-off) and the light (on or off).

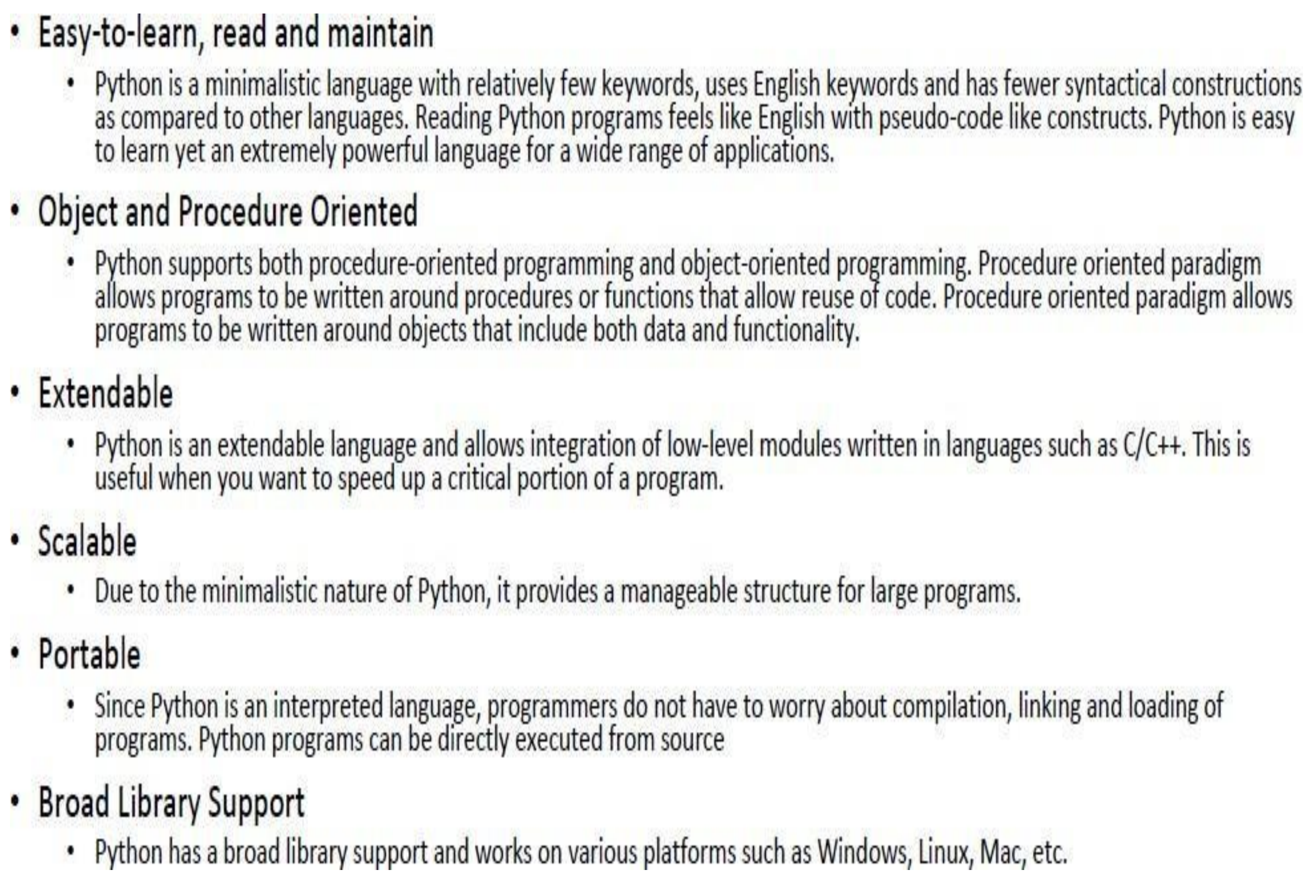


Introduction to Python

Python is a general-purpose high level programming language and suitable for providing a solid foundation to the reader in the area of cloud computing.

The main characteristics of Python are:

1) Multi-paradigm programming language.
2) Python supports more than one programming paradigms including object- oriented programming and structured programming.
3) Interpreted Language.
4) Python is an interpreted language and does not require an explicit compilationstep.
5) The Python interpreter executes the program source code directly, statement by statement, as a processor or scripting engine does.
6) Interactive Language
7) Python provides an interactive mode in which the user can submit commands at the Python prompt and interact with the interpreter directly.



- Easy-to-learn, read and maintain
  - Python is a minimalistic language with relatively few keywords, uses English keywords and has fewer syntactical constructions as compared to other languages. Reading Python programs feels like English with pseudo-code like constructs. Python is easy to learn yet an extremely powerful language for a wide range of applications.
- Object and Procedure Oriented
  - Python supports both procedure-oriented programming and object-oriented programming. Procedure oriented paradigm allows programs to be written around procedures or functions that allow reuse of code. Procedure oriented paradigm allows programs to be written around objects that include both data and functionality.
- Extendable
  - Python is an extendable language and allows integration of low-level modules written in languages such as C/C++. This is useful when you want to speed up a critical portion of a program.
- Scalable
  - Due to the minimalistic nature of Python, it provides a manageable structure for large programs.
- Portable
  - Since Python is an interpreted language, programmers do not have to worry about compilation, linking and loading of programs. Python programs can be directly executed from source
- Broad Library Support
  - Python has a broad library support and works on various platforms such as Windows, Linux, Mac, etc.

Data types

Every value in Python has a data type. Since everything is an object in Python programming, datatypes are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

Python Numbers

Integers, floating point numbers and complex numbers fall under Python numbers category. They are defined as int, float and complex class in Python. We can use the type() function to know which class a variable or a value belongs to and the is instance() function to check if an object belongs to a particular class.

Script.py

```
1. a = 5
2. print(a, "is of type",
type(a))3. a = 2.0
4. print(a, "is of type",
type(a))5. a = 1+2j
6. print(a, "is complex number?", isinstance(1+2j,complex))
```

Integers can be of any length, it is only limited by the memory available. A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number. Complex numbers are written in the form, x +yj, where x is the real part and y is the imaginary part. Here are some examples.

```
>>> a = 1234567890123456789

>>> a
1234567890123456789
>>> b = 0.1234567890123456789

>>> b
0.12345678901234568
>>> c = 1+2j

>>> c
(1+2j)
```

Python List

List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type. Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [].

```
>>> a = [1, 2.2, 'python']
```

We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.

Script.py

```
1. a = [5,10,15,20,25,30,35,40]
2. # a[2] = 15
3. print("a[2] = ", a[2])
4. # a[0:3] = [5, 10, 15]
5. print("a[0:3] = ", a[0:3])
6. # a[5:] = [30, 35, 40]
7. print("a[5:] = ", a[5:])
```

Lists are mutable, meaning; value of elements of a list can be altered.

```
>>> a = [1,2,3]
>>> a[2]=4
>>>    a
[1, 2, 4]
```

Python Tuple

Tuple is an ordered sequences of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically. It is defined within parentheses () where items are separated by commas.

```
>>> t = (5,'program', 1+3j)
```

Script.py

```
t = (5,'program', 1+3j)
#   t[1]   =   'program'
print("t[1] = ", t[1])
# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])
# Generates error
# Tuples are immutable
t[0] = 10
```

Python Strings

String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

```
>>> s = "This is a string"
>>> s = '''a multiline
```

Like list and tuple, slicing operator [ ] can be used with string. Strings are immutable.
Script.py

```
a ={5,2,3,1,4}
# printing set variableprint("a = ", a)
# data type of variable a
print(type(a))
```

We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates. Since, set are unordered collection, indexing has no meaning. Hence the slicing operator [] does not work. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value. In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of anytype.

```
>>> d = {1:'value','key':2}

>>> type(d)

<class 'dict'>
```

We use key to retrieve the respective value. But not the other way around.

Script.py

```
d      ={1:'value','key':2}
print(type(d))
print("d[1] = ",d[1]);
print("d['key'] = ", d['key']);
#        Generates        error
print("d[2] = ",d[2]);
```

Python if...else Statement

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes. Decision making is required when we want to execute a code only if a certain condition is satisfied.

The if…elif…else statement is used in Python for decision making.

Python if Statement

Syntax

```
if test expression:
   statement(s)
```

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True.
If the text expression is False, the statement(s) is not executed. In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end. Python interprets non-zero values as True. None and 0 are interpreted as False.

Python if Statement Flowchart

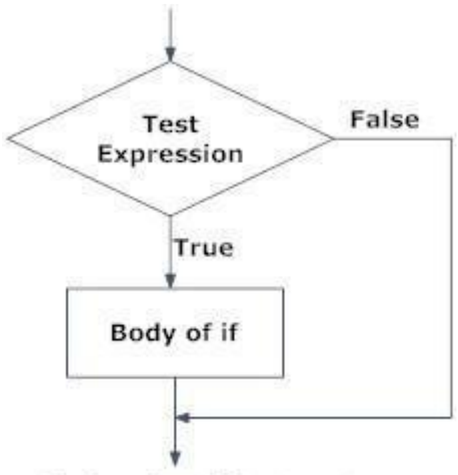


Fig: Operation of if statement

Example: Python if Statement

# If the number is positive, we print an appropriate

messagenum = 3
if num > 0:
   print(num, "is a positive number.") print("This is always

printed.")
num = -1
if num >0:
print(num, "is a positive number.") print("This is also always printed.")

When you run the program, the output will be: 3 is a positive numberThis is always sprinted This is also always printed.

In the above example, num > 0 is the test expression. The body of if is executed only if this evaluates to True.
When variable num is equal to 3, test expression is true and body inside body of if is executed. If variable num is equal to -1, test expression is false and body inside body of if is skipped. The print() statement falls outside of the if block (unindented). Hence, it is executed regardless of the test expression.

Python if...else Statement

Syntax

```
if test
   expression:
   Body of if
else:
   Body of else
```

The if..else statement evaluates test expression and will execute body of if only when test condition is True.
If the condition is False, body of else is executed. Indentation is used to separate the blocks.

Python if..else Flowchart

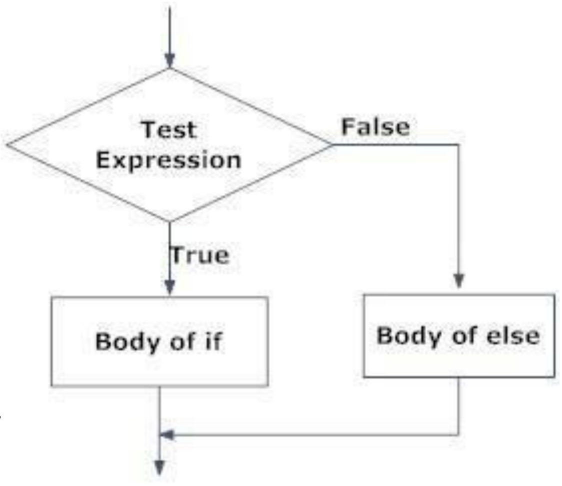


Fig: Operation of if...else statement

Example of if...else

```
# Program checks if the number is positive or
negative# And displays an appropriate message
num = 3
# Try these two variations as
well.# num = -5
# num = 0
if      num      >=      0:
   print("Positive        or
   Zero")
else:
   print("Negative number")
```

In the above example, when num is equal to 3, the test expression is true and body of if is executed and body of else is skipped.

If num is equal to -5, the test expression is false and body of else is executed and body of if is skipped.

If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped.

Python if...elif...else Statement

Syntax

```
if test expression:
   Body of if
elif test expression:
   Body of elif
else:
   Body of else
```

The elif is short for else if. It allows us to check for multiple expressions. If the condition for if is False, it checks the condition of the next elif block and so on. If all the conditions are False, body of else is executed. Only one block among the several if...elif...else blocks is executed according to the condition. The if block can have only one else block. But it can have multiple elif blocks.

Flowchart of if...elif...else

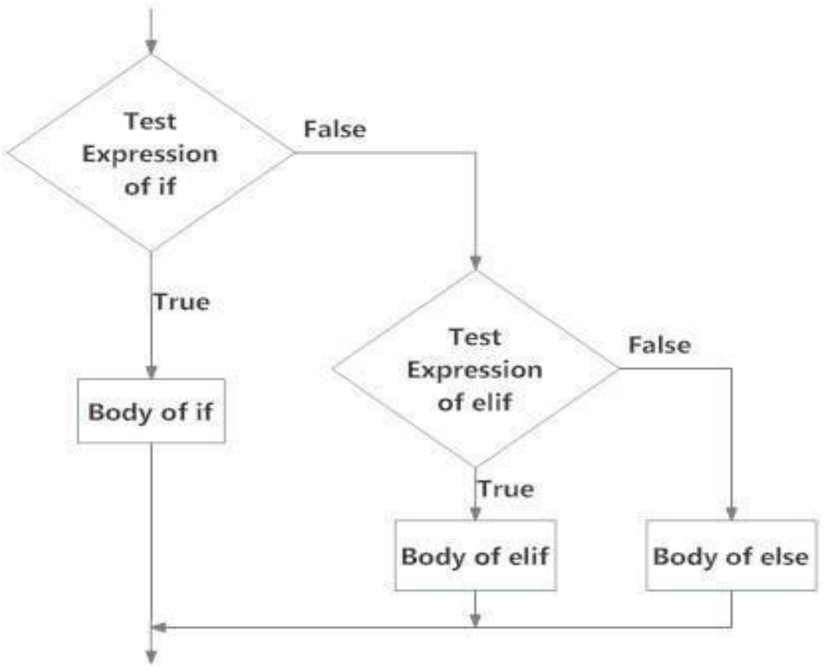


Fig: Operation of if...elif...else statement

Example of if...elif...else

```
# In this program,
# we check if the number is positive
or# negative or zero and
# display an appropriate
messagenum = 3.4
# Try these two variations as well:
# num = 0
# num = -
4.5 if num >
0:
   print("Positive
number")elif num == 0:
   print("Zero
")else:
   print("Negative number")
```

When variable num is positive, Positive number is printed.

If num is equal to 0, Zero is printed.
If num is negative, Negative number is printed

Python Nested if statements

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming. Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

Python Nested if Example

```
# In this program, we input a number
# check if the number is positive or
# negative or zero anddisplay
# an appropriate message
# This time we use nested if

num = float(input("Enter a number: "))
if num >= 0:
   if  num  ==  0:
      print("Zero")
   else:
      print("Positive number")
else:
   print("Negative number")
```

Output 1

```
Enter a number: 5
Positive   number
```

Output 2

```
Enter a number: -1
Negative number
```

Output 3

```
Enter a number: 0
Zero
```

Python for Loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

```
for val in sequence:
Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.
Flowchart of for Loop

Syntax

```
# Program to find the sum of all numbers stored in a
list# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
# variable to store the
sumsum = 0

# iterate over the
list for val in
numbers:
        sum = sum+val
# Output: The sum is
48 print("The sum is",
sum)
```

when you run the program, the output will be:
The sum is 48

The range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers). We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided. This function does not store all the values in emory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on thego.
To force this function to output all the items, we can use the function list().
The following example will clarify this.

```
# Output: range(0, 10)
print(range(10))
# Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(list(range(10)))
# Output: [2, 3, 4, 5, 6, 7]
print(list(range(2, 8)))
# Output: [2, 5, 8, 11, 14, 17]
print(list(range(2, 20, 3)))
```

We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the len() function to iterate though a sequence using indexing. Here is an example.

```
# Program to iterate through a list using indexing
genre = ['pop', 'rock', 'jazz']
# iterate over the list using index
for i in range(len(genre)):
        print("I like", genre[i])
```

When you run the program, the output will be:

```
I  likepop
I likerock
I likejazz
```

What is while loop in Python?

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true. We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax of while Loop in Python

while test_expression:Body of while

In while loop, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False. In Python, the body of the while loop is determined through indentation. Body starts with indentation and the first unindented line marks the end. Python interprets any non-zero value as True. None and 0 are interpreted asFalse.

Flowchart of while Loop

```
# Program to add
natural # numbers
upto
# sum = 1+2+3+...+n
# To take input from the user,# n = int(input("Enter n: "))
n = 10
# initialize sum and countersum = 0
i = 1
while i <= n: sum = sum + i
  i=i+1   #
updatecounter # print
thesum
```

print("The sum is", sum)
When you run the program, the output will be:
Enter n: 10 The sum is 55

In the above program, the test expression will be True as long as our counter variable i is less than orequal to n (10 in ourprogram).
We need to increase the value of counter variable in the body of the loop. This is very important (andmostly forgotten). Failing to do so will result in an infinite loop (never ending loop).
Finally the result is displayed.

Python Modules

A file containing a set of functions you want to include in the application is called Module.

Create a Module

To create a module just save the code you want in a file with the file extension .py:

Example

Save this code in a file named mymodule.py

```
def greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the import statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule
mymodule.greeting("Jonathan")
```

Note: When using a function from a module, use the syntax: module_name.function_name.
Variables in Module

The module can contain functions, as already described, but also variables of all types(arrays, dictionaries, objects etc):

Example

Save this code in the file mymodule.py

person1 = {"name": "John","age": 36,"country": "Norway"}

Example

Import the module named mymodule, and access the person1 dictionary:

import mymodule
a = mymodule.person1["age"]
print(a)

Naming a Module

You can name the module file whatever you like, but it must have the file extension .py

Re-naming a Module

You can create an alias when you import a module, by using the as keyword:

Example

Create an alias for mymodule called mx:
import mymodule as mxa = mx.person1["age"] print(a)

Built-in Modules

There are several built-in modules in Python, which you can import whenever you like.

Example

Import and use the platform module:
import platform
x = platform.system()print(x)

Using the dir() Function

There is a built-in function to list all the function names (or variable names) in a module. The dir() function:

Example

List all the defined names belonging to the platform module:import platform

x = dir(platform)
print(x)

Note: The dir() function can be used on all modules, also the ones you create yourself.

Import from Module

You can choose to import only parts from a module, by using the from keyword.

Example

The module named mymodule has one function and one dictionary:
def greeting(name):
print("Hello, " + name)
person1 = {"name": "John", "age": 36, "country": "Norway"}

Example

Import only the person1 dictionary from the module:
from mymodule import person1
print (person1["age"])

Note: When importing using the from keyword, do not use the module name when referring to elements in the module. Example: person1["age"], not mymodule.person1["age"].

Packages

We don't usually store all of our files in our computer in the same location. We use a well-organized hierarchy of directories for easier access. Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and modules for files. As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear.

Similar, as a directory can contain sub-directories and files, a Python package can have sub-packages and modules. A directory must contain a file namedinit.py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file. Here is an example. Suppose we are developing a game, one possible organization of packages and modules could be as shown in the figure below.

Package Module Structure in Python Programming
Importing module from a package

We can import modules from packages using the dot (.) operator. For example, if want to import the start module in the above example, it is done as follows.

import Game.Level.start

Now if this module contains a function named select_difficulty(), we must use the full name to reference it.

Game.Level.start.select_difficulty(2)

If this construct seems lengthy, we can import the module without the package prefix as follows.
from Game.Level import start

We can now call the function simply as follows.

start.select_difficulty(2)

Yet another way of importing just the required function (or class or variable) form a module within a package would be as follows.

from Game.Level.start

import select_difficultyN

ow we can directly call this function.

select_difficulty(2)

Although easier, this method is not recommended. Using the full namespace avoids confusion and prevents two same identifier names from colliding. While importing packages, Python looks in the list of directories defined in sys.path, similar as for module search path.

Files

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk). Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data. When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed. Hence, in Python, a file operation takes place in the followingorder.

1. Open afile
2. Read or write (perform operation)

3. Close thefile

How to open a file?

Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

>>> f=open("test.txt") # open file in currentdirectory
>>> f = open("C:/Python33/README.txt") # specifying full path

We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode. The default is reading in text mode. In this mode, we get strings when reading from the file. On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

Python File Modes

Mode	**Description**
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

f=open("test.txt") # equivalent to 'r' or
'rt' f = open("test.txt",'w') # write in
textmode
f = open("img.bmp",'r+b') # read and write in binary mode

Unlike other languages, the character 'a' does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings). Moreover, the default encoding is platform dependent. In windows, it is 'cp1252' but 'utf-8' in Linux. So, we must not also rely on the default encoding or else our code will behave differently in different platforms. Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

```
f = open("test.txt",mode = 'r',encoding = 'utf-8')
```

How to close a file Using Python?

When we are done with operations to the file, we need to properly close the file. Closing a file will free up the resources that were tied with the file and is done using Python close() method. Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

```
f = open("test.txt",encoding = 'utf-8')
#      perform      file
operationsf.close()
```

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.

A safer way is to use a try...finally block.

```
try:
  f = open("test.txt",encoding = 'utf-
  8')# perform file operations
finally:
  f.close()
```

This way, we are guaranteed that the file is properly closed even if an exception is raised, causing program flow to stop. The best way to do this is using the with statement. This ensures that the file is closed when the block inside with is exited. We don't need to explicitly call the close() method. It is doneinternally.

```
with open("test.txt",encoding = 'utf-8') as f:
  # perform file operations
```

How to write to File Using Python?

In order to write into a file in Python, we need to open it in write 'w', append 'a' or exclusive creation 'x' mode. We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased. Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

```
with open("test.txt",'w',encoding = 'utf-8') as f:
  f.write("my     first     file\n")
  f.write("This           file\n\n")
  f.write("contains           three
  lines\n")
```

This program will create a new file named 'test.txt' if it does not exist. If it does exist, it is overwritten. We must include the newline characters ourselves to distinguish different lines.

How to read files in Python?

To read a file in Python, we must open the file in reading mode. There are various methods available for this purpose. We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
>>> f.read(4) # read the first 4 data
'This'

>>>f.read(4)   #  read  the  next  4
data' is'

>>>f.read()    # read in the rest till end of file'my first file\nThis file\ncontains threelines\n'

>>> f.read() # further reading returns empty
sting''
```

We can see that, the read() method returns newline as '\n'. Once the end of file is reached, we get empty string on further reading. We can change our current file cursor (position) using the seek() method. Similarly, the tell() method returns our current position (in number of bytes).

```
>>>f.tell()   #   get   the   current   file
position56

>>>  f.seek(0)  #  bring  file  cursor  to  initial
position0

>>> print(f.read()) # read the entire fileThis is my first file
This file
contains three lines
```

We can read a file line-by-line using a for loop. This is both efficient and fast.

```
>>> for line in f:
```

... print(line, end = '')
...
This is my first fileThis file
contains three lines
The lines in file itself has a newline character '\n'.

Moreover, the print() end parameter to avoid two newlines when printing. Alternately, we can use readline() method to read individual lines of a file. This method reads a file till the newline, including the newlinecharacter.

>>> f.readline()
'This is my first file\n'

>>> f.readline()'This file\n'
>>> f.readline() 'contains three lines\n'

>>> f.readline()''
Lastly, the readlines() method returns a list of remaining lines of the entire file. All these reading method return empty values when end of file (EOF) is reached.

>>> f.readlines()

['This is my first file\n', 'This file\n', 'contains three lines\n']

Python File Methods

There are various methods available with the file object. Some of them have been used in above examples. Here is the complete list of methods in text mode with a brief description.

Python File Methods

Method	**Description**
close()	Close an open file. It has no effect if the file is already closed.
detach()	Separate the underlying binary buffer from the TextIOBase and return it.
fileno()	Return an integer number (file descriptor) of the file.
flush()	Flush the write buffer of the file stream.
isatty()	Return True if the file stream is interactive.
read(n)	Read at most n characters form the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.
readline(n=-1)	Read and return one line from the file. Reads in at most n bytes if specified.
readlines(n=-1)	Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.

seek(offset,from=SEEK_SET)	Change the file position to offset bytes, in reference to from (start, current, end).
seekable()	Returns True if the file stream supports random access.
tell()	Returns the current file location.
truncate(size=None)	Resize the file stream to size bytes. If size is not specified, resize to current location.
writable()	Returns True if the file stream can be written to.
write(s)	Write string s to the file and return the number of characters written.
writelines(lines)	Write a list of lines to the file.

Python packages of interest to IOT

JSON

A **JSON** file is a file that stores simple data structures and objects in JavaScript Object Notation (JSON) format, which is a standard data interchange format. It is primarily used for transmitting data between a web application and a server. A **JSON** object contains data in the form of a key/value pair. The keys are strings and the values are the JSON types. Keys and values are separated by a colon. Each entry (key/value pair) is separated by a comma. JSON files are lightweight, text-based, human-readable, and can be edited using a text editor.
Note: For more information, refer to Working With JSON Data in Python

XML

XML is a markup language which is designed to store data. It is case sensitive. XML offers you to define markup elements and generate customized markup language. The basic unit in the XML is known as an element. The XML language has no predefined tags. It simplifies data sharing, data transport, platform changes, data availability Extension of an XML file is .xml
Both JSON and XML file format are used for transferring data between client and server.

However, they both serve the same purpose though differ in their own way.

Urllib

Urllib package is the URL handling module for python. It is used to fetch URLs (Uniform Resource Locators). It uses the urlopen function and is able to fetch URLs using a variety of different protocols.
Urllib is a package that collects several modules for working with URLs, such as:

- urllib.request for opening and reading.
- urllib.parse for parsing URLs
- urllib.error for the exceptions raised
- urllib.robotparser for parsing robot.txt files

If urllib is not present in your environment, execute the below code to install it.

```
pip install urllib
```

urllib.request

This module helps to define functions and classes to open URLs (mostly HTTP). One of the most simple ways to open such URLs is :
urllib.request.urlopen(url)
urllib.parse
This module helps to define functions to manipulate URLs and their components parts, to build or break them. It usually focuses on splitting a URL into small components; or joining different URL components into URL strings.

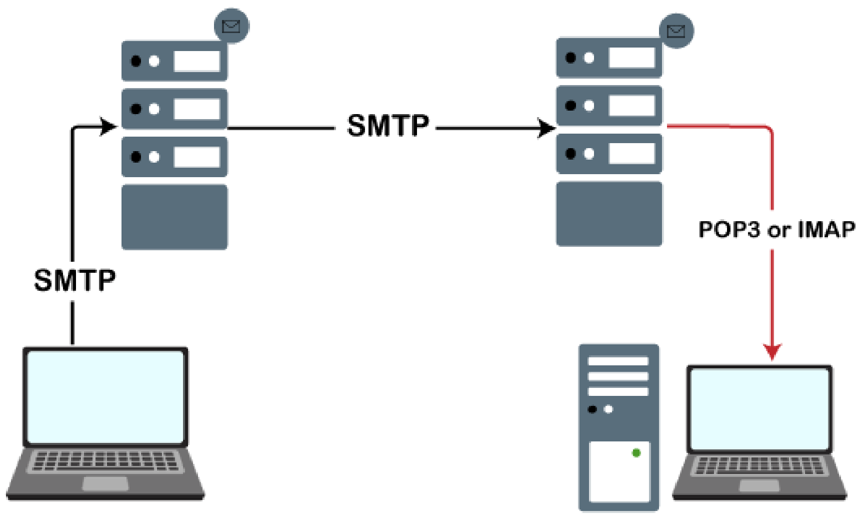
urllib.error
This module defines the classes for exception raised by urllib.request. Whenever there is an error in fetching a URL, this module helps in raising exceptions. The following are the exceptions raised :

- URLError – It is raised for the errors in URLs, or errors while fetching the URL due to connectivity, and has a 'reason' property that tells a user the reason of error.
- HTTPError – It is raised for the exotic HTTP errors, such as the authentication request errors. It is a subclass or URLError. Typical errors include '404' (page not found), '403' (request forbidden), and '401' (authentication required).

urllib.robotparser
This module contains a single class, RobotFileParser. This class answers question about whether or not a particular user can fetch a URL that published robot.txt files. *Robots.txt is a text file webmasters create to instruct web robots how to crawl pages on their website.* The robot.txt file tells the web scraper about what parts of the server should not be accessed.

SMTP

Simple Mail Transfer Protocol (SMTP) is used as a protocol to handle the email transfer using Python. It is used to route emails between email servers. It is an application layer protocol which allows to users to send mail to another. The receiver retrieves email using the protocols **POP(Post Office Protocol)** and **IMAP(Internet Message Access Protocol)**.



When the server listens for the TCP connection from a client, it initiates a connection on port 587.

Python provides a **smtplib** module, which defines an the SMTP client session object used to send emails to an internet machine. For this purpose, we have to import the **smtplib** module using the import statement.

```
$ import smtplib
```

The SMTP object is used for the email transfer. The following syntax is used to create the smtplib object.

```
import smtplib
smtpObj = smtplib.SMTP(host, port, local_hostname)
```

It accepts the following parameters.

- **host:** It is the hostname of the machine which is running your SMTP server. Here, we can specify the IP address of the server like (https://www.javatpoint.com) or localhost. It is an optional parameter.
- **port:** It is the port number on which the host machine is listening to the SMTP connections. It is 25 by default.
- **local_hostname:** If the SMTP server is running on your local machine, we can mention the hostname of the local machine.

The sendmail() method of the SMTP object is used to send the mail to the desired machine. The syntax is given below.

```
smtpObj.sendmail(sender, receiver, message)
```

There are cases where the emails are sent using the Gmail SMTP server. In this case, we can pass Gmail as the SMTP server instead of using the localhost with the port 587.

Use the following syntax.

```
$ smtpObj = smtplib.SMTP("gmail.com", 587)
```

UNIT-IV
Integrated Billing Solutions in the Internet of Things

INTRODUCTION

The Internet of Things is one of the most promising technological developments in information technology. It promises huge financial and nonfinancial benefits across supply chains, in product life cycle and customer relationship applications as well as in smart environments. However, the adoption process of the Internet of Things has been slower than expected. One of the main reasons for this is the missing profitability for each individual stakeholder. Costs and benefits are not equally distributed. Cost benefit sharing models have been proposed to overcome this problem and to enable new areas of application. However, these cost benefit sharing approaches are complex, time consuming, and have failed to achieve broad usage.

While RFID and other Auto-ID technologies continue to be major components of the Internet of Things, there are other technologies, such as sensors, actuators, and networked infrastructures that will further add to the ongoing cost discussion.

The major hurdles in the process of deploying the Internet of Things are

- Cost for hardware &Software,
- Integration,
- Maintenance,
- Business process reengineering
- Data analysis.

Some Internet of Things related applications may never come true, because some of the stakeholders would need to spend more on technology and integration than can be justified by internal benefits. There are several problematic aspects in cost benefit analysis and sharing:

- Detailed cost benefit analysis can be time consuming
- It is difficult to identify, measure and analyze all costs and benefits associated with an Internet of Things
- Companies are reluctant to share benefits

Cost benefit sharing models do not scale, as they are subject to bi-directional negotiations. An alternative solution to cost benefit sharing could be based on selling and buying information that is provided through the Internet of Things. For this, a billing solution is needed to price and bill information. Similar concepts are known from the telecommunications industry, where billing solutions are an integral part of the overall infrastructure, allowing billing of different services, such as

- Voice calls,

- SMS,
- Internet access and
- Premium services

across service providers and different countries.

Cost of RFID and the Internet of Things

There are numerous costs associated with the adoption of the Internet of Things. While the Internet of Things is not synonymous with RFID, results from cost analysis for RFID can be used as a basis for further calculations. Agarwal (2001) lists six different costs of RFID deployment for manufacturing firms

- The cost of the tag itself,
- Cost of applying tags to products,
- Cost of purchasing and installing tag readers in factories and/or warehouses,
- Systems integration costs,
- Cost of training and reorganization,
- Cost of implementing application solutions.

Feinbier et al. (2008) list relevant costs for RFID installation in detail, based on experiences in the steel industry. On the basis of both approaches, similar cost structures can be inferred for the Internet of Things

Cost level	**Cost of tagging (Agarwal 2001)**	**Cost considerations for RFID (Feinbier et al. 2008)**	**Cost of Internet of Things adoption**
1 Mobile devices	• Cost of the tag itself • Cost of applying tags to products	• Tags	• Cost of mobile technologies, such as data carriers (e.g., tags), sensors, actuators or smart devices • Cost of applying mobile technologies to things
2 Aggregation devices and software	• Cost of purchasing and installing tag readers in factories and/or warehouses	• Readers • Antenna and cabling • Installation • Tuning • Controllers • Software platform (middleware)	• Cost of purchasing edge devices (e.g., readers, gateways, controllers, accessories) and edgeware for fixed and/or mobile environments • Installation and technical optimization costs

3 Integration	• Systems integration costs	• Integration (to legacy systems)	• Systems integration costs including new interfaces as well as necessary updates, extensions, or replacement of existing systems
4 Training and reorganization	• Cost of training and reorganization	• Process (incl. redesign and human elements)	• Training cost • Reorganization / business reengineering / business model innovation
5 Application	• Cost of implementing application solutions		• Cost of implementing internal application solutions beyond existing applications
6 Networking (technical and organizational)			• Cost for networking in an open environment, including e.g., improved security, fine layered access control, multi-directional communication, product data contracts, service level agreements, standardized syntax and semantics, data conversion, synchronization, trust concepts
7 Operational		• Maintenance	• Cost for maintenance • Other operational costs for running (e.g., data storage and analysis), extending and improving the system

Benefits of RFID and the Internet of Things

Baars et al. (2008) have identified four different approaches towards systemization of RFID benefits

➢ **Collecting and grouping** –
Benefits are collected and grouped.

➢ **Layer of impact**
Benefits are structured to impact layers such as short term and long term automation, informational and transformational benefits, proven or potential.

➢ **Locus of impact**

These studies highlight who benefits, thus it automatically considers benefits to multiple stakeholders.

➢ **Indicator system**

Established evaluation systems, such as Balanced Score cards are used to structure RFID benefits. Collective benefits can be achieved by all of these stakeholders. These include:

• **Reduced product shrinkage**: reduction of loss of goods through misplacement, spoilage, and theft

• **Improved information sharing**: product related data may be exchanged to benefit multiple stakeholders, problems resulting from converting paper-based information to digital information are avoided and manual data-entry is drastically reduced

• **Compensatory benefits**: benefits provided through other stakeholders, including for example cost benefit sharing, funded research, bonus payments, vouchers, information (e.g. sales data)

Companies in general may benefit from:

• **Increased inventory, shipping and data accuracy**: e.g., differences between real stock numbers and assumed stock, based on false data

• **Subsequent fault reduction**: inaccurate and incomplete visibility may lead to false decisions and can be avoided through the Internet of Things

• **Faster exception management (agility):** capability of responding to unplanned events in a timely manner before critical problems escalate

• **Asset management**: better asset utilization may lead to an opportunity to reduce asset inventory, reduced asset shrinkage, better shipment consolidation, reduced energy consumption and improved reverse logistics.

• **Product rotation**: methods of inventory control, such as First In, First Out (FIFO) can be used more accurately to ensure efficient stock rotation e.g. in time sales for perishable goods.

Manufacturers and suppliers benefit mainly from:

• **Production tracking**: tracking of raw material, work-in-progress inventory, assembly status tracking and finished products

• **Quality control**: ensured quality control in production

• **Supply / production continuity**: enabled through improved material tracking

• **Compliance:** e.g., in case of mandates issued for example by large retailers or legislators and regulators

Distributor and logistics provider as well as internal distribution and logistics departments benefit from:

• **Material handling**: time (labor) savings for loading / unloading of trucks, administrative overhead at the goods receipt , cross-docking , customs clearance delivery lead times and reduced delays, faster inventory, goods receiving, loading and unloading as well as reduced human errors through Auto-ID

• **Space utilization**: achieved through reduced buffers and reduction of product storage incompatibilities (e.g., placement of hazardous goods102 Retailer benefits include:), based on better data accuracy through RFID usage

Retailer benefits include: based on better data accuracy through RFID usage

• **Customer service**: RFID can be used to simplify checkouts and payments as well as for promotion management

• **Lower inventory**: reduced stockouts and smaller buffer stocks, due to improved inventory data

• **Reduced stockouts**: substantially reduced stockouts can be achieved through RFID if movements to the shop floor can be tracked

• **Promotion execution**: RFID and the Internet of Things may be used to obtain better visibility for timely placements of promotional items

• **After sales services**: in after-sales service, RFID may be used for warranty issues, repair and goods authentication

Benefits for consumers are:

• **Personal access to product specific information**: e.g., to be able to access the product history of a car, based on a vehicle identification number

• **Active participation opportunity**: e.g., through beta testing, product ratings, field reports, applications and more

• **Interaction with other stakeholders**: e.g., automatic updates and repairs, dynamic safety warnings, product recalls, public applications

• **Home automation and leisure applications**: e.g., room monitoring, smart devices, intelligent toys

Benefits to society include:

• **Consumer protection / safety**: e.g., food and health safety, environmental monitoring

• **Security:** e.g., to avoid terrorist attacks, customs support

• **Trade facilitation**: comparable with the introduction of Infrastructure optimization: e.g., roads, public transportation

Cost Benefit Sharing

- Costs and benefits of the Internet of Things that are not evenly distributed between the stakeholders.
- Cost benefit sharing models may be used as a tool to balance these asymetries.
- Cost benefit sharing in combination with RFID has been researched by several authors.
- Sharing benefits and investments in multi-tiered situations is seen as a core requirement for wide-scale deployment of RFID.
- Hirthammer and Riha (2005) define cost benefit sharing as:

"A systematic and system-oriented incentive system that motivates companies in a network to participate in joint projects that do not benefit them directly. A Joint Project is a cooperative effort to improve the processes or resource allocation in the network. It involves at least two parties in the network."

According to Hirthammer and Riha (2005), the cost benefit sharing process loop can be structured in several subtasks:

1. Detailed process analysis in the network through auditing
2. Enquiry of weak points through benchmarking
3. Development of corresponding actions to solve or lessen the effect of the weak points based on overall strategies and goals
4. Cost benefit sharing
   a. Calculation of costs
   b. Evaluation of benefits
      i. Calculate monetary benefits
      ii. Calculate qualitative benefits
      iii. Evaluate total benefit

iv. Calculate share of benefit
c. Distribution of costs
5. Implementation of actions proposed in step 3
6. Controlling
7. Feedback loop to adjust the system to external dynamics

A Technical Framework for Integrating Billing Capabilities into the EPCglobal Network

The well-known Fosstrak EPCIS software has been integrated with jBilling, an open source billing solution that is mainly being used in telecommunication companies. The jBilling system has been chosen for the following three reasons.

- Firstly, it does not require an upfront investment in software.
- Secondly, it is open source and, therefore, allows modification to the software.

- And thirdly, it aligns well with the technologies used in Fosstrak and therefore may allow a tighter integration

Both products use Tomcat as Web-server, but there are two different relational databases in use, Hypersonic for jBilling and MySQL for Fosstrak. jBilling can run on MySQL, so that Hypersonic could be eliminated in a further integration effort. To combine the two different systems, there are two initial requirements:

1.There should be an integrated login procedure .

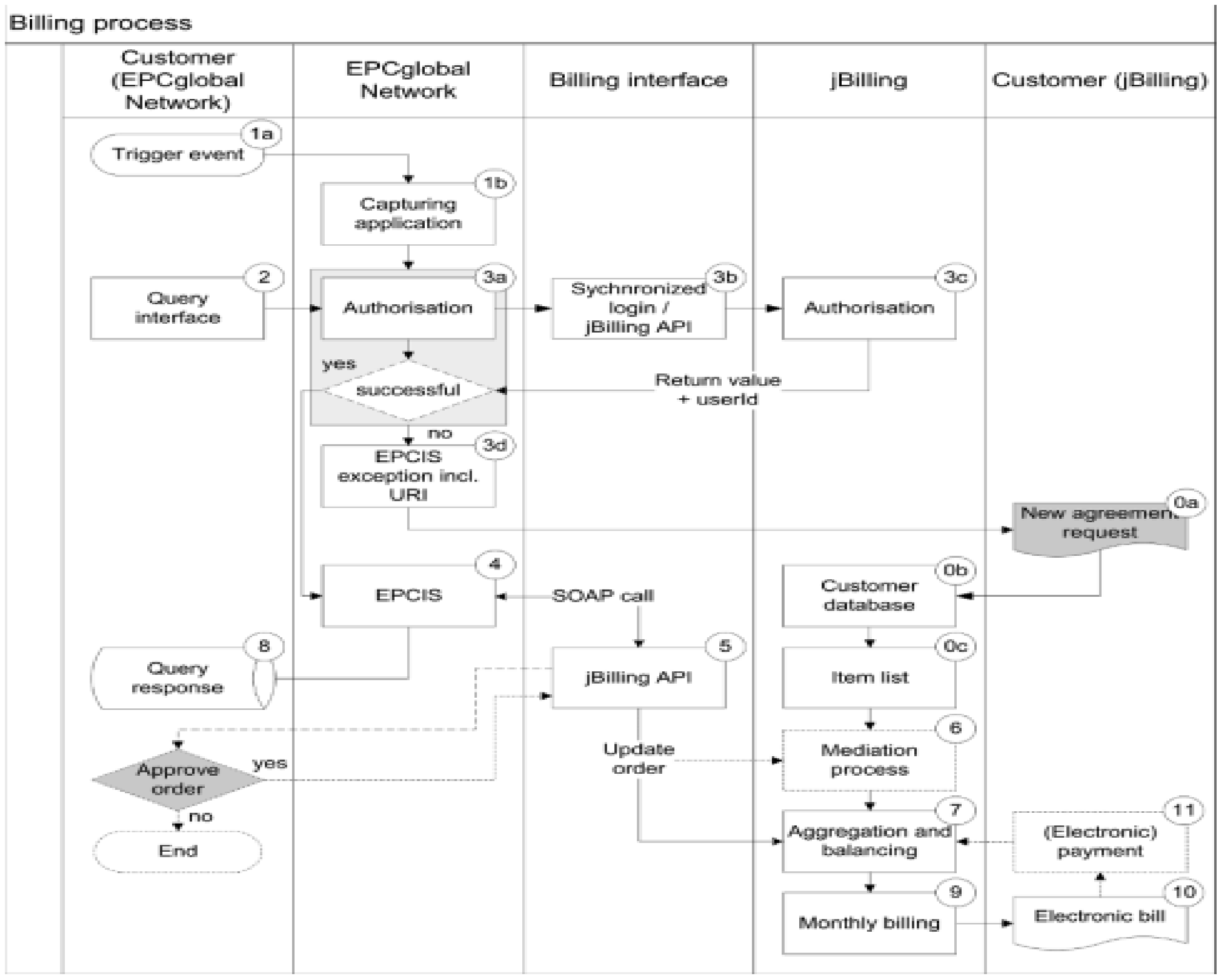
2. Selected EPCIS events should be translated to jBilling purchase orders

The accounting process may be triggered by an event, such as a pallet with an RFID tag passing a dock-door (1a, 1b). Other billing activities may be started through a query for payable information (2). As part of the Fosstrak authentication process (3a), the access rights, including the availability of a billing account (3c), are checked via the jBilling Application Programming Interface (API).

For this purpose, a combined login process has been implemented as an option in the Fosstrak EPCIS query interface (NextFigure) at the LogDynamics Lab.

- Other billing activities may be started through a query for payable information (2).
- As part of the Fosstrak authentication process (3a), the access rights, including the availability of a billing account (3c), are checked via the jBilling Application Programming Interface (API).
- For this purpose, a combined login process has been implemented as an option in the Fosstrak EPCIS query Interface.
- Currently, only basic authentication is enabled.
- If the input data is null or missing, jBilling generates an API exception (jBilling 2010).
- Otherwise jBilling returns different integer values as described below.
- 0 -The user was successfully authenticated and his current status allows him entrance to the system.
- 1 -Invalid credentials. The user name or password are not correct.
- 2 -Locked account: The user name or password are incorrect, and this is the last allowed login attempt. From now on, the account is locked.
- 3 -Password expired: The credentials are valid, but the password is expired. The user needs to change it before logging in
- If no valid contract in jBilling can be found, JbillingAPIException could be converted into an EPCIS exception, containing a Uniform Resource Identifier (URI) that links to a new agreement request (3d).
- The agreement may contain pricing information, financial details, such as preferred payment service, and payment options (e.g. monthly).
- For further usage in the Internet of Things it would be favorable, if individual service level agreements and information quality details could be included or linked as well.

- The agreement is stored within the jBilling customer database (0b) and will be used for calculating customer-specific prices later on.
- In a further effort it would be possible to create, update and delete new jBilling users from within the EPCIS, using the jBilling API.
- Consequently, users would not need to deal with two different systems.
- After successful authorisation, the EPCIS queries are processed.
- The EPCIS will make a SOAP call to the jBilling API (5).
- The userID provided during the authorisation process is used to link an order to a jBilling account.
- The createOrder and updateOrder methods are used to transfer events into corresponding orders.
- Optionally, a mediation process (6)can be called to enable dynamic pricing, based on business rules.
- If prices per item are predefined in jBilling and if no changes are required, the mediation process does not need to be called (jBilling 2010).
- The jBilling API updates the account balance (7). Optionally, an approval request for the end-user can be implemented.
- Finally the query response is delivered and the account balance is updated by jBilling.
- Usually, monthly billing will be used to invoice the aggregated values in business scenarios (9).
- In order to avoid problems resulting from analogue to digital media conversions and cost intensive manual labour, electronic bills (10) and electronic payment (11) will be preferred.
- Additionally, an invoice is sent via e-mail or traditional postal services to a defined recipient.

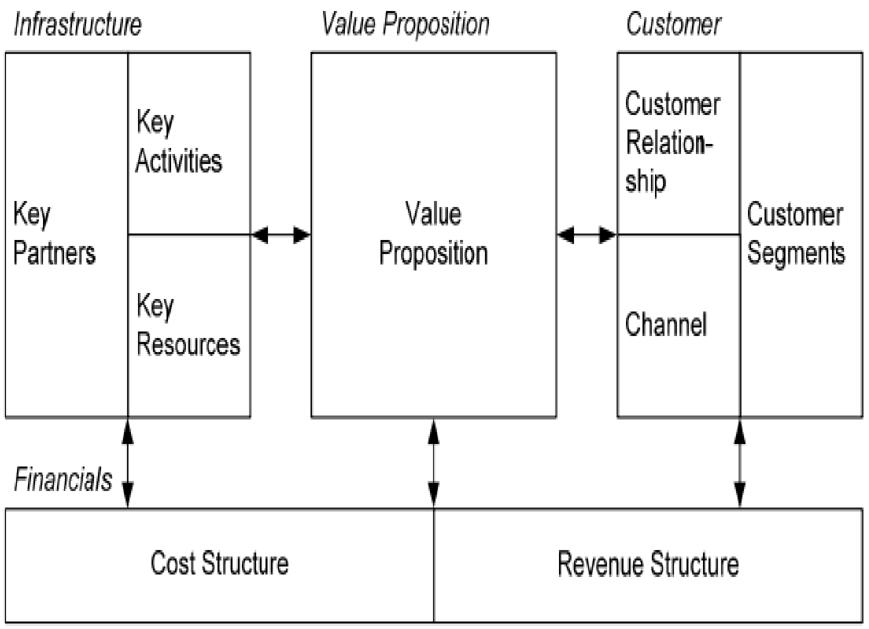


Business Models for the Internet of Things

The Internet has significantly changed the way products and services are marketed and distributed and thus let to a series of new types of business models. Current applications in the Internet of Things generally focus on the optimization of existing processes and associated cost reductions within companies and along value chains. Product Life Cycle Management, Customer Relationship Management, and Supply Chain Management are typical application scenarios.

New application scenarios, sometimes referred to as smart technologies and smart services, are more focused on revenue generation The business model has to be seen as the replacement or complement of the traditional unit of analysis, as a result of the altered surrounding conditions. Today's business condition is determined by technological progress, service orientation, the digitalization of products as well as increasing relevance of cooperation and ecosystems of different companies, which blur the boundaries of the individual enterprise. Accordingly, each business is implicitly based on a business model, even though it is not always explicitly presented. Timmers (1998) defines a business model as "an architecture of the products, services and information flows ". This includes the involved actors and roles as well as the potential value created for all participants and the source of revenue.

In relation to the Internet of Things we see the business model as a major element to unite its technical developments with its economical business perspective. According to Afuah and Tucci (2000), "a business model can be conceptualized as a system that is made up of components, linkages between the components, and dynamics". Components refer to the elements to be addressed by a business model. Just like the definitions of the term "business model" the proposed components vary largely between different authors. Osterwalder and Pigneur (2009), defined a framework which is referred to as the "**business model canvas**". The applicability of the model is proven by its use in practice



The business model framework depicted in Figure includes four main perspectives of the business model, namely

- The value proposition,
- The customer,

- Financials
- The infrastructure.

The components are not stand-alone but mutually influence each other. The value proposition specifies what is actually delivered to the customer. This goes beyond the product or service offered. It describes which customer needs are satisfied and details what other quantitative (e.g., price or speed of service) and qualitative aspects (e.g., brand, design, cost/risk reduction) contribute to the offered value. In the Internet of Things we consider raw data about physical objects as well as any aggregated or processed information a core component of the value proposition. The customer perspective includes the customer segments addressed by the company, such as related channels and customer relationships.

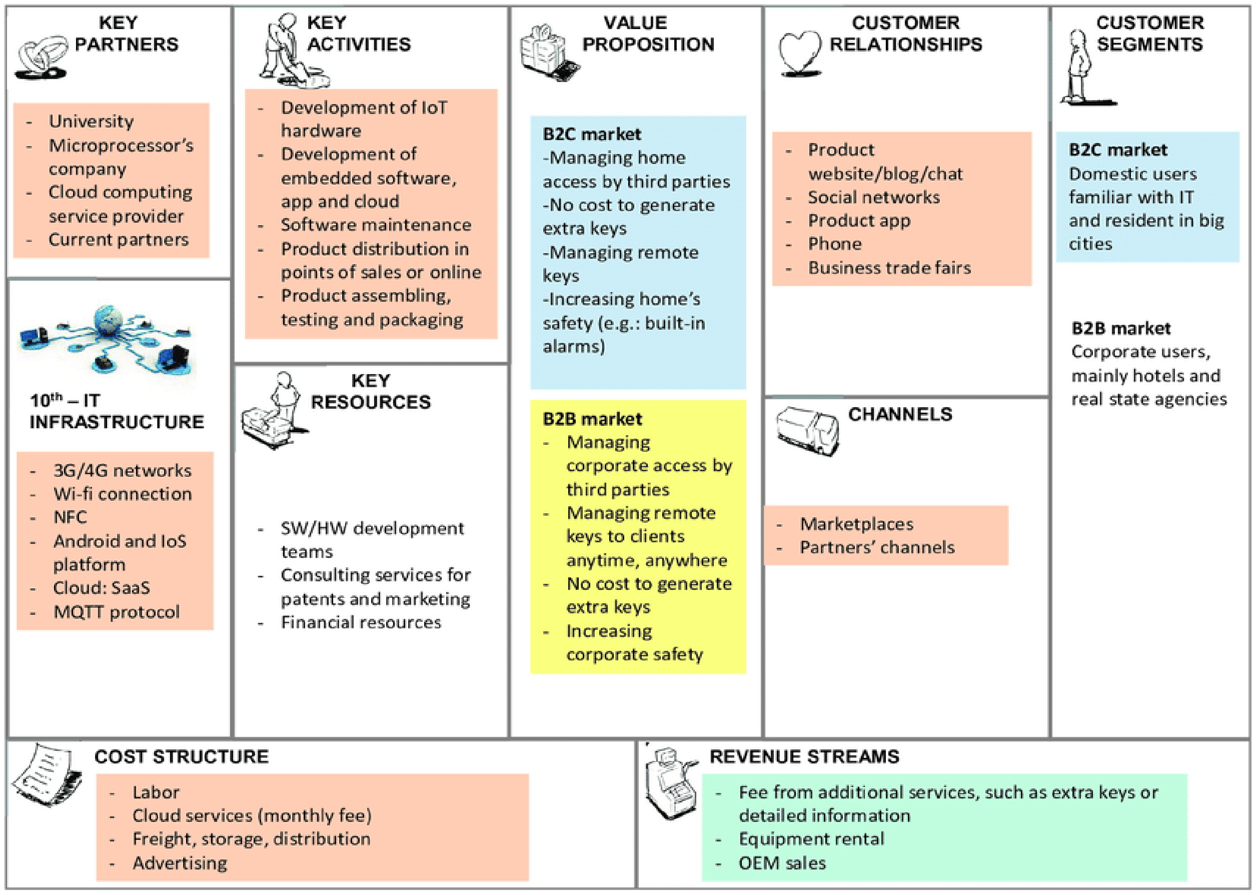
- The **customer segments** define the different groups of people that are served. Different types of customer segments can be distinguished: mass market vs. niche market, segmented vs. diversified or multisided platforms. Multisided platforms will exist, if two or more interdependent customer segments are served by the company (e.g. credit card companies). The company can reach its customers, respectively customer segments through different channels. These can be direct or indirect and owned by the company itself or by partners.

- **Channels** can be aligned to the different phases of the lifecycle, such as creating awareness for the value proposition, evaluation of the value proposition through the customer, purchase, delivery and after sales.

- **Customer relationships** are often determined by the channels used. Relationships can range from very loose (self-service, automated services) to highly engaged (personal assistance, communities, co-creation). The financial perspective comprises the costs as well as the revenues.
- The **revenue structure** depicts the sources and ways of revenue generation. Here, different types of revenue streams can be distinguished: asset sale, usage fee, subscription fee, lending / renting / leasing, licensing, brokerage fee, and advertising.
- The **cost structure** describes the most important costs (variable and fixed) inherent to the business model. The business model can be rather value or cost driven. Companies can use economies of scale or economies of scope to create a successful business model. Key partners, key activities and key resources can be referred to as the infrastructure components.
- The **key resources** are the assets required to make the business model work. Key

resources can be physical, intellectual, human or financial.

- The **key activities** describe the most important actions to be performed by the company in order to create, offer and market the value proposition. These can be producing, problem solving or developing and maintaining a platform, respectively.
- **Key partners** are the network of suppliers and collaboration partners (strategic alliances, outsourcing partners, co-creation) the business model depends on.



KEY PARTNERS
- University
- Microprocessor's company
- Cloud computing service provider
- Current partners

10th – IT INFRASTRUCTURE
- 3G/4G networks
- Wi-fi connection
- NFC
- Android and IoS platform
- Cloud: SaaS
- MQTT protocol

KEY ACTIVITIES
- Development of IoT hardware
- Development of embedded software, app and cloud
- Software maintenance
- Product distribution in points of sales or online
- Product assembling, testing and packaging

KEY RESOURCES
- SW/HW development teams
- Consulting services for patents and marketing
- Financial resources

VALUE PROPOSITION

B2C market
-Managing home access by third parties
-No cost to generate extra keys
-Managing remote keys
-Increasing home's safety (e.g.: built-in alarms)

B2B market
- Managing corporate access by third parties
- Managing remote keys to clients anytime, anywhere
- No cost to generate extra keys
- Increasing corporate safety

CUSTOMER RELATIONSHIPS
- Product website/blog/chat
- Social networks
- Product app
- Phone
- Business trade fairs

CHANNELS
- Marketplaces
- Partners' channels

CUSTOMER SEGMENTS

B2C market
Domestic users familiar with IT and resident in big cities

B2B market
Corporate users, mainly hotels and real state agencies

COST STRUCTURE
- Labor
- Cloud services (monthly fee)
- Freight, storage, distribution
- Advertising

REVENUE STREAMS
- Fee from additional services, such as extra keys or detailed information
- Equipment rental
- OEM sales

Business Model Innovation

Business model innovation is the art of enhancing advantage and value creation by making simultaneous—and mutually supportive—changes both to an organization's value proposition to customers and to its underlying operating model. External factors, such as technological innovations, increased competition, and market changes as well as legal or regulatory changes are seen as the dominant triggers of business model innovation. Business model innovation can help to align innovation activities within the company. Business model innovation as a process resulting in a qualitatively new business model, which differs distinctively from the previous. A deliberate change of one or more key elements of the business model, respectively their interrelations, has to take place. Some of the most successful companies that have used a distinctively new business approach based on the Internet are shown in Table

Company	**Traditional business**	**Initial business model innovation**	**Further developments**
Amazon	Book trade	Online shopping Automated distribution model Collaborative filtering	Shopping portal Digitalisation (mp3, books), Terminals (Kindle), Mobile payments Amazon web services (incl. billing)
eBay	Classifieds Flea markets Auctions	Online auctions	Shopping portal Payment services (PayPal)
Google	Yellow pages	Hypertext web search Prioritized advertisements	Terminals (Android), Video (You Tube), Maps (Google Maps), Web based software (e.g. Google Docs), Digitalized books Payment services (Checkout)
Apple iTunes	Music shops	Music digitalization Terminals (iPod, iPhone, iPad) Applications (apps)	Videos, Newspapers

Value Creation in the Internet of Things

A typical business transaction today is defined by a physical product, information stream, and money stream. The Internet of Things may be seen as an approach to align these different streams. It provides a higher level of visibility and control mechanisms. Moreover, in the Internet of Things, information itself may become a major source for value creation and thus the value proposition. This includes information only made possible through Internet of Things technologies as well as the association of existing information to physical products. Traditionally, the money stream is exclusively dependent on the product stream prices.

A separate price for the information is not defined. Instead, information is most often expected to be free of charge. It is obvious that the costs of information are hidden in the product price. However, the reluctance to pay for information may change over time. In B2C-markets, the willingness to pay for digital goods has increased to 88%. Even though digital goods (e.g. software, tickets, travel, songs, and videos) and information are not synonymous, it is still obvious that there is a change in society to accept the Internet as a business transaction platform. In addition to direct information payments, alternative revenue streams should be considered.

Laws of Information

Even though information is recognized as an asset on its own right, quantitative measurements are difficult to achieve. It consumes a growing number of organizational resources for data capturing, storage, processing and maintenance. While hardware and sometimes software may be capitalized, the value of information in general is not financially recognized in the balance sheets. Information may be considered a product that is produced out of raw data through hard- and software utilization. Moody and Walsh (2002) define seven "laws of information", explaining the specifics of information compared to other (physical) assets. These "laws of information" provide opportunities for new business and pricing models for the Internet of Things:

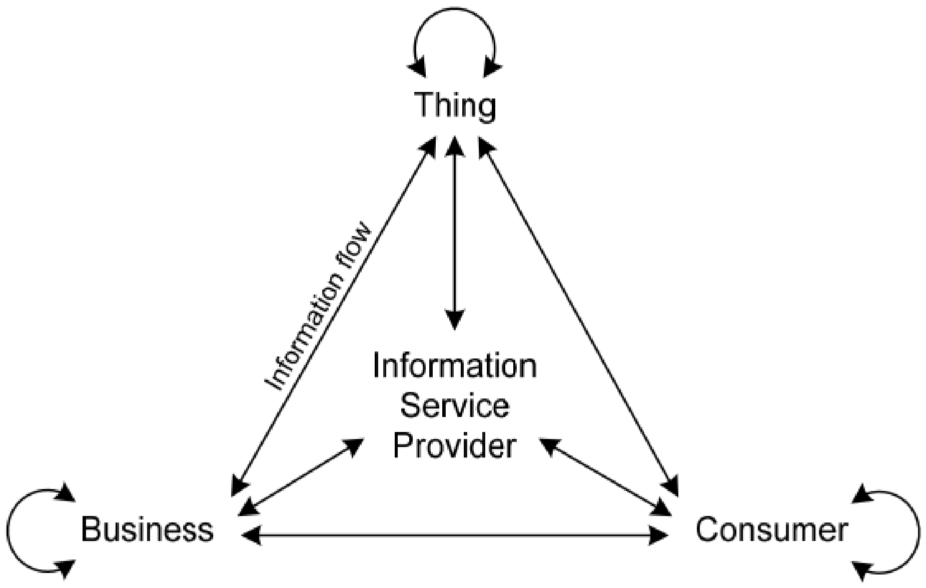
- **First Law of Information**: Information is (Infinitely) Shareable and Can Be Shared with Others Without a Loss of Value
- **Second Law of Information**: The Value of Information Increases with Use and It Does Not Provide Any Value, If It Is Not Used at All
- **Third Law of Information**: Information Is Perishable and It Depreciates Over Time
- **Fourth Law of Information**: The Value of Information Increases with Accuracy.
- **Fifth Law of Information**: The Value of Information Increases When Combined with Other Information
- **Sixth Law of Information**: More Information Is Not Necessarily Better .
- **Seventh Law of Information**: Information Is Not Depletable

Revenue Generation in the Internet of Things

Information may become the main source of value creation and thus a major part of the value proposition in the Internet of Things. Information can be directly associated to things or products and instances of products. The usage, status, and location of things become traceable. This allows for new value proposition scenarios, such as the provision of additional product-related data to or the exact billing of products or services based on the actual use (e.g., rental car, returnable transport items). The following requirements constitute the specifics for the value proposition:

- **Providing the right information** ... – Linked through a unique identifier to a physical product
- **... in the right granularity** ... – High information granularity, providing a new dimension of clarity and insight
- **... and the right condition** ... – High information accuracy – Aggregation of information from various sources, such as tags, sensors or embedded systems – Correlation, integration, and further analysis of information in a way that allows new insights to be derived – Defined syntax and semantics
- **... at the right time ...** – Timeliness of information – Access to real time information as well as to historical data for business analysis – Real-time analytics and business intelligence for high resolution management – Intelligent real-time decision-making capability based on real-time physical events.
- **... anywhere in the network ...**– Online access and possibly offline usage – Mobile access
- ... **at an appropriate price.** – Price transparency – Low premium for billing service, the price should be paid for the information rather than the infrastructure

In order to fully understand the information exchange on the Internet of Things the information flows and actors involved have to be considered. Figure depicts information providers in the Internet of Things and information flows between them.



They can be depicted as a triangle of information exchange. Information flows can be direct, such as for example thing-to-thing, business-to-consumer or consumer-to–thing, or indirect, such as from thing-to business through an information provider or from business-to-business through a thing. Things include products that communicate their ID and status through sensors as well as data processing units and actuators.

Exemplary Business Model Scenarios for the Internet of Things

The field of application for Internet of Things technology is much wider then we have seen so far. The control of processes and the quality of goods in manufacturing, logistics, service and maintenance are still valid applications. Moreover, new areas of applications have to be considered. The following exemplary scenarios will include the use of Internet of Things technology to support the offer of PaaS, the role of information service providers in the Internet of Things, the integration of end-users and opportunities through right-time business analysis and decision making.

Scenario 1: Product as a Service (PaaS)

The shift from providing products to providing services is a major trend in business model innovation. Not only software companies provide SaaS instead of selling software licenses, but more and more manufacturers follow this trend. Hilti, an international manufacturer and supplier of professional construction tools, launched what they call “Fleet Management”. The customer is no longer required to own a tool. Instead, Hilti offers its customers access to a range of tools on a contract basis and monthly fee, including additional services, such as repairs.

Today, the shift to PaaS is often hindered by missing means of performance measuring and billing as well as unsuitable pricing models. The Internet of Things offers a range of possibilities to support such PaaS scenarios. Sensors allow for the tracking of a product and the location of its current position. In addition, the usage times of a product can be exactly documented as well as the condition under which a product was used (e.g. the speed at which a car has been driven). Sensors also enable a company to monitor the condition of the product or parts and tools and thus support maintenance and repairs.

Scenario 2: Information Service Providers

If information can be measured and billed, new business opportunities for information service providers will be enabled. Data centres can provide storage and processing capabilities for Internet of Things-related data. Information service providers can aggregate and process information from different sources, thus providing a higher value of information. However, the value proposition which was only made possible through unique identification and billing in the Internet of Things differs significantly. Most important cost factors are the acquisition and aggregation of information (data) and the purchase and maintenance of needed information systems. A potential application scenario for information service providers is anti-counterfeiting. The problem of anti-counterfeiting is prevalent in the consumer goods market. Brand items, such as apparel and accessories or even worse drugs or spare parts are copied and sold as original products. This results in economic damage and can have severe impacts on the consumer side. The Internet of Things supports this scenario through the association of information to a product instance. The information service is aimed at the verification of the originality of a certain product in order to detect counterfeits. The information service provider has specialized on the verification of spare parts in the machinery and equipment industry as well as the automotive industry. The consumer – the buyer of the spare part or a service partner installing it – can submit a request to the information provider through the Internet of Things. The information service provider could query its information systems where information from different sources is aggregated and find out whether the serial number is valid

Scenario 3: End-user Involvement

The Internet of Things provides a new level of consumer integration into co-creation processes. While “living labs” have been used to integrate limited user groups into product and service development at a certain stage in the product life cycle, the Internet of Things will link all consumers across the life cycle of a product. Companies that will know how to utilize this huge potential will be in the lead for new business models in B2C scenarios. Other services include a payment scheme for product reviews, that is based on positive review ratings. Ciao11 is offering their users a small financial benefit as low as 0.5 pence every time their product rating is positively reviewed. In any case a high level of security and privacy as well as the freedom of choice to participate are mandatory. Through the use of a mobile phone, the end-user is enabled to supply and retrieve product related information at the point of sale – in this scenario a large supermarket chain. Both actions are supported by the use of RFID-chips or barcodes.

Scenario 4: Right-time Business Analysis and Decision making

In production engineering, real-time usually refers to M2M-systems that record events and responds within milliseconds. In logistics, the time frame is not as well-defined, yet seconds, minutes, or even hours are sometimes still considered real-time, compared to longer traditional processes, such as transportation that causes information gaps of days or weeks. Real-time is often used as a qualitative rather than a quantitative value to differentiate timely from out-of-date information distribution thus allowing acting instead of reacting. Therefore, it is more appropriate to use right-time business analysis and decision making. The amount of time between a business event and a decision is influenced by time periods, including data capturing latency, analysis latency, and decision latency. For perishable goods manual spot tests and visual inspections are common, but these cannot provide real-time monitoring or proactive strategies.

The Internet of Things provides real-time access and analysis opportunities across supply-chains or product lifecycles. Data analysis can be provided in proximity to things (smart objects), at the business premises or anywhere in the Internet of Things. Agile management strategies are enabled based on real-time availability and analysis of data. The envisioned scenario is based on an intelligent truck that combines different technologies and applications to increase the value of information and to a boost utilization of the Internet of Things infrastructure. Some easy tasks, such as navigation and dynamic routing, can be achieved without the Internet of Things, more complex tasks, such as tracking and condition monitoring, would largely benefit from it.

UNIT-V
From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices

5.1From the Internet of Things to the Web of Things

As more and more devices are getting connected to the Internet, the next logical step is to use the World Wide Web and its associated technologies as a platform for smart things.

Cool Town project, Kindberg et al. (2002) proposed to link physical objects with Web pages containing information and associated services. Using infrared interfaces or bar codes on objects, users could retrieve the URI of the associated page simply by interacting with the object.

Another way to use the Web for real-world objects is to incorporate smart things into a standardized Web service architecture (using standards, such as SOAP, WSDL(Web Services Description language), UDDI (Universal Description, Discovery, and Integration) a way to publish and discover information.

Instead of these heavyweight Web services often referred to as WS-* technologies, recent "Web of Things" projects have explored simple embedded Hypertext Transfer Protocol (HTTP) servers and Web 2.0 technologies.

So far, projects and initiatives, subsumed here under the umbrella term "Internet of Things", have focused mainly on establishing connectivity in a variety of challenging and constrained networking environments. A promising next step is to build scalable interaction models on top of this basic network connectivity and thus focus on the application layer.

The services that smart things expose on the Web usually take the form of a structured XML document or a JavaScript Object Notation (JSON) object, which are directly machine-readable. These formats can be understood not only by machines, but are also reasonably accessible to people. With this smart things can not only communicate on the Web, but also provide a user-friendly representation of themselves. This makes it possible to interact with them via Web browsers and thus explore the world of smart things with its many relationships

Web Of Things definition:

Web of Things is all about making devices accessible over the web using web protocols like HTTP, Web Socket, JSON etc. agnostic to any thing below the application layer, so that any device can be part of the universal WOT regardless of what protocols it uses to connect to internet.

IoT	WoT
Devices can be connected with any form of internet	WoT is made to handle and use the potential of IoT
It deals with actuators, sensors, computation, communication Interfaces.	It deals with web servers and Protocols.
Digitally Augmented objects make IoT	WoT is made up of the applications that are made for Io Devices.
Every IoT devices have a different Protocol	A single protocol is used for multiple/various IoT devices.
Programing is difficult because of multiple protocols	Programming is easy so it doesn't have multiple protocols.
All the protocols and standard are private and it cannot be accessed publicly	WoT can be accessed freely by anyone, anytime.

Dynamically generated real-world data on smart objects can be displayed on such "representative" Web pages, and then processed with Web 2.0 tools. For example, things can be indexed like Web pages via their representations, users can "google" for them, and their URI can be emailed to friends or it can be bookmarked. The physical objects themselves can become active and publish blogs or inform each other using services, such as Twitter.

5.2 Designing RESTful Smart Things

The "Web of Thing was unheard of before. Now, the use of REST as a universal interaction architecture, that interacts with smart things can be built around universally supported methods and a set of guidelines to Web-enable smart things and illustrate them with concrete examples of implemented prototypes. It was the architecture of the Web that allowed data and services to be shared in a way that

5.2.1 Modeling Functionality as Linked Resources

The central idea of REST revolves around the notion of a resource as any component of an application that is worth being uniquely identified and linked to. On the Web, the identification of resources relies on Uniform Resource Identifiers (URIs), and representations retrieved through resource interactions contain links to other resources, so that applications can follow links through an interconnected web of resources. Clients of RESTful services are supposed to follow these links, just like one browses Web pages, in order to find resources to interact with. In the case of the Sun SPOT, each node has a few sensors (light, temperature, accelerometer, etc.), actuators (digital outputs, LEDs, etc.), and a number of internal components (radio, battery). Each of these components is modeled as a resource and assigned a URI. For instance, typing a URI such as

http://.../sunspots/spot1/sensors/light

in a browser requests a representation of resource light of the resource sensors of spot1

Resources are primarily structured hierarchically and each resource also provides links back to its parent and forward to its children.
As an example, the resource

http://.../sunspots/spot1/sensors/

provides a list of links to all the sensors offered by spot1.

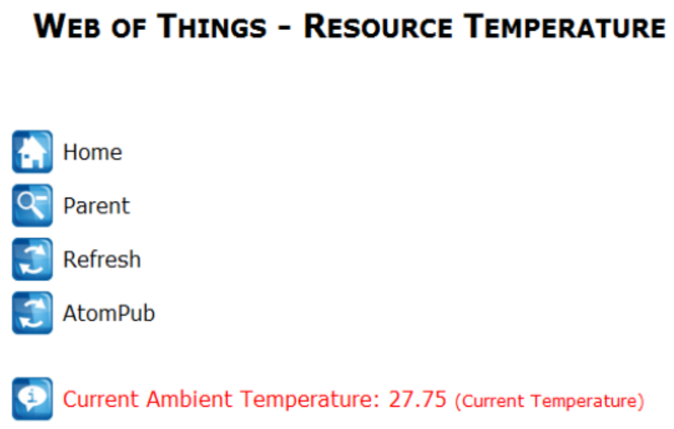
This interlinking of resources that is established through both, resource links and hierarchical URI, is not strictly necessary, but well-designed URIs make it easier for developers to "understand" resource relationship and even allow non-link based "ad-hoc interactions". In a nutshell, the first step when Web-enabling a smart thing is to design its resource network, Identification of resources and their relationships are the two important aspects

5.2.2 Representing Resources

Resources are abstract entities and are not bound to any particular representation. Thus, several formats can be used to represent a single resource. However, agreed-upon resource representation formats make it much easier for a decentralized system of clients and servers to interact without the need for individual negotiations. On the Web, media type support in HTTP and the Hypertext Markup Language (HTML) allow peers to cooperate without individual agreements. It further allows clients to navigate amongst the resources using hyperlinks.

For machine-to-machine communication, other media types, such as the XML and the JSON have gained widespread support across services and client plat- forms. JSON is a lightweight alternative to XML that is widely used in Web 2.0 applications. In the case of smart things, it has been suggested to support for at least an HTML representation to ensure browsability by humans. Note that since HTML is a rather verbose format, it might not be directly served by the things themselves, but by intermediate proxies. For machine-to-machine communications, it has been suggested using JSON. Since JSON is a more lightweight format compared to XML, because it is better adapted to devices with limited capabilities such as smart things. Furthermore, it can directly be parsed to JavaScript objects. This makes it an ideal candidate for integration into Web mashups(A Mashup is a technique that websites use to provide resources, functionalities, and services from multiple sources.). In the Sun SPOT example, each resource provides both, an HTML and a JSON representation. As an example, the listing in Figure shows the JSON representation of the temperature resource of a Sun SPOT and Figure shows the same resource represented as an HTML page with links to parents, sub resources, and related resources.

```
{"resource":

 {"methods":["GET"],

  "name":"Temperature",

  "children":[],

  "content":

  [{"description":"Current Temperature",  "name":"Current Ambient Temperature",
"value":"27.75"}]}}
```


WEB OF THINGS - RESOURCE TEMPERATURE

Home
Parent
Refresh
AtomPub

Current Ambient Temperature: 27.75 (Current Temperature)

5.2.3 Servicing Through a Uniform Interface

In REST, interacting with resources and retrieving their representations all happens through a uniform interface which specifies a service contract between the clients and servers. The uniform interface is based on the identification of resources, and in case of the Web, this interface is defined by the HTTP

1.Operations

HTTP provides four main methods to interact with resources, often also referred to as "verbs": GET, PUT, POST, and DELETE. GET is used to retrieve the representation of a resource. PUT is used to update the state of an existing resource or to create a resource by providing its identifier. POST creates a new resource without specifying any identifier. DELETE is used to remove (or "unbind") a resource. In the Web of Things, these operations map rather naturally, since smart things usually offer quite simple and atomic operations. As an example, a GET on

http://.../spot1/sensors/temperature

returns the temperature observed by spot1, i.e., it retrieves the current representation of the temperature resource. A PUT on

http://.../sunspots/spot1/actuators/leds/1

with the updated JSON representation {"status":"on"} (which was first retrieved with a GET on /leds/1) switches on the first LED of the Sun SPOT, i.e., it updates the state of the LED resource.

A POST on

http://.../spot1/sensors/temperature/rules

with a JSON representation of the rule as {"threshold":35} encapsulated in the HTTP body, creates a rule that will notify the caller whenever the temperature is higher than 35 degrees, i.e., it creates a new rule resource without explicitly providing an identifier.

Finally, a DELETE on

http://.../spot

is used to shutdown the node, or a DELETE on

http://.../spot1/sensors/temperature/rules/1

is used to remove rule number 1.

Additionally, another less-known verb is specified in HTTP and implemented by most Web servers: OPTIONS can be used to retrieve the operations that are allowed on a resource. In a programmable Web of Things, this feature is quite useful, since it allows applications to find out at runtime what operations are allowed for any URI.

As an example, an OPTIONS request on
http://.../sunspots/spot1/sensors/tilt
returns GET, OPTIONS.

2. Content Negotiation

HTTP also specifies a mechanism for clients and servers to communicate about the requested and provided representations for any given resource; this mechanism is called content negotiation. Since content negotiation is built into the uniform interface of HTTP, clients and servers have agreed-upon ways in which they can exchange information about requested and available resource representations. The negotiation allows clients and servers to choose the best representation for a given scenario. A typical content-negotiation for the Sun SPOTs looks as follows. The client begins with a GET request on
http://.../spot1/sensors/temperature/rules

It also sets the Accept header of the HTTP request to a weighted list of media types it understands,

for example to: application/json;q=1,
application/xml;q=0.5.

The server then tries to serve the best possible format it knows about and specifies it in the Content-Type of the HTTP response. In the Sun SPOT, it cannot offer XML and would thus return a JSON representation and set the HTTP header Content-Type: application/json.

3. Status Codes

Finally, the status of a response is represented by standardized status codes sent back as part of the header in the HTTP message. There exist several dozens of codes which each have well-known meaning for HTTP clients. In a Web of Things, this is very valuable since it gives us a lightweight but yet powerful way of notifying abnormal requests execution.
As an example, a POST request on
http://.../sunspots/spot1/sensors/acceleration
returns a 405 status code that the client has to interpret as the notification that “the method specified in the request is not allowed for the resource identified by the request URI.”

5.2.4 Syndicating Things

Many applications for smart things require to syndicate information about objects or collections of objects. With Atom, the Web has a standardized and RESTful model for interacting with collections, and the Atom Publishing Protocol (AtomPub) extends Atom's read-only interactions with methods for write access to collections. Because Atom is RESTful, interactions with Atom feeds can be based on simple GET operations which can then be cached. Atom enables decoupled scenarios by allowing clients to monitor smart things by subscribing to feeds and polling a feed on a remote server, instead of directly polling data from each device. This model implemented for the Sun SPOTs, since it fits the interaction model of sensor networks. Thus, the nodes can be controlled (e.g., turning LEDs on, enabling the digital outputs, etc.) using synchronous HTTP calls, but can also be monitored by subscribing to feeds (node push).

For example, a subscription to a feed can be done by creating a new "rule" on a sensor resource and POSTing a threshold (e.g., > 100).

http://.../sunspots/spot1/sensors/light/rules

In response, the Sun SPOT returns a URI to an Atom feed. Every time the threshold is reached, the node pushes a JSON message to the Atom server using AtomPub. This allows for thousands of clients to monitor a single sensor by outsourcing the processing onto an intermediate, more powerful server.

5.2.5 Things Calling Back: Web Hooks

While Atom allows asynchronous communication between clients and smart things, clients still need to pull the feed server on a regular basis to get data. In addition to being inefficient in terms of communications, this might be problematic for scenarios where the focus is on monitoring. This is often the case with applications communicating with wireless sensor networks. For those applications, that is to suggest supporting HTTP callbacks, sometimes called Web hooks. Web hooks are a mechanism for clients and applications that want to receive notifications from other Web sites using user-defined callbacks over HTTP. Users can specify a callback URI where the application will POST data to once an event occurs. This mechanism has been used by the PayPal service which allows you to specify a URI to be triggered by the service once payment has been accepted. As an example, let us consider again the case of creating a new rule on a Sun SPOT:

http://.../sunspots/spot1/sensors/light/rules

Now, alongside with the rule, the client POSTs a URI on which it will listen for incoming messages. Every time the threshold is reached, the node (or an intermediate) will push a JSON message to the given URI(s). Using Web hooks is a first step towards bi-directional, real-time interaction with smart things. However, this model has a number of limitations as it requires from clients to have a public URI where data can be posted to, which is rarely the case when clients are behind a firewall.

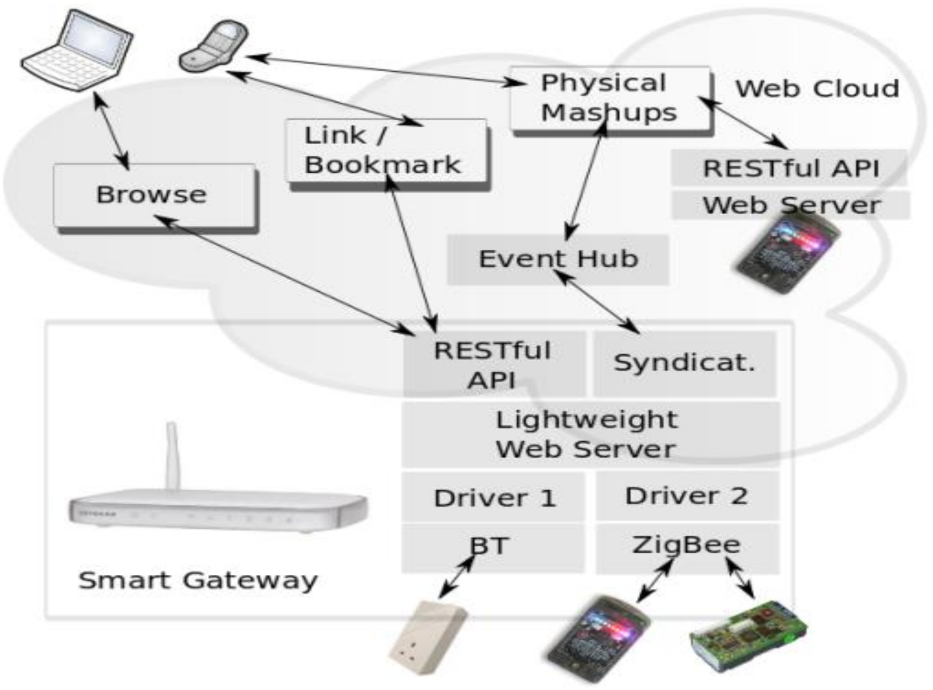
5.3 Web-enabling Constrained Devices

Although Web servers are likely to be embedded into more and more devices, we cannot assume that every smart device will directly offer a RESTful interface. In some cases, it makes sense to hide the platform-dependent protocol to access the resources of a particular device, and to expose them as RESTful service provided by a gateway. The actual interactions behind that RESTful service are invisible and often will include specialized protocols for the specific implementation scenario. REST defines the notion of intermediaries as a core part of the architectural style, and therefore such a design can easily be achieved by implementing the RESTful service on intermediaries. By using either proxies or reverse proxies, it is furthermore possible to establish such an intermediary from the client or from the server side, effectively introducing a robust pattern for wrapping non-RESTful services in RESTful abstractions.

In practice, two solutions are possible:

- Web connectivity directly on the smart things or
- indirectly through a proxy.

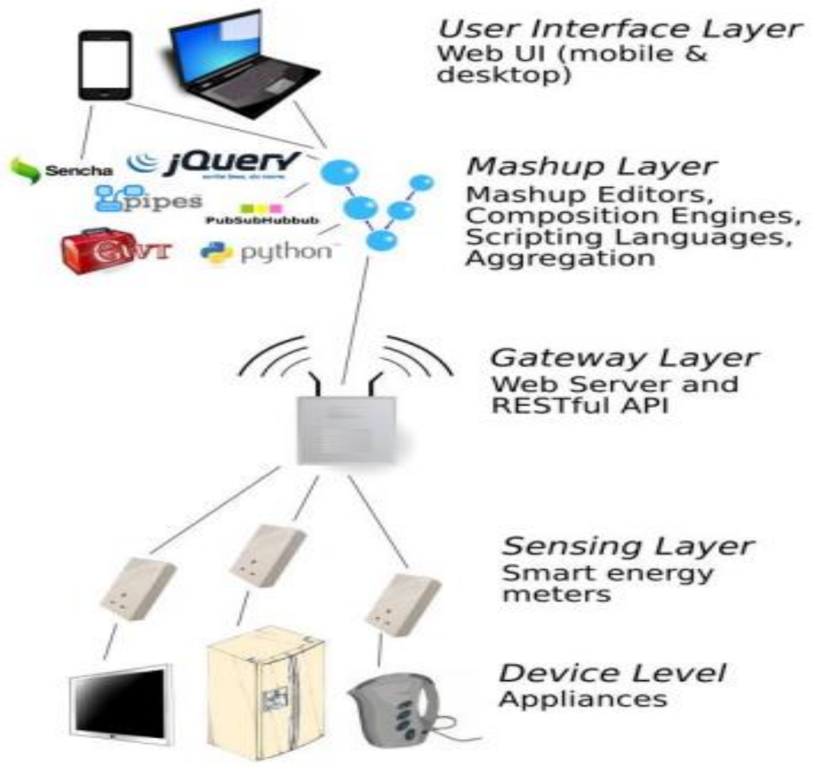
As there is no need to translate HTTP requests from Web clients into the appropriate protocol for the different devices, and thus devices can be directly integrated and make their RESTful APIs directly accessible on the Web, as shown in the right part of Figure. However, when an on-board HTTP server is not possible or not desirable, Web integration takes place using a reverse proxy that bridges devices that are not directly accessible as Web resources. We call such proxy as a Smart Gateway to account for the fact that it is a network component that does more than only data forwarding. A Smart Gateway is a Web server that hides the actual communication between networked devices (e.g., Bluetooth or Zigbee) and the clients through the use of dedicated drivers behind a RESTful service.



As an example, consider a request to a sensor node coming from the Web through the RESTful service. The gateway maps this request to a request into the proprietary API of the node and transmits it using the communication protocol understood by the sensor node. A Smart Gateway can support several types of devices through a driver architecture, as shown in the above Figure, where the gateway supports three types of devices and their corresponding communication protocols. Ideally, gateways should have a small memory footprint to be integrated into embedded computers already present in network infrastructures, such as wireless routers, set-top boxes, or Network Attached Storage (NAS) devices. A Smart Gateway can also provide more complex functions to devices such as orchestration and composition of several low-level services, offered by various devices into higher-level services available through the RESTful service.

Example: A Smart Gateway for Smart Meters

A prototype for a smart meter infrastructure illustrates the application of the WoT architecture and the concept of Smart Gateways for monitoring and controlling the energy consumption of households. We used intelligent power sockets, called Plogg , which can measure the electricity consumption of the appliance plugged into them. Each Plogg is also a wireless sensor node that communicates over Bluetooth or Zigbee. However, the integration interface offered by the Ploggs is proprietary, which makes the development of applications using Ploggs rather tedious, and does not allow for easy Web integration.



The Web-oriented architecture we have implemented using the Ploggs is based on five main layers as shown in the above Fig.

- The **Device Layer** is composed of appliances we want to monitor and control through the system.
- In the **Sensing Layer**, each of these appliances is then plugged into a Plogg sensor node.
- In the **Gateway Layer**, the Ploggs are discovered and managed by a Smart Gateway as described before.
- In the **Mashup layer** the Ploggs' services are composed together to create an energy monitoring and control application, using Web scripting languages or composition tools.

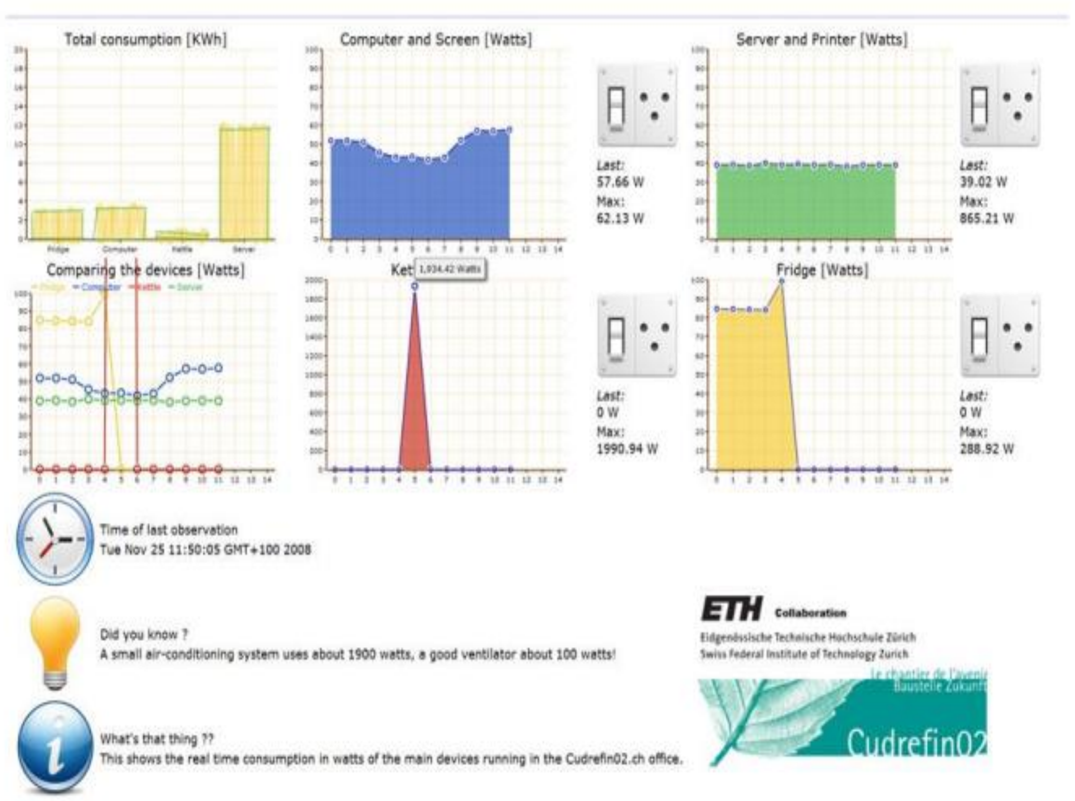
Finally, this application is made available through a **Web User Interface** in a Web browser (e.g., on a mobile phone, a desktop computer, a tablet PC, etc.) The Smart Gateway in this example is a C++ application running on an embedded machine, whose role is to automatically find all the Ploggs in the environment and make them available as Web resources. The gateway first periodically looks for the Ploggs in the area by scanning the environment for Bluetooth devices. The next step is to expose them as RESTful resources. A small footprint Web server (Mongoose9) is used to enable access to the Ploggs' functionalities over the Web, simply by mapping URIs to the various requests of the native Plogg Bluetooth API.

5.4 Physical Mashups: Recomposing the Physical World

A Web mashup is an application that takes several Web resources and uses them to create a new application. Unlike traditional forms of integration, mashups focus mainly on opportunistic integration occurring on the Web for an end-user's personal use and generally for non-critical applications. They are usually created ad-hoc, using lightweight and well-known Web technologies, such as JavaScript and HTML, and contribute to serving short terms needs. As an example, a mashup can be created to display, on Google Maps, the location of all the pictures posted to Flickr. By extending the mashup concept to physical objects and applying RESTful patterns to smart things, we allow their seamless integration into the Web, thus enabling a new range of applications based on this unified view of a Web of information resources and physical objects. We call this concept "physical mashup", because it is directly inspired from Web 2.0 mashups.

5.4.1 Energy Aware Mashup: "Energie Visible"

The first example is to create a mashup to help households to understand their energy consumption and to be able to remotely monitor and control it. The idea of the "Energie Visible" project is to offer a Web dashboard that enables people to visualize and control the energy consumption of their household appliances. The dashboard is shown in the following Figure and provides six real-time interactive graphs. The four graphs on the right side provide detailed information about the current electricity consumption of all the detected Ploggs.



Time of last observation
Tue Nov 25 11:50:05 GMT+100 2008

Did you know ?
A small air-conditioning system uses about 1900 watts, a good ventilator about 100 watts!

What's that thing ??
This shows the real time consumption in watts of the main devices running in the Cudrefin02.ch office.

The dashboard can be implemented using any Web scripting language or tool (PHP, Ruby, Python, JavaScript, etc.) in the Ploggs Smart Gateway described before. The Energie visible application was built using Google Web Toolkit (GWT), which is a platform for developing JavaScript Web applications in Java, and provides a large number of easily customizable widgets. To display the current energy consumption in real time, the application simply sends HTTP GET requests to the gateway

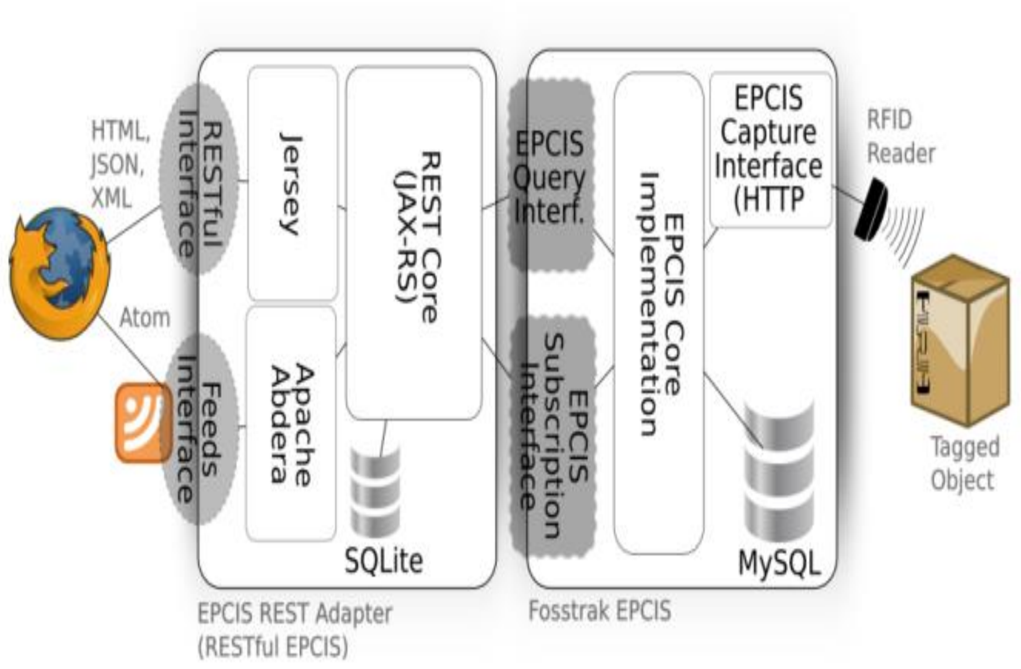
http://.../EnergieVisible/SmartMeters/all.json

on a regular basis or subscribes to this resource using Web hooks. The resulting feed entry is then dispatched to the corresponding graphs widgets, which can directly parse JSON, and extract the relevant data in it to be displayed. The aim of the project was to help visitors and members to better understand how much each device consumes in operation and in standby. The Ploggs are used to monitor the energy consumption of various devices, such as a fridge, a kettle, several printers, a file server, computers and screens. A large display in the office enables people passing by to experiment with the energy consumption of the devices. The staff can also access the user interface of any Plogg with the Web browser of their office computer.

5.4.2 Business Intelligence Mashup: RESTful EPCIS

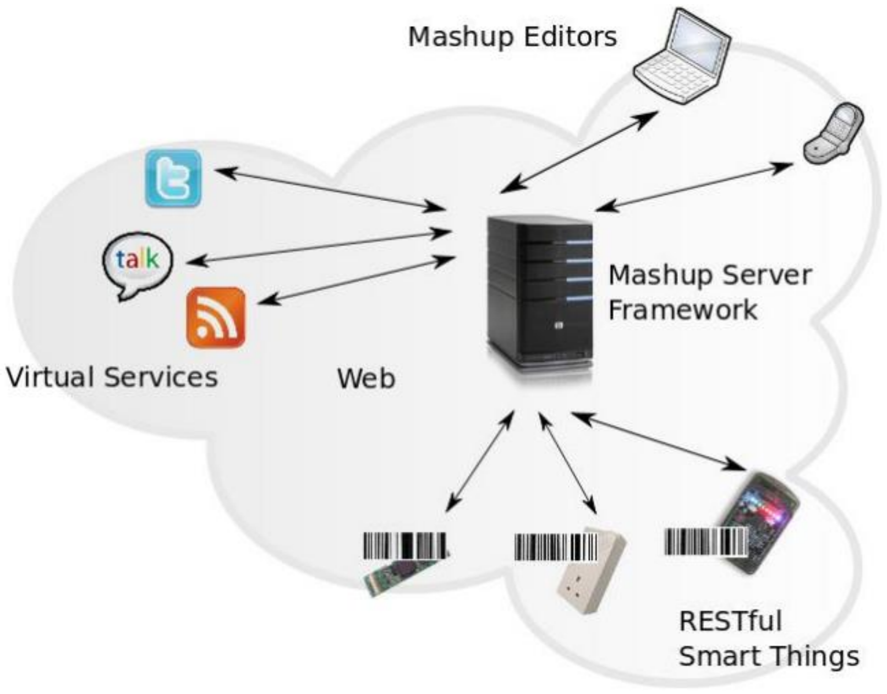
The Electronic Product Code (EPC) Network is a set of standards established by industrial key players towards a uniform platform for tracking and discovering RFID-tagged objects and goods in supply chains. This network offers a standardized server-side EPC Information Service (EPCIS) for managing and offering access to track and trace RFID events. Implementations of EPCIS provide a standard query and capture API through WS-* Web Services. In order to integrate not only embedded devices, but also RFID-tagged everyday items into the Web of Things, it uses the concepts presented to turn the EPCIS into a “Smart Gateway”. This helps to better grasp the benefits of a seamless Web integration based on REST, as opposed to using HTTP as a transport protocol only . The EPCIS offers three core features.

First, it offers an interface to query for RFID events. The WS-* interface, however, does not allow to directly query for RFID events using Web languages, such as JavaScript or HTML. More importantly, it does not allow to explore the EPCIS using a Web browser, or to search for tagged objects or exchange links pointing to traces of tagged objects. To remedy the problem, it was implemented a RESTful translation of the EPCIS WS-* interface. As shown in Figure, the RESTful EPCIS is a software module based on Jersey, a software framework for building RESTful applications. Clients of the RESTful EPCIS, such as browsers or Web applications, can query for tagged objects directly using REST and its uniform HTTP interface. Requests are then translated by the framework into WS-* calls on the standard.



5.4.3 A Mashup Editor for the Smart Home

Tech-savvy users can create Web mashups using “mashup editors”, such as Microsoft Popfly or Yahoo Pipes. These editors usually provide visual components representing Web sites and operations (add, filter, etc.) that users only needs to connect together to create new applications. The same principles will be applied to allow users to create physical mashups without requiring any programming skills, It was briefly introduced the physical mashup architecture and two mashup editors built on top of it. As shown in Figure, the system is composed of four main parts. First one is RESTful Web-enabled smart things and appliances. In this prototype, it was tagged with small 2D barcodes in order to ease their identification with mobile phones. Second, “virtual“ services on the Web, such as Twitter, Google Visualization API, Google Talk, etc. In the middle, the mashup server framework allows to compose services of different smart appliances as well as virtual services on the Web. It is incharge of executing the workflows created by end-users in their mashup applications. It discovers, listens, and interacts with the devices over their RESTful API. The last components are the mashup editors themselves, which allow users to create mashup applications very easily.



Two mashup editors implemented using this architecture. The first one is based on the Clickscript project. A Firefox plugin written on top of an Ajax library allows people to visually create Web mashups by connecting building blocks of resources (Web sites) and operations (greater than, if/then, loops, etc.). Since it is written in JavaScript, Clickscript cannot use resources based on proprietary service protocols. However, it can easily access RESTful services, such as those provided by Web-enabled smart appliances. This makes it straightforward to create Clickscript building blocks that represent smart appliances. The mashup gets the room temperature by GETting the temperature resource. If it is below 36 degrees, it turns off the Web-enabled air-conditioning system.

The second editor was implemented on the Android Mobile Phone. Once again, thanks to the support of HTTP in Android, RESTful communication with smart appliances was straightforward. Similarly to Clickscript, the mobile editor allows the creation of simple mashups. However, due to the screen constraints of the mobile phone, a mashup is created by going through a wizard. Users first select the appliances they want to include in the mashup. They do this simply by scanning a barcode on the appliance using the phone's camera. These codes are basically pointing back to the root URLs of the appliance's RESTful APIs. They then set up the rules they want to implement and the virtual services they want to interact with. For example, users can create a mashup that switches on their appliances, e.g, turning the heating up, whenever their phone detects that they are moving towards home (based on their GPS traces).

5.5 Advanced Concepts: The Future Web of Things

Even though there are many web standards and design principles that are leveraged for smart things, there are many open challenges remain. Three such challenges discussed are

1. Needs for real-time data of many smart things applications.
2. Finding and understanding services available in a global Web of Things.
3. Mechanisms for sharing smart things.

5.5.1 Real-Time Web of Things

HTTP is a stateless client/server protocol, This interaction model is well-suited for control-oriented applications where clients read/write data from/to embedded devices. However, this client initiated interaction models seem inappropriate for bi-directional event-based and streaming systems, where data must be sent asynchronously to the clients as soon as it is produced. Many pervasive scenarios must deal with real-time information to combine stored or streaming data from various sources to detect spatial or temporal patterns, as is the case in many environmental monitoring applications. As such applications are often event-based and embedded devices usually have a low-duty cycle (i.e., sleep most of the time), smart things should also be able to push data to clients (rather than being continuously polled). To support the complex, data centric queries required for such scenarios, more flexible data models are required to expose sensor data streams over the Web.

Recent developments in the real-time Web to build such a data model that is more suited to the data-centric, stream-based nature of sensor-driven applications. As mentioned before, using syndication protocols, such as Atom, improves the model when monitoring, since devices can publish data asynchronously using AtomPub on an intermediate server or Smart Gateway. Nevertheless, clients still have to pull data from Atom servers. Web streaming media protocols (RTP/RTSP) have enabled transmission of potentially infinite data objects, such as Internet radio stations. Sensor streams are similar to streaming media in this respect. However, streaming media mainly support play and pause commands, which is insufficient for sensor streams where more elaborate control commands are needed.

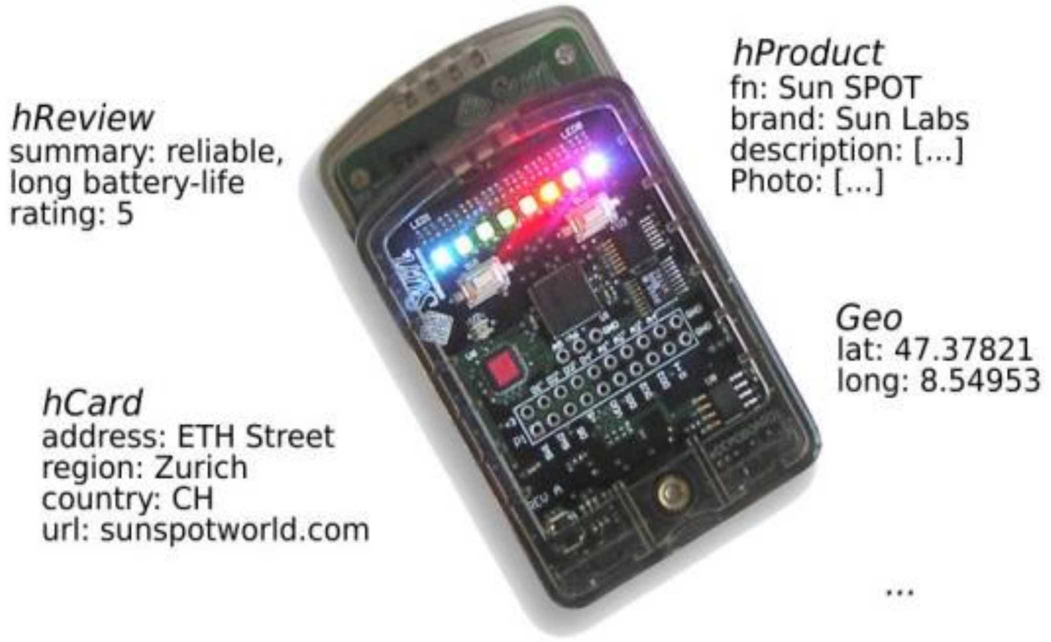
The Extensible Messaging and Presence Protocol (XMPP) is an open standard for real-time communication based on exchanges of XML messages, and powers a wide range of applications including instant messaging. Although widely used and successful, XMPP is a fairly complex standard, which is often too heavy for the limited resources of embedded devices used in sensor networks. This model, called Comet, enables a Web server to push data back to the browser without the client requesting it explicitly. Since browsers are not designed with server-sent events in mind, Web application developers have tried to work around several specification loopholes to implement Comet-like behavior, each with different benefits and drawbacks. One general idea is that a Web server does not terminate the TCP connection after response data has been served to a client, but leaves the connection open to send further events.

The recent developments in Web techniques have allowed to build efficient and scalable publish/subscribe systems, It is suggested that a Web-based pub/sub model could be used to connect sensor networks with applications. PubSubHubbub (PuSH) is a simple, open pub/sub protocol as an extension to Atom and RSS. Parties (servers) speaking the PuSH protocol can get near-instant notifications (via callbacks) when a feed they are interested in is updated. The following model can be used to enable Web-based stream processing applications where users can post queries using an HTTP request to one or more sensors. The HTTP request collects the light and temperature sensor readings twice per second only if the light sensor value is not over "200" and the temperature reading is less than "19". All the data samples corresponding to these queries are then pushed into a feed on the message broker, where users can subscribe using the PuSH protocol. They will then receive the data from the stream pushed from the broker via callbacks.

5.5.2 Finding and Describing Smart Things

- Another major challenge for a global Web of Things is searching and finding relevant devices among billions of smart things that will be connected to the Web.

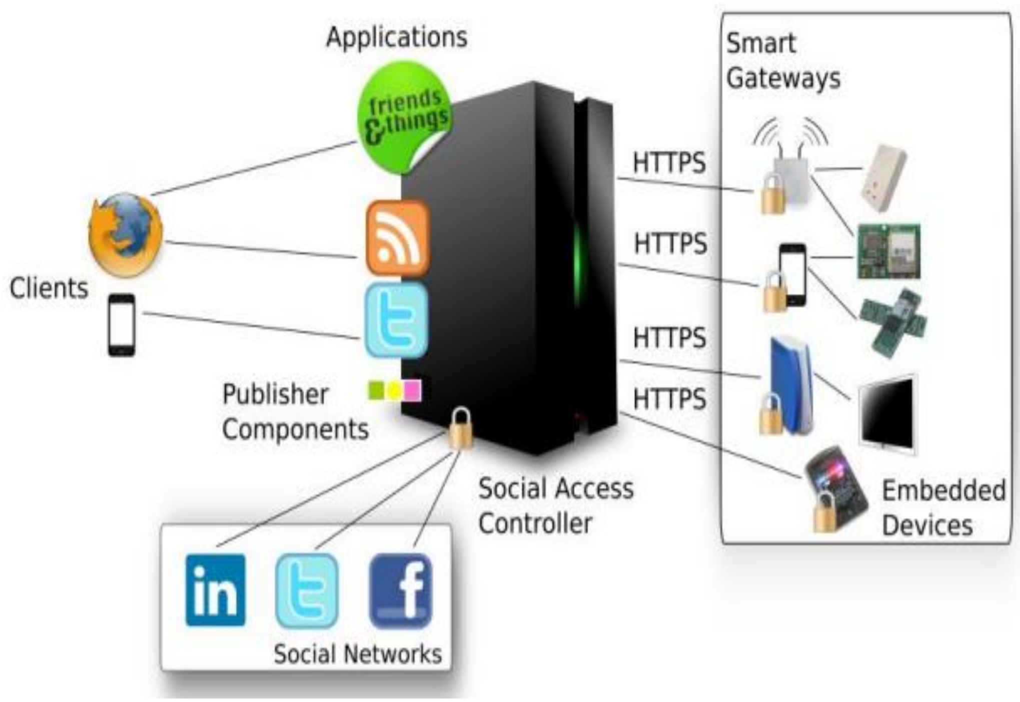
- Finding them by browsing HTML pages with hyperlinks is literally impossible in this case, hence the idea of searching for smart things.
- Searching for things is significantly more complicated than searching for documents, as things are tightly bound to contextual information, such as location, are often moving from one context to another, and have no obvious easily indexable properties, such as human readable text in the case of documents.
- Beyond location, smart things need a mechanism to describe themselves and their services to be (automatically) discovered and used.
- But what is the best way to describe a thing on the Web so that both, humans and machines, can understand what services it provides?
- This problem is not inherent to smart things, but more generally a complex problem of describing services, which has always been an important challenge to be tackled in the Web research community, usually in the area of the Semantic Web.
- To overcome the limited descriptive power of resources on the Web, several languages have been proposed, such as RDF(Resource Description Format) or Microformats, designed for both, human and machines.
- Microformats provide a simple way to add semantics to Web resources.
- There is not one single Microformat, but rather a number of them, each one for a particular domain; a “geo” and “adr” microformat for describing places or an “hProduct” and “hReview” microformat for describing products.
- Each Microformat undergoes a “standardization” process that ensures its content to be widely understood and used, if accepted.
- Microformats are especially interesting in a Web of Things for two reasons; first they are directly embedded into Web pages and thus can be used to semantically annotate the HTML representation of a thing’s RESTful API.
- Secondly, Microformats (as well as RDFa) are increasingly supported by search engines, such as Google and Yahoo, where it is used to enhance the search results.
- For example, the “Geo” microformat could be used to localize search results close to you or, in our context, to to localize smart things in your direct vicinity.
- As an example, in Figure we use 5 microformats to describe a Sun SPOT and embed this semantic information directly in the HTML representation of the SPOT resources localize smart things in your direct vicinity.



Compound Microformats for Describing a Sun SPOT Using the Geo, hCard, hProduct and hReview Microformats

5.5.3 Sharing Smart Things

- The success of Web 2.0 mashups depends on the trend for Web 2.0 service providers (e.g., Google, Twitter, Wordpress, etc.) to provide access to some of their services through relatively simple, often RESTful, open APIs on the Web.
- Mashup developers often share their mashups on the Web and expose them through open APIs as well, making the service ecosystem grow with each application and mashup.
- The following Figure shows the simplified component architecture of a Social Access Controller (SAC), which serves as authentication proxy between clients and smart things.
- However, enabling such an open model for a Web of Things requires a sharing mechanism for physical things supporting access control to the RESTful services provided by devices.
- For example, one could share the energy consumption sensors in one's house with the community.
- However, this is a potentially risky process, given that these devices are part of our everyday life and their public sharing might result in serious privacy implications.



- HTTP already provides authentication mechanisms based on credentials and server-managed user groups. While this solution is already available for free on most Web servers, it still presents a number of drawbacks in the WoT context.
- First, for a large number of smart things it becomes quite unmanageable to share credentials for each of them.
- Then, as the shared resources are not advertised anywhere, sharing also requires the use of secondary channels, such as sending emails containing credentials to people.
- Several platforms, such as SenseWeb or Pachube propose to overcome these limitations by providing a central platform for people to share their sensor data.
- However, these approaches are based on a centralized data repository and are not designed to support decentralization and direct interaction with smart things.
- A promising solution is to leverage existing social structures of social networks (e.g., Facebook, Linkedin, Twitter, etc.) and their (open) APIs to share things.
- Using social networks enables users to share things with people they know and trust (e.g., relatives, friends, colleagues, fellow researchers, etc.), without the need to recreate yet another social network or user database from scratch on a new online service.
- Additionally, this enables advertising and sharing through a unique channel: you can use various well-known social networks to inform your friends about the sensors you shared with them by automatically posting messages to their profile or newsfeed.

- The SAC platform is an implementation of this idea. SAC is an authentication proxy between clients (e.g., Web browsers) and smart things.