Introduction to IoT

Dr. M. Kalpana Devi
Professor

Outline

- IoT definition
- Characteristics of IoT
- Physical Design of IoT
- Logical Design of IoT
- IoT Protocols
- IoT Levels & Deployment Templates

Definition of IoT

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

Characteristics of IoT

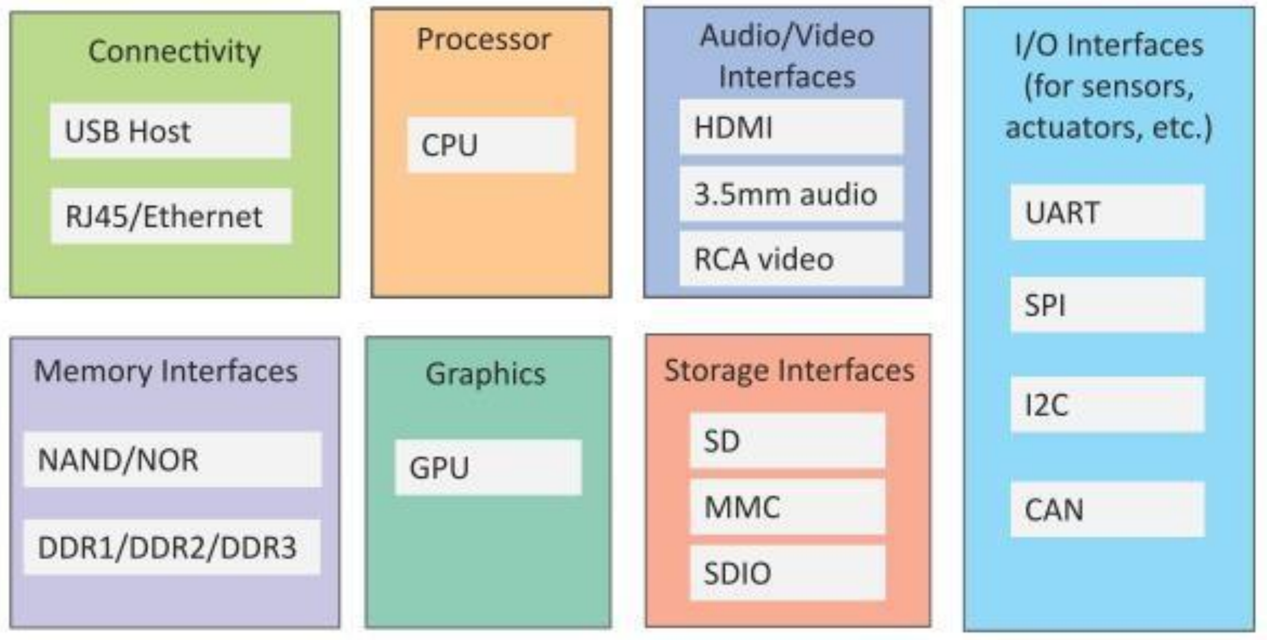
- Dynamic & Self-Adapting
- Self-Configuring
- Interoperable Communication Protocols
- Unique Identity
- Integrated into Information Network

Physical Design of IoT

- The "Things" in IoT usually refers to IoT devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities.
- IoT devices can:
  - Exchange data with other connected devices and applications (directly or indirectly), or
  - Collect data from other devices and process the data locally or
  - Send the data to centralized servers or cloud-based application back-ends for processing the data, or
  - Perform some tasks locally and other tasks within the IoT infrastructure, based on temporal and space constraints

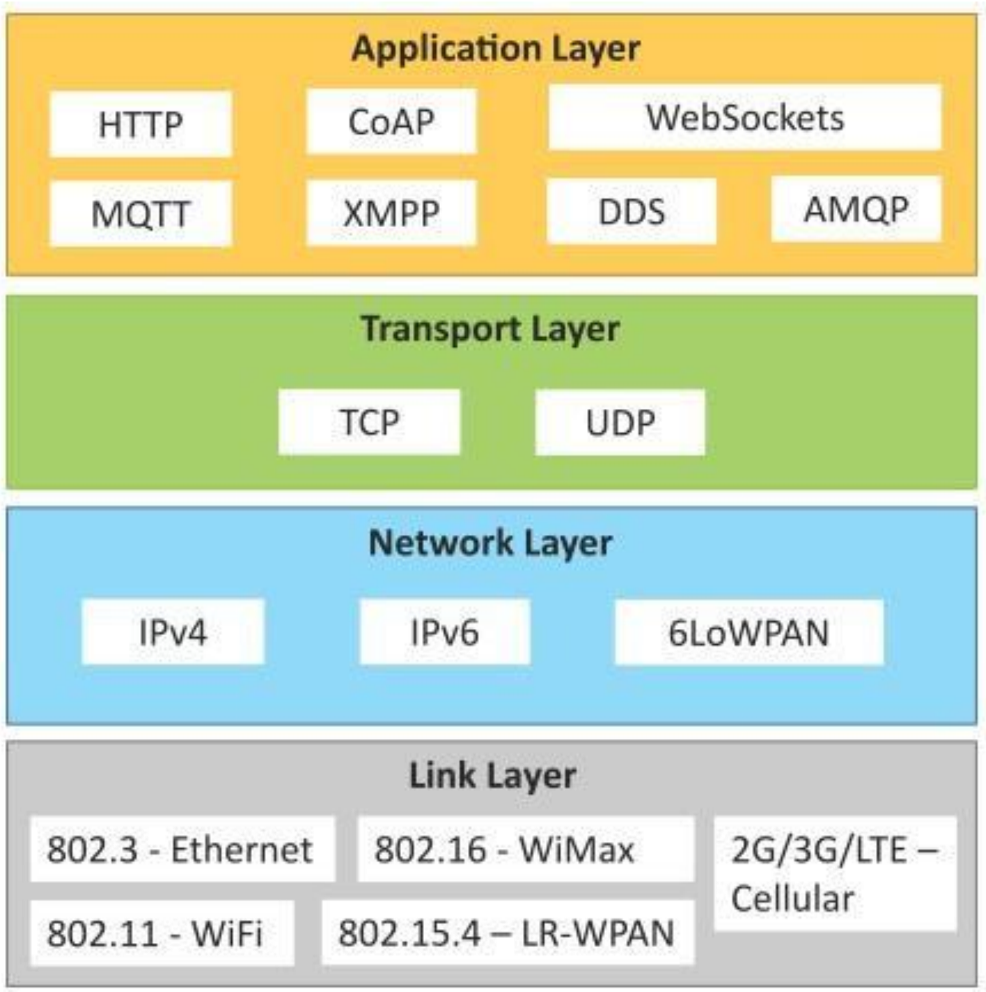
Generic block diagram of an IoT Device

- An IoT device may consist of several interfaces for connections to other devices, both wired and wireless.
  - I/O interfaces for sensors
  - Interfaces for Internet connectivity
  - Memory and storage interfaces
  - Audio/video interfaces.



IoT Protocols

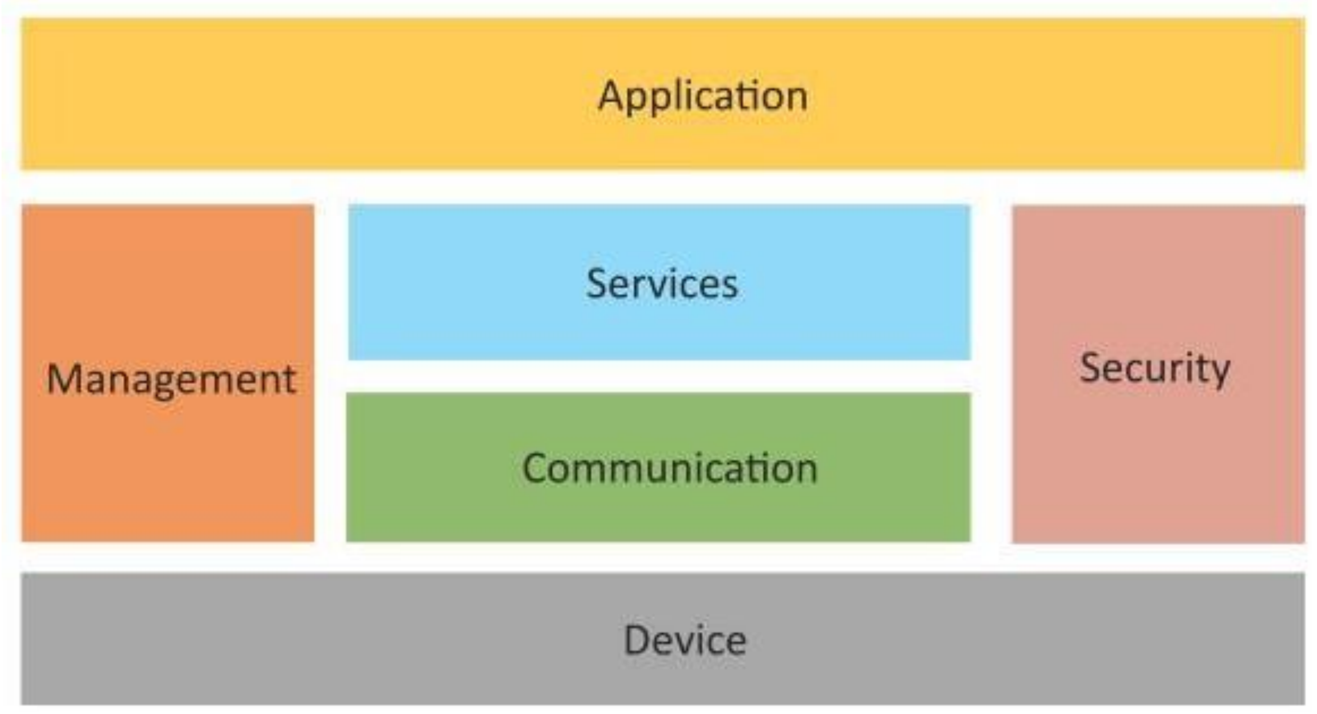
- Link Layer
  - 802.3 – Ethernet
  - 802.11 – WiFi
  - 802.16 – WiMax
  - 802.15.4 – LR-WPAN
  - 2G/3G/4G
- Network/Internet Layer
  - IPv4
  - IPv6
  - 6LoWPAN
- Transport Layer
  - TCP
  - UDP
- Application Layer
  - HTTP
  - CoAP
  - WebSocket
  - MQTT
  - XMPP
  - DDS
  - AMQP



Logical Design of IoT

- Logical design of an IoT systemrefers to an abstract representation of the entities and processes without going into the low-level specifics of the implementation.

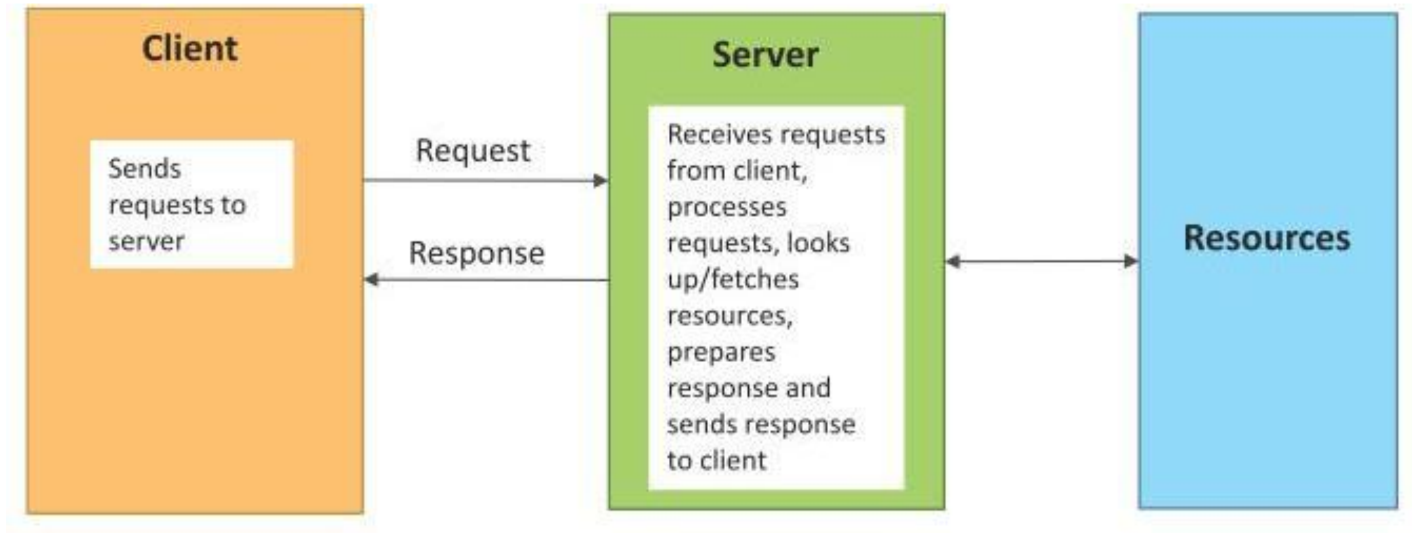
- An IoT system comprises of a number of functional blocks that provide the system the capabilities for identification, sensing, actuation, communication, and management.



Request-Response communication model

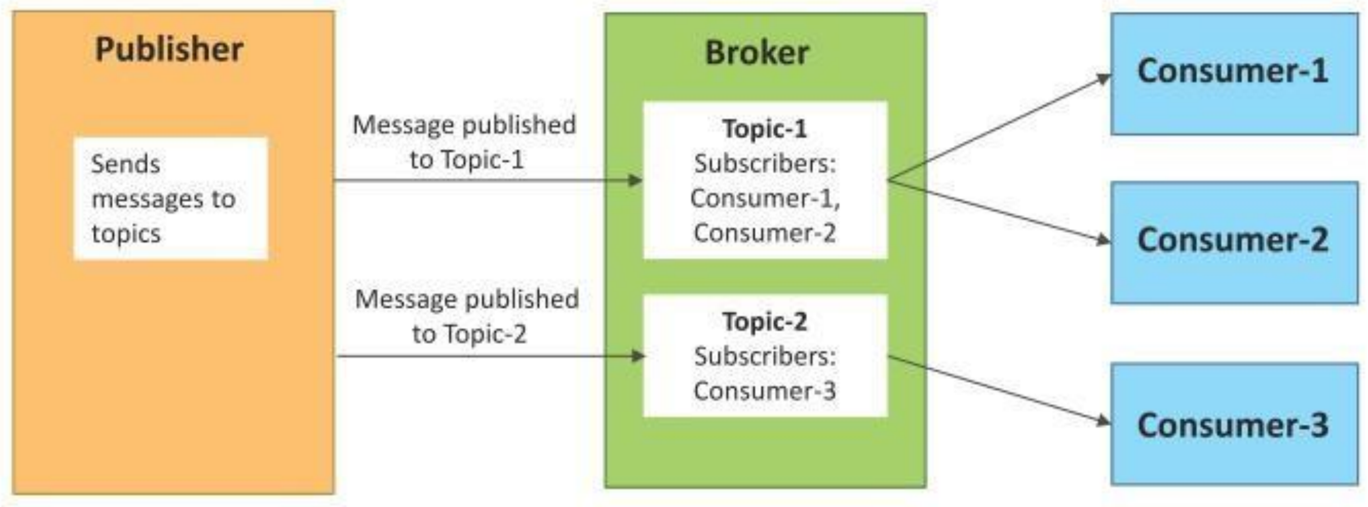
- Request-Response is a communication model in which the client sends requests to the server and the server responds to the requests.

- When the server receives a request, it decides how to respond, fetches the data, retrieves resource representations, prepares the response, and then sends the response to the client.


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

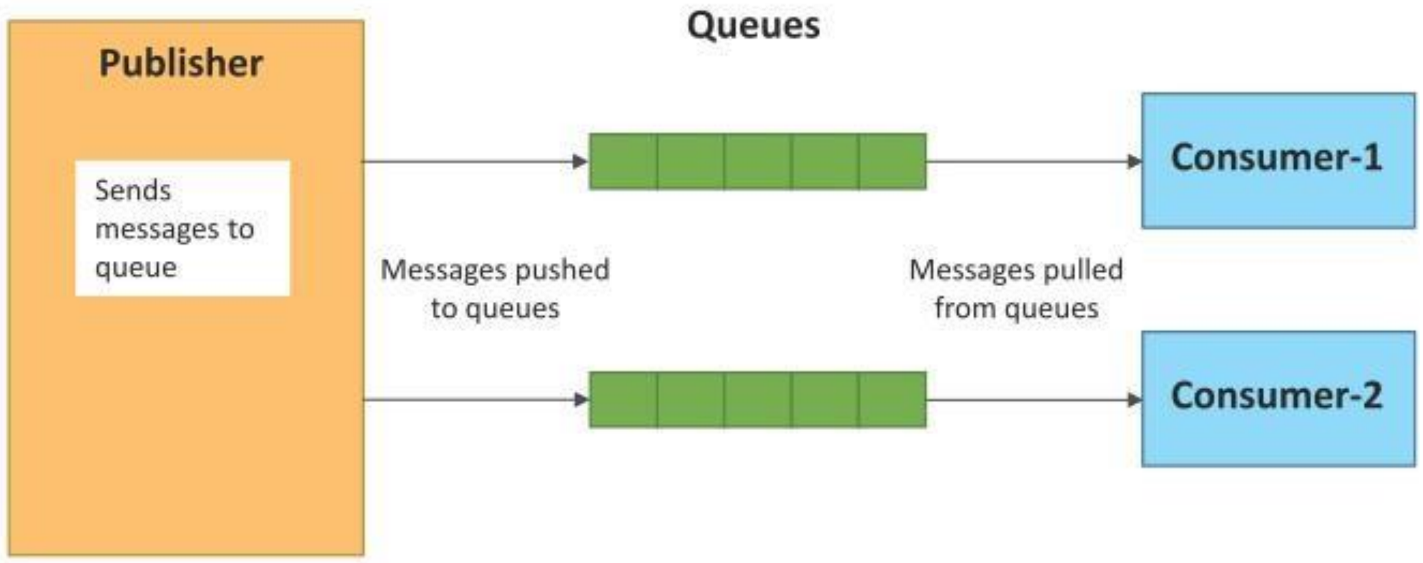
Publish-Subscribe communication model

- Publish-Subscribe is a communication model that involves publishers, brokers and consumers.
- Publishers are the source of data. Publishers send the data to 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.


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

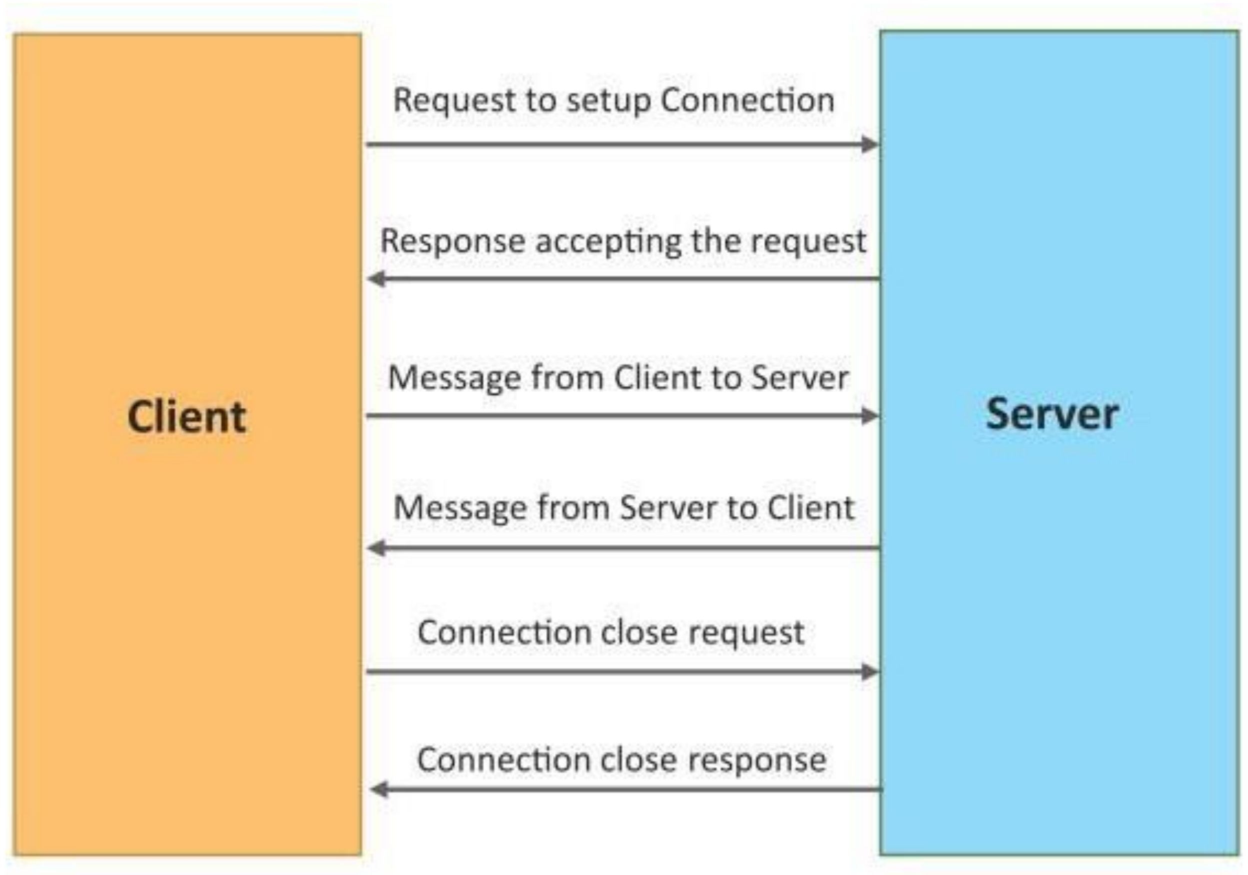
Push-Pull communication model

- Push-Pull is a communication model in which the data producers push the data to queues and the consumers pull the data from the queues. Producers do not need to be aware of the consumers.
- Queues help in decoupling the messaging between the producersand consumers.
- Queues also act as a buffer which helps in situations when there is a mismatch between the rate at which the producers push data and the rate rate at which the consumers pull data.



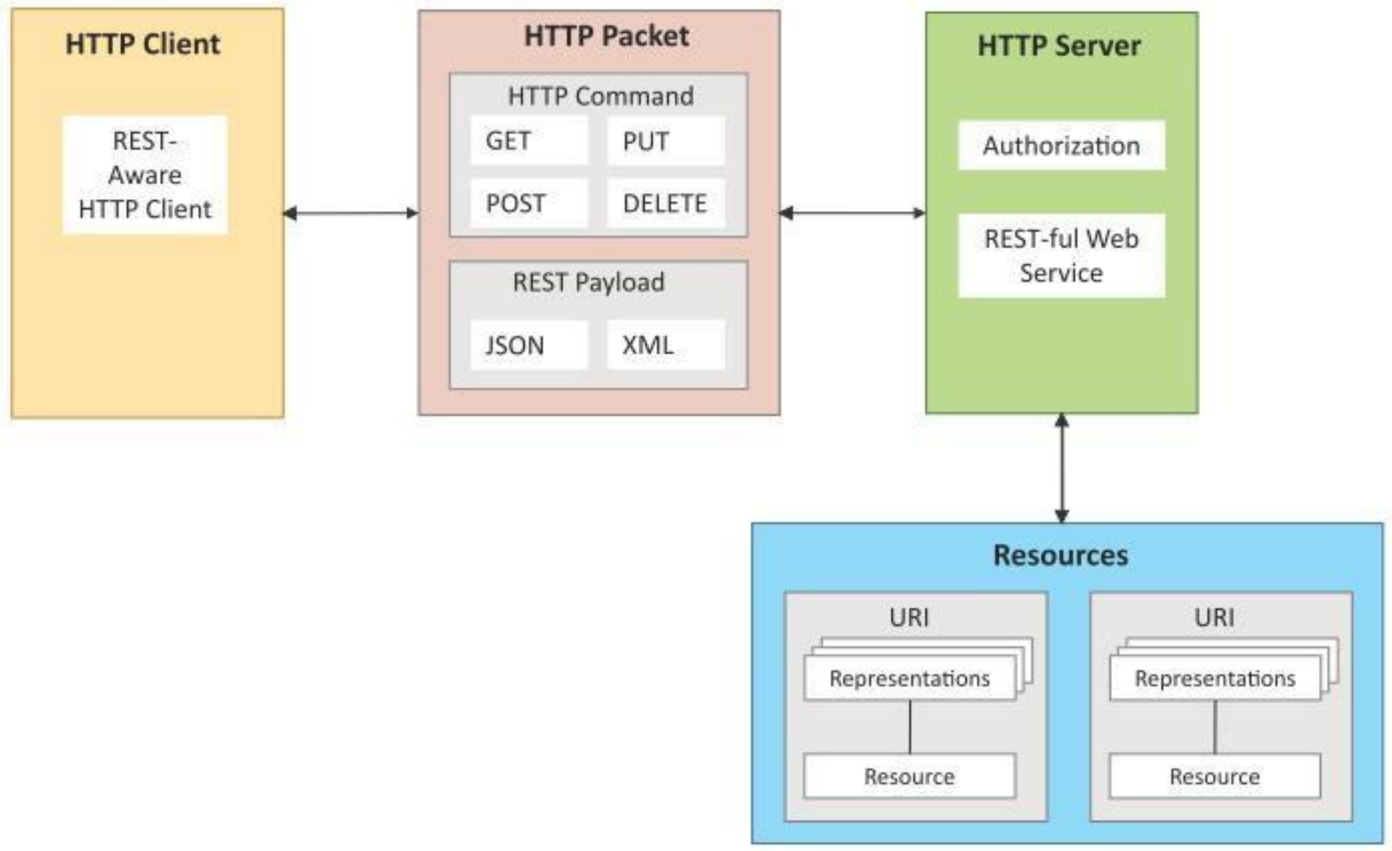
Exclusive Pair communication model

- Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server.
- Once the connection is setup it remains open until the client sends a request to close the connection.
- Client and server can send messages to each other after connection setup.


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

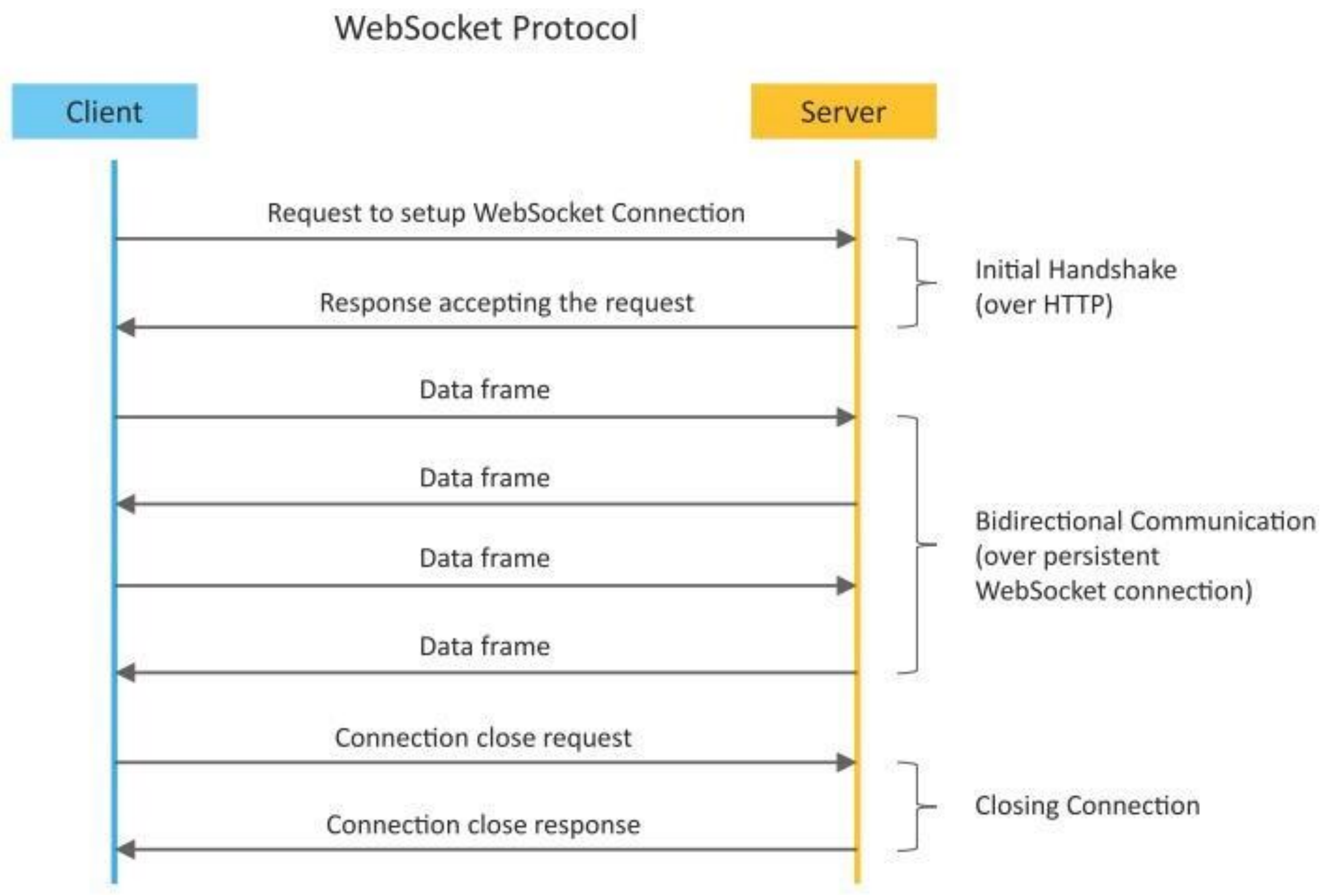
REST-based Communication APIs

- Representational State Transfer (REST) is a set of architectural principles by which you can design web services and web APIs that focus on a system’s resources and how resource states are addressed and transferred.
- REST APIs follow the request-response communication model.
- The REST architectural constraints apply to the components, connectors, and data elements, within a distributed hypermedia system.



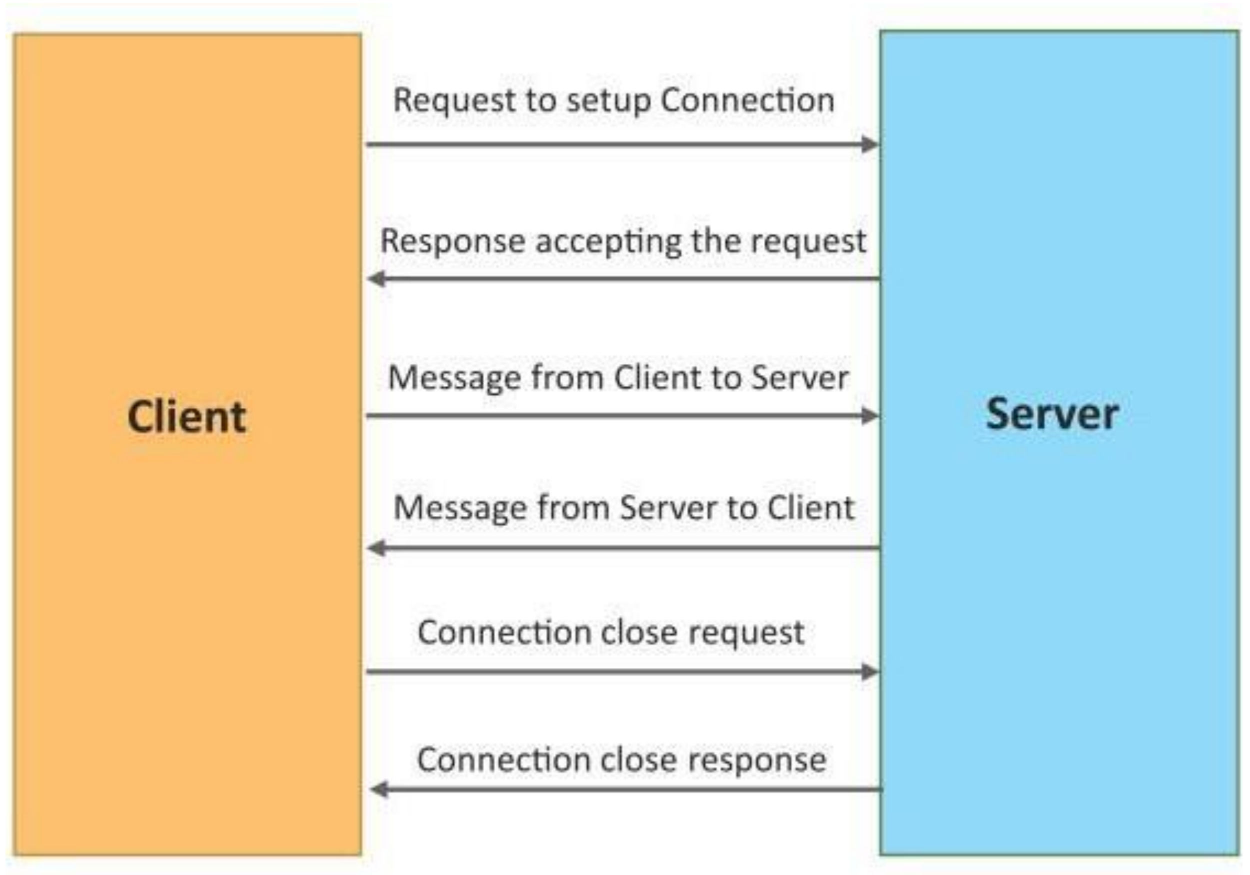
WebSocket-based Communication APIs

- 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

Exclusive Pair communication model

- Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server.
- Once the connection is setup it remains open until the client sends a request to close the connection.
- Client and server can send messages to each other after connection setup.


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

IoT Levels & Deployment Templates

An IoT system comprises of the following components:

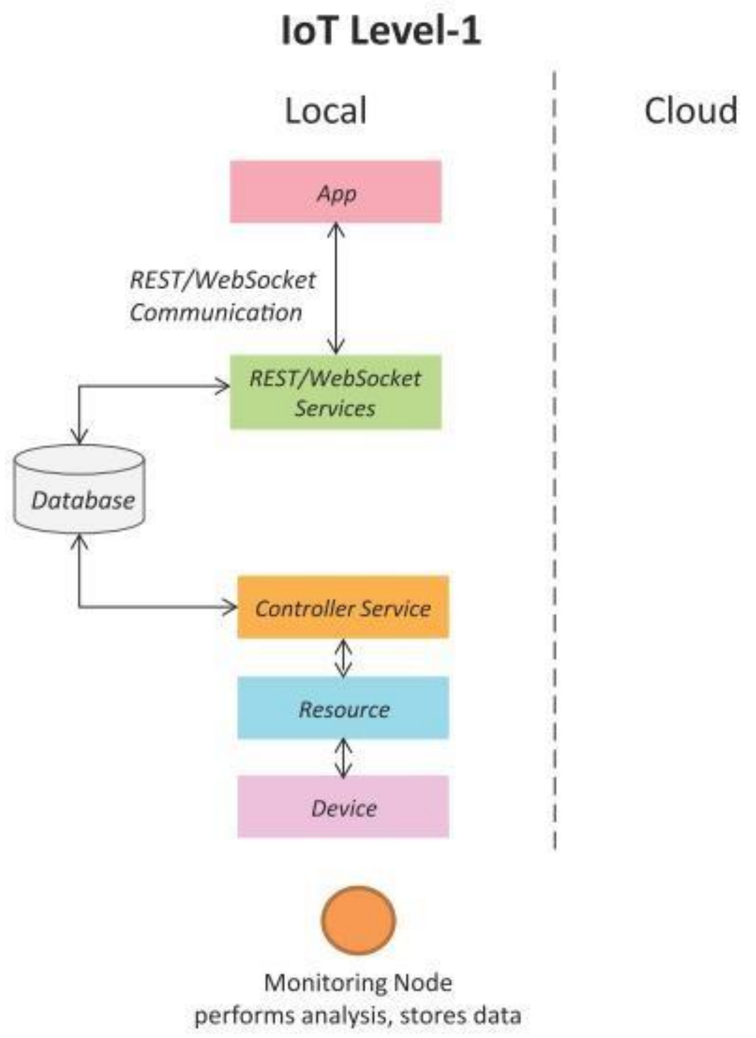
- **Device**: An IoT device allows identification, remote sensing, actuating and remote monitoring capabilities. You learned about various examples of IoTdevices in section
- **Resource**: Resources are software components on the IoT device for accessing, processing, and storing sensor information, or controlling actuators connected to the device. Resources also include the softwarecomponents that enable network access for the device.
- **Controller Service**: Controller service is a native service that runs on thedevice and interacts with the web services. Controller service sends datafrom the device to the web service and receives commands from the application (via web services) for controlling the device.

IoT Levels & Deployment Templates

- **Database**: Database can be either local or in the cloud and stores the data generated by the IoT device.
- **Web Service**: Web services serve as a link between the IoT device, application, database and analysis components. Web service can be eitherimplemented using HTTP and REST principles (REST service) or using WebSocket protocol (WebSocket service).
- **Analysis Component**: The Analysis Component is responsible for analyzing the IoT data and generate results in a form which are easy for the user to understand.
- **Application**: IoT applications provide an interface that the users can use tocontrol and monitor various aspects of the IoT system. Applications also allow users to view the system status and view the processed data.

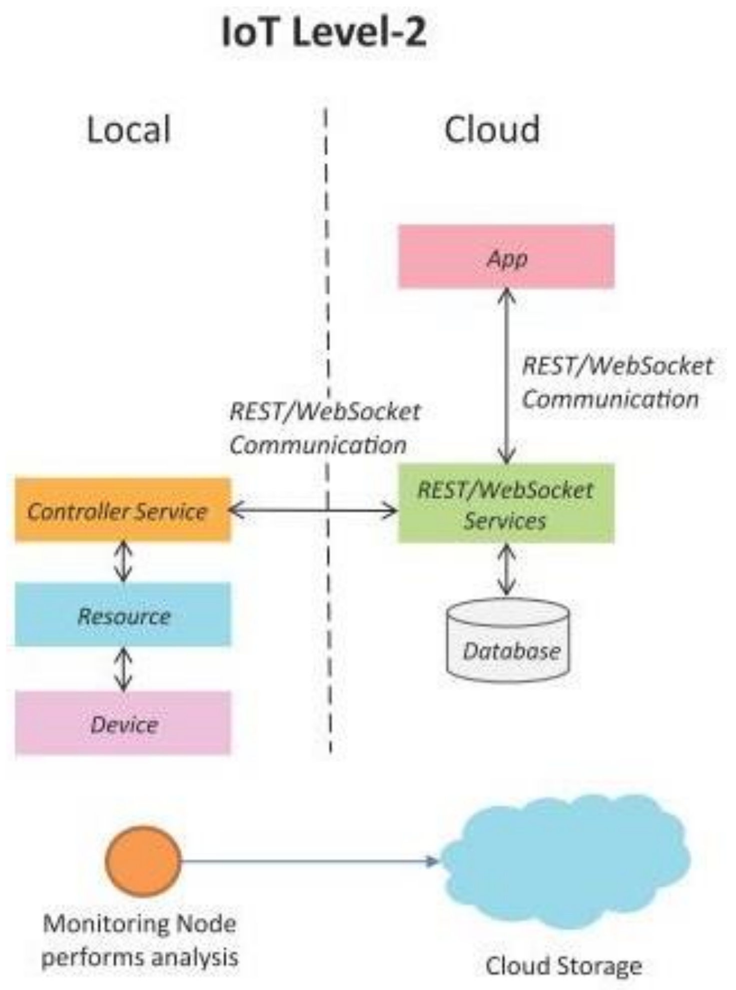
IoT Level-1

- A level-1 IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application
- Level-1 IoT systems are suitable for modeling low-cost and low-complexity solutions where the data involved is not big and the analysis requirements arenot computationally intensive.



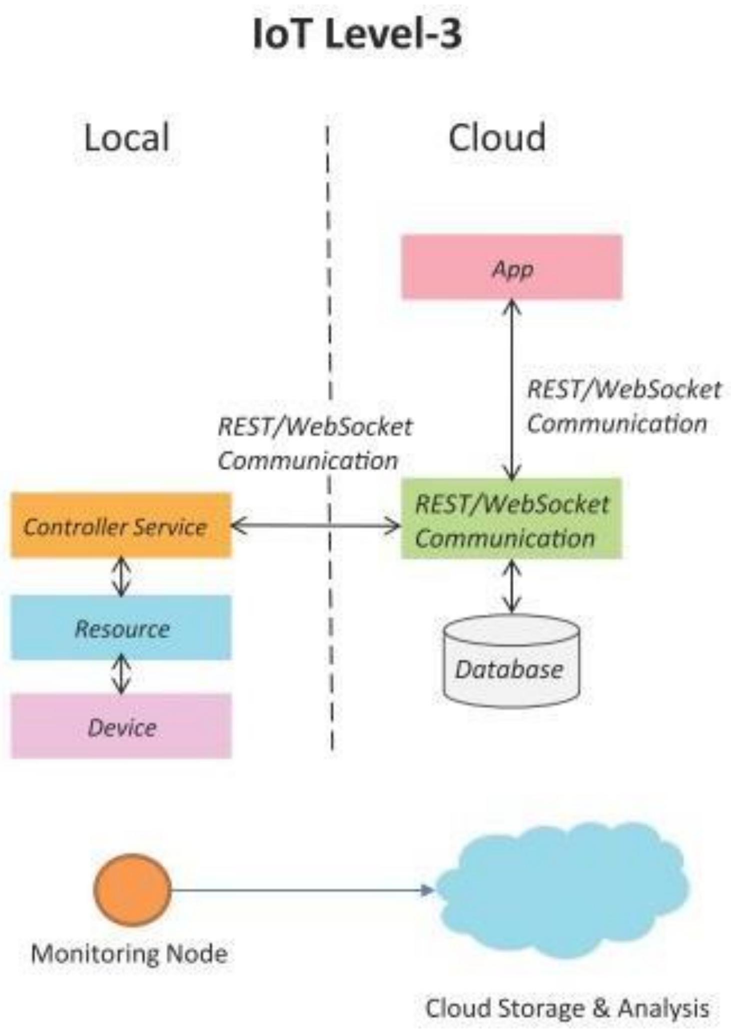
IoT Level-2

- A level-2 IoT system has a single node that performs sensing and/or actuation and local analysis.
- Data is stored in the cloud and application is usually cloud-based.
- Level-2 IoT systems are suitable for solutions where the data involved is big, however, the primary analysis requirement is not computationally intensive and can be done locally itself.



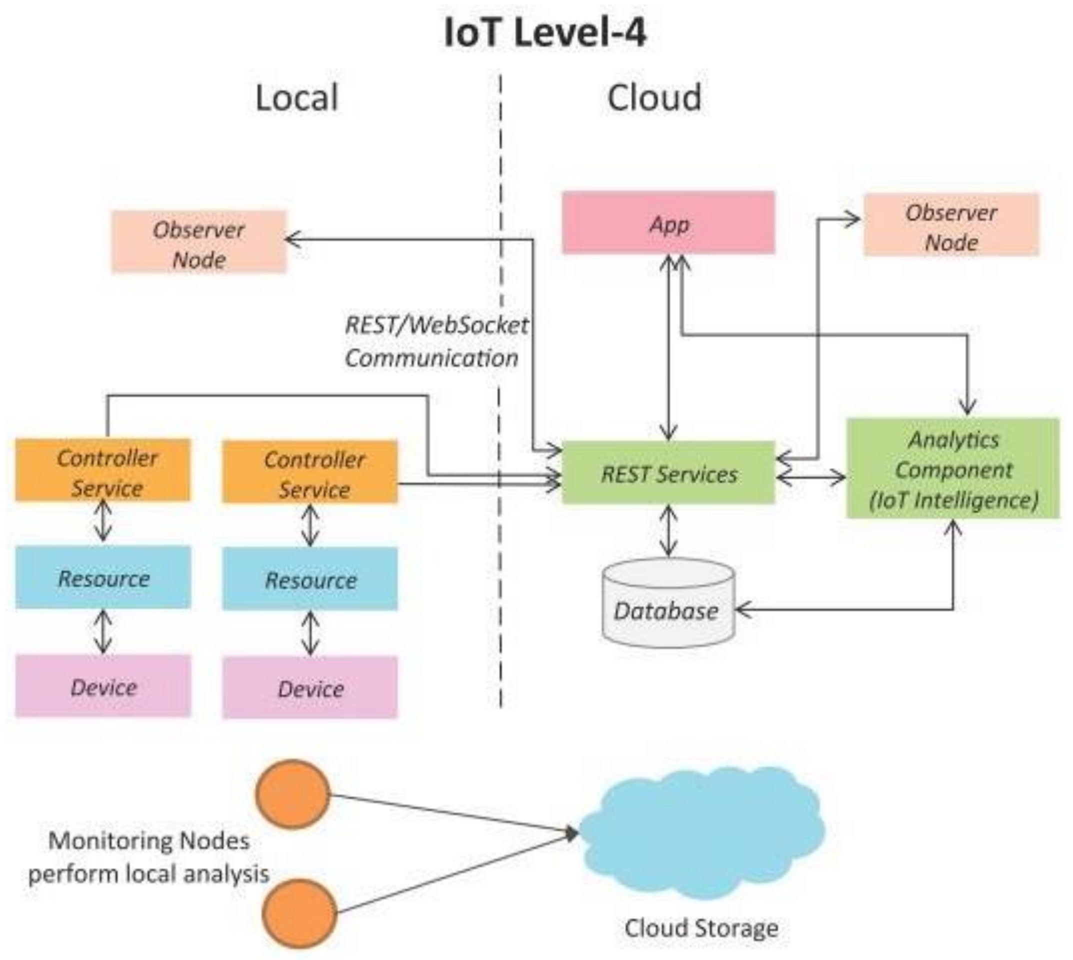
IoT Level-3

- A level-3 IoT system has a single node. Data is storedand analyzed in the cloud and application is cloud- based.
- Level-3 IoT systems are suitable for solutions where the data involved isbig and the analysis requirements are computationally intensive.



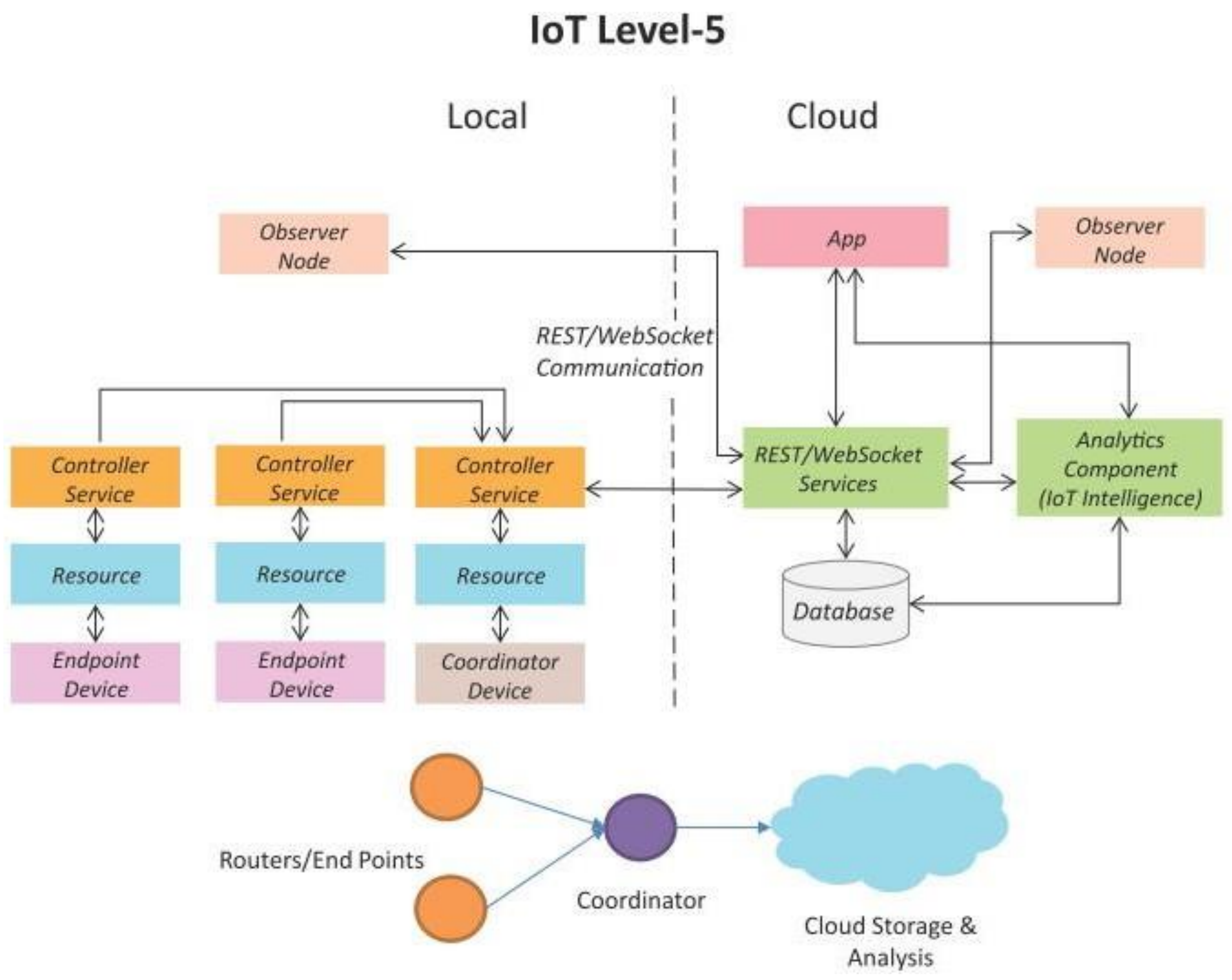
IoT Level-4

- A level-4 IoT system has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud-based.
- Level-4 contains local and cloud-based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices.
- Level-4 IoT systems are suitable for solutions where multiple nodes are required, the data involved is big and the analysis requirements are computationallyintensive.



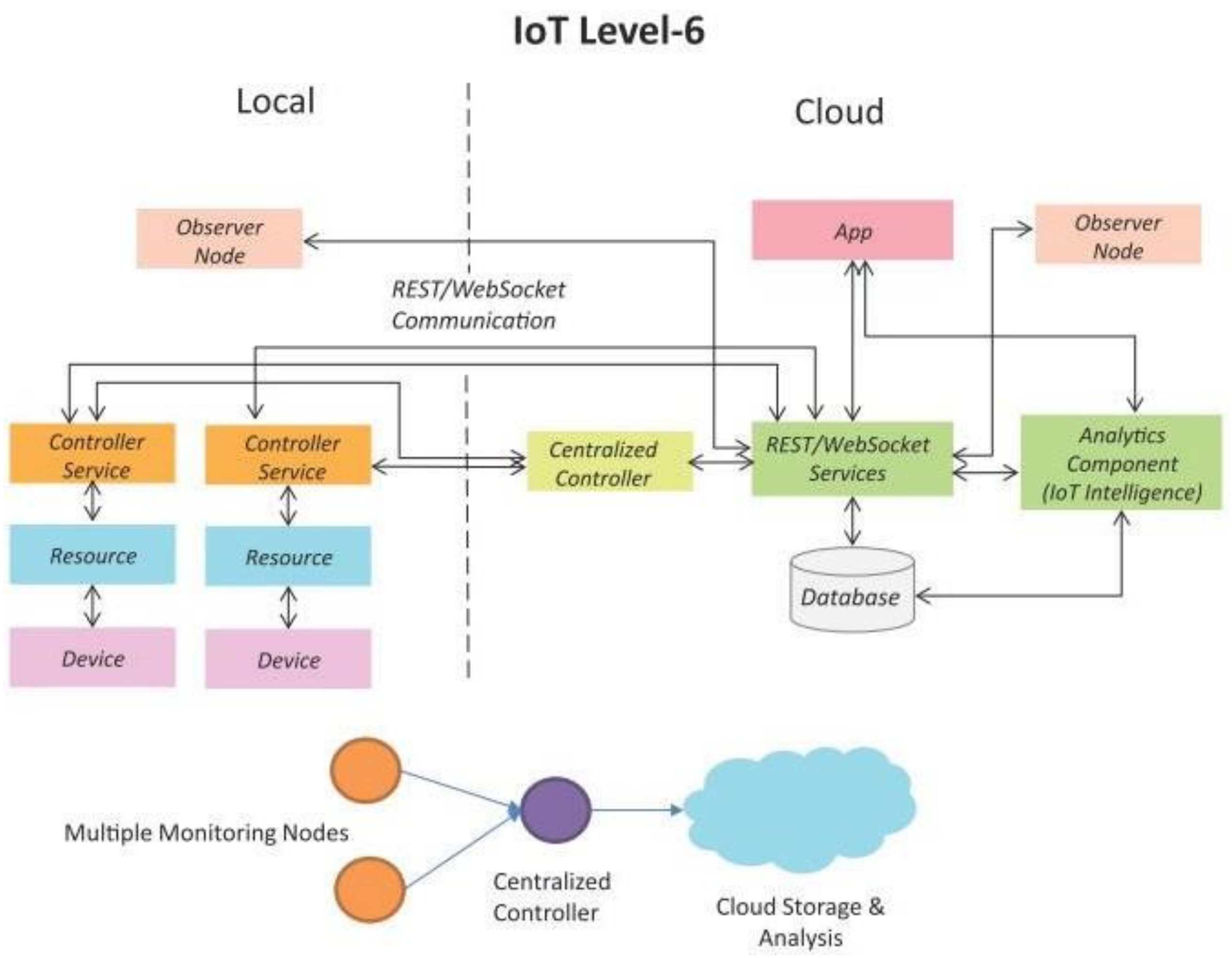
IoT Level-5

- A level-5 IoT system has multiple end nodes and one coordinator node.
- 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 application is cloud-based.
- Level-5 IoT systems are suitable for solutions based on wireless sensor networks, in which the data involvedis big and the analysis requirements are computationally intensive.



IoT Level-6

- A level-6 IoT system has multiple independent end nodes that perform sensing and/or actuation and send data to the cloud.
- Data is stored in the cloud and application is cloud-based.
- The analytics component analyzes the data and stores the results in the cloud database.
- The results are visualized with the cloud-based application.
- The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes.

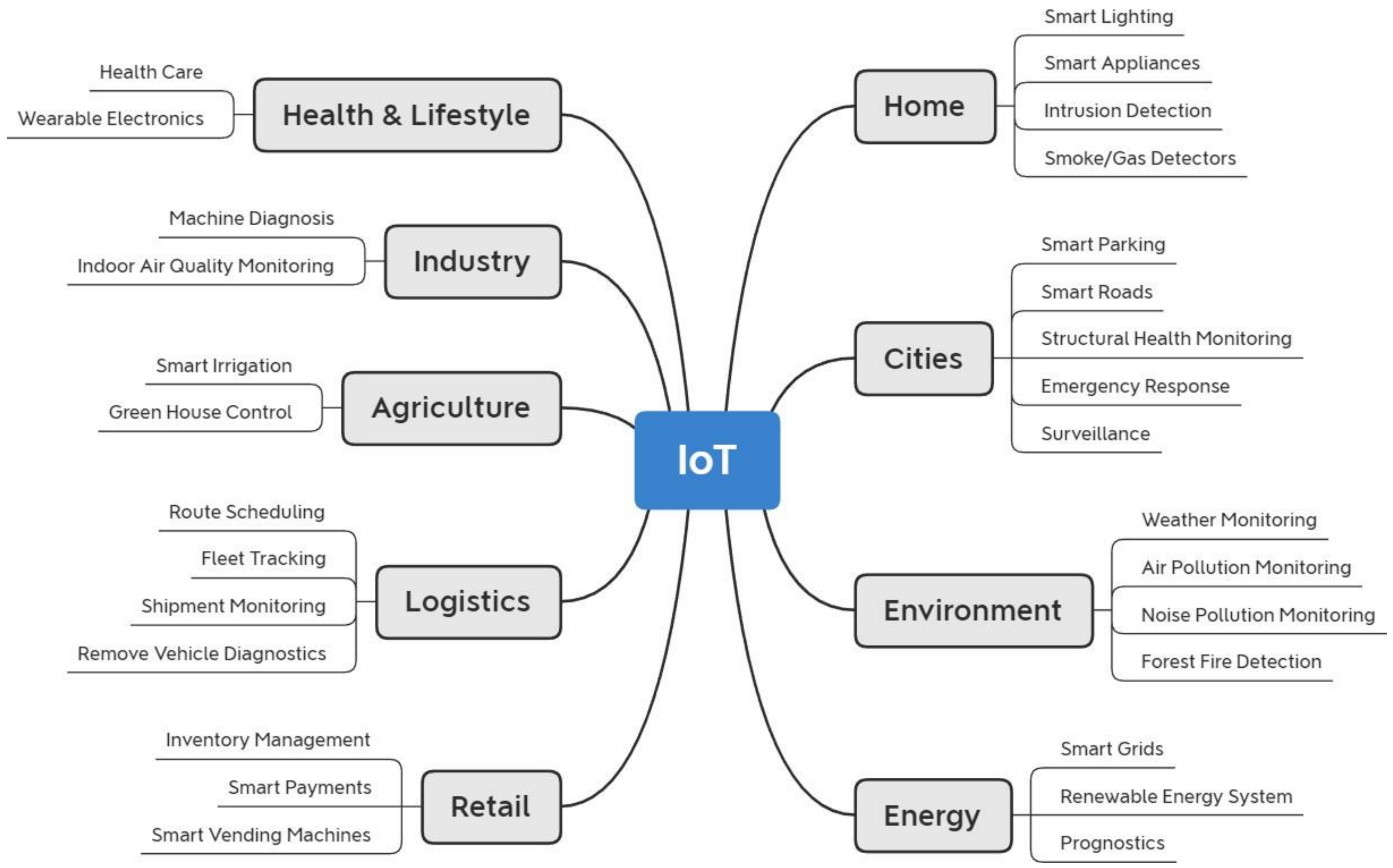


Domain Specific IoTs

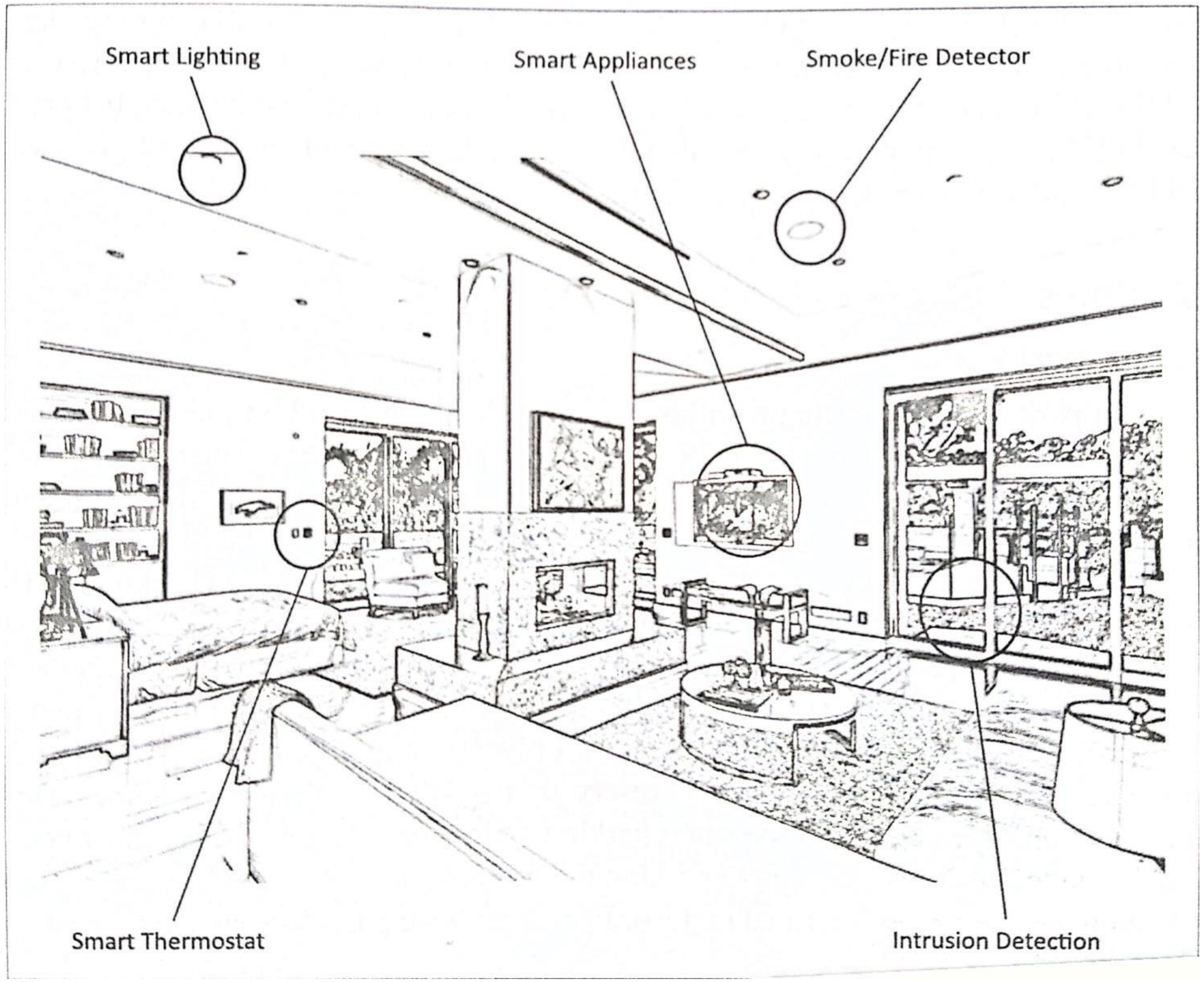
Outline

- Introduction
- Home Automation
- Cities
- Environment
- Energy
- Retail
- Logistics
- Agriculture
- Industry
- Health & Lifestyle

Introduction – Applications of IoT



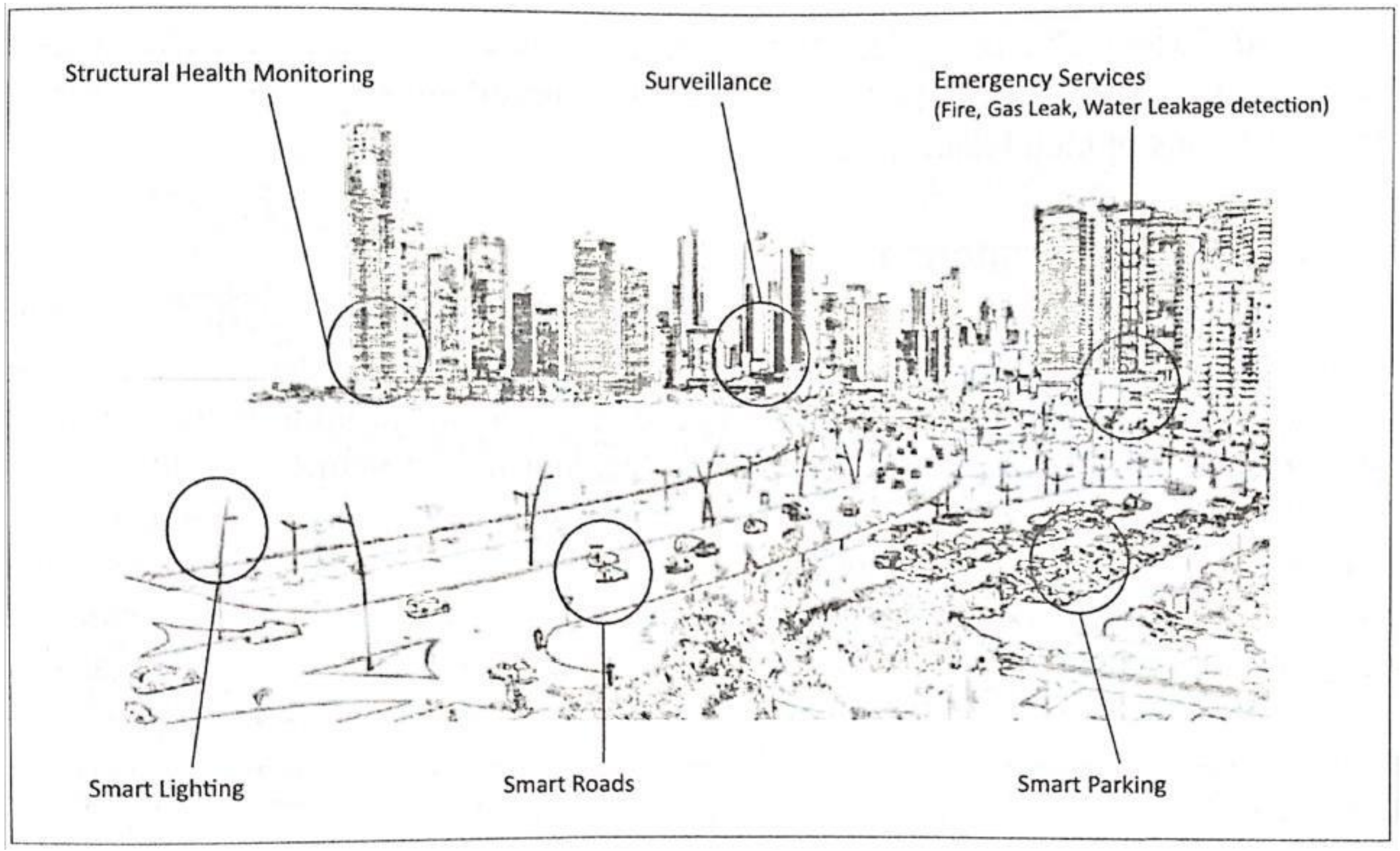
Home Automation



Home Automation (2/2)

- Smart Lighting
  - Control lighting by remotely (mobile or web applications)
- Smart Appliances
  - Provide status information to the users remotely
- Intrusion Detection
  - Use security cameras and sensors (PIR sensors and door sensors)
  - Detect intrusions and raise alerts
  - The alerts form: an SMS or an email sent to the user
- Smoke/Gas Detectors
  - Use optical detection, ionization, or air sampling techniques to detect the smoke
  - Gas detectors can detect harmful gases
    - Carbon monoxide (CO)
    - Liquid petroleum gas (LPG)
  - Raise alerts to the user or local fire safety department

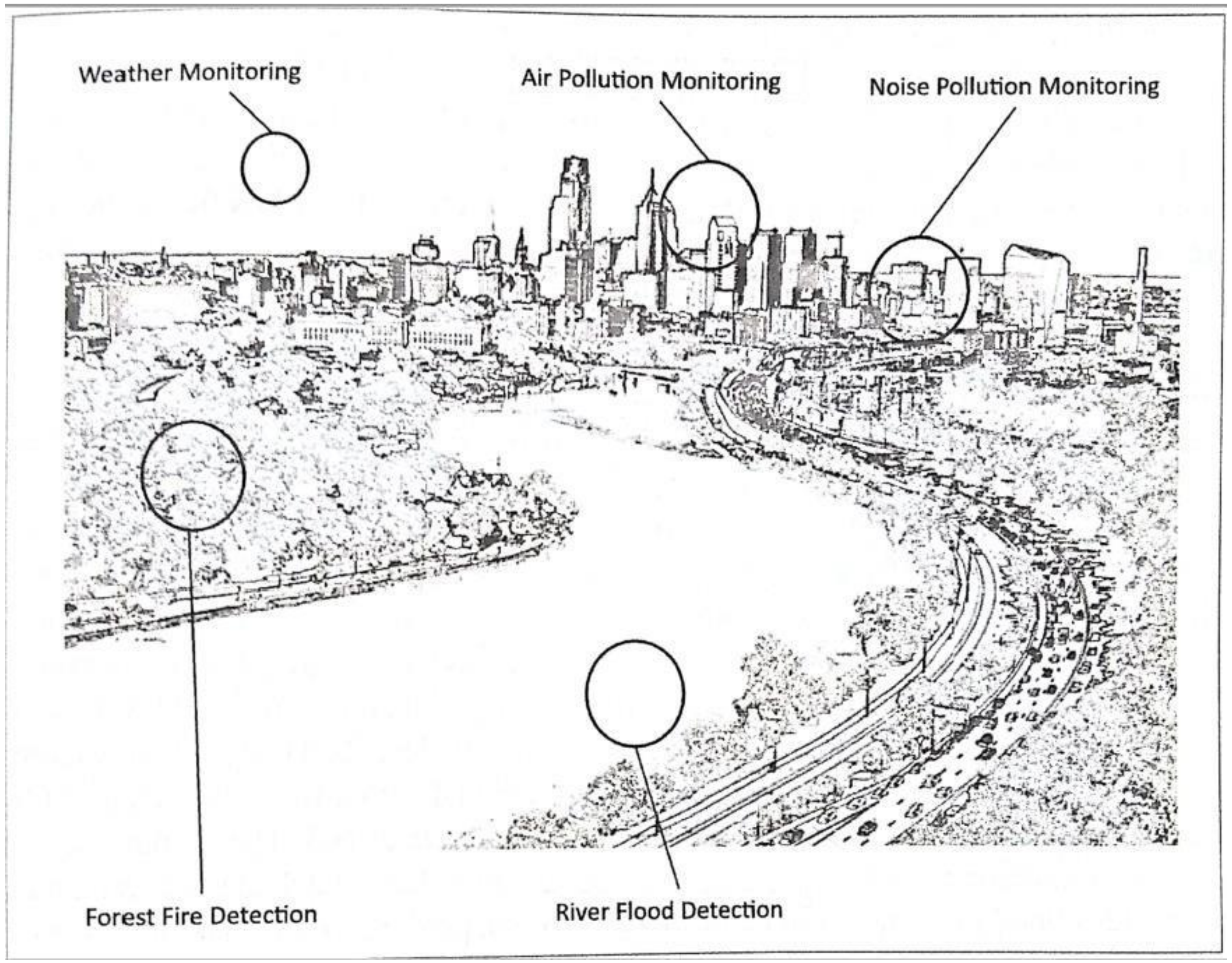
Cities (1/2)


Structural Health Monitoring
Surveillance
Emergency Services
(Fire, Gas Leak, Water Leakage detection)
Smart Lighting
Smart Roads
Smart Parking

Cities (2/2)

- Smart Parking
  - Detect the number of empty parking slots
  - Send the information over the internet and accessed by smartphones
- Smart Roads
  - Provide information on driving conditions, traffic congestions, accidents
  - Alert for poor driving conditions
- Structural Health Monitoring
  - Monitor the vibration levels in the structures (bridges and buildings)
  - Advance warning for imminent failure of the structure
- Surveillance
  - Use the large number of distributed and internet connected video surveillance cameras
  - Aggregate the video in cloud-based scalable storage solutions
- Emergency Response
  - Used for critical infrastructure monitoring
  - Detect adverse events

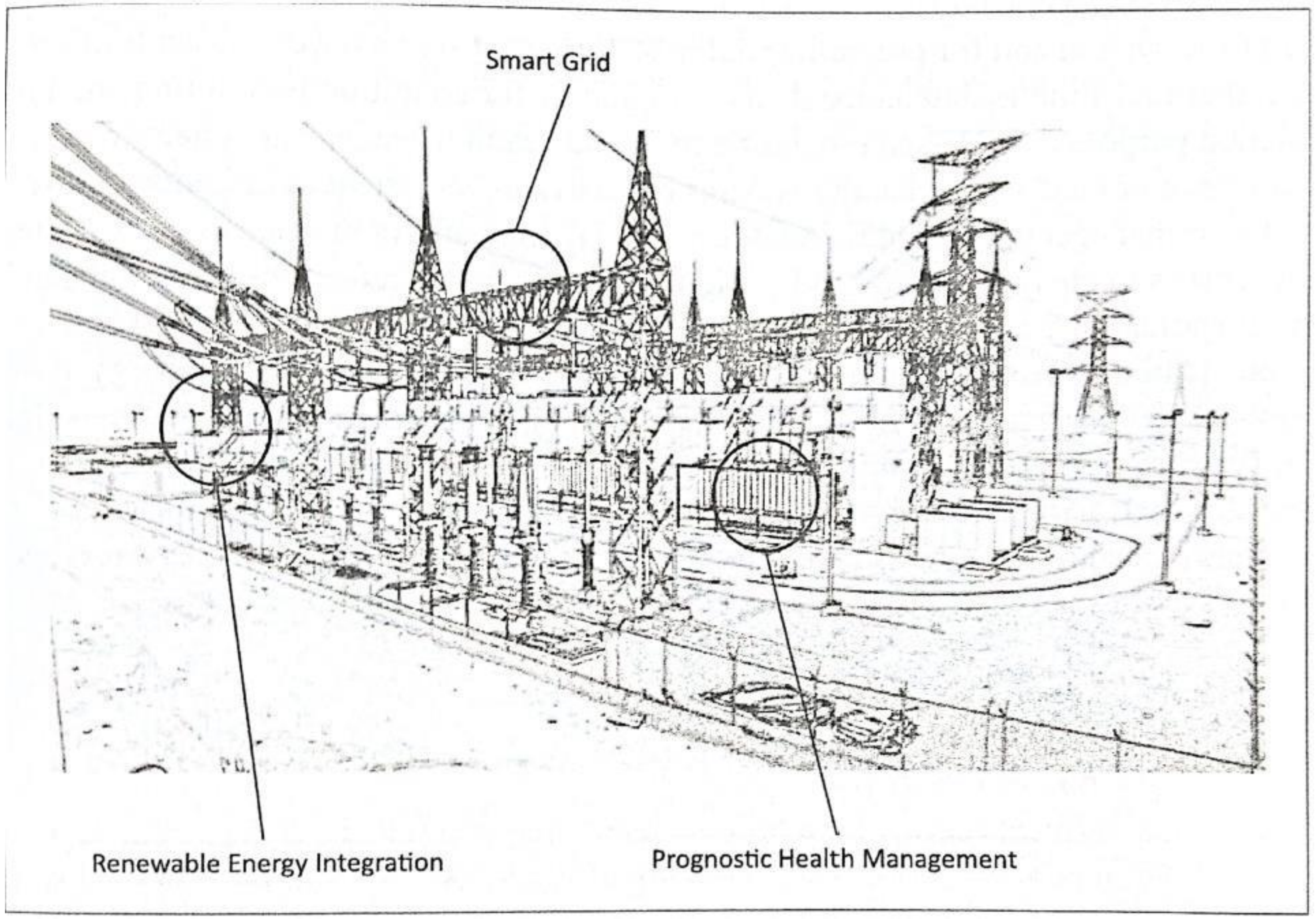
Environment (1/2)



Environment (2/2)

- Weather Monitoring
  - Collect data from several sensors (temperature, humidity, pressure, etc.)
  - Send the data to cloud-based applications and storage back-ends
- Air Pollution Monitoring
  - Monitor emission of harmful gases (CO_2, CO, NO, NO_2, etc.)
  - Factories and automobiles use gaseous and meteorological sensors
  - Integration with a single-chip microcontroller, several air pollution sensors, GPRS-modem, and a GPS module
- Noise Pollution Monitoring
  - Use a number of noise monitoring stations
  - Generate noise maps from data collected
- Forest Fire Detection
  - Use a number of monitoring nodes deployed at different locations in a forests
    - Use temperature, humidity, light levels, etc.
  - Provide early warning of potential forest fire
  - Estimates the scale and intensity
- River Floods Detection
  - Monitoring the water level (using ultrasonic sensors) and flow rate (using the flow velocity sensors)
  - Raise alerts when rapid increase in water level and flow rate is detected

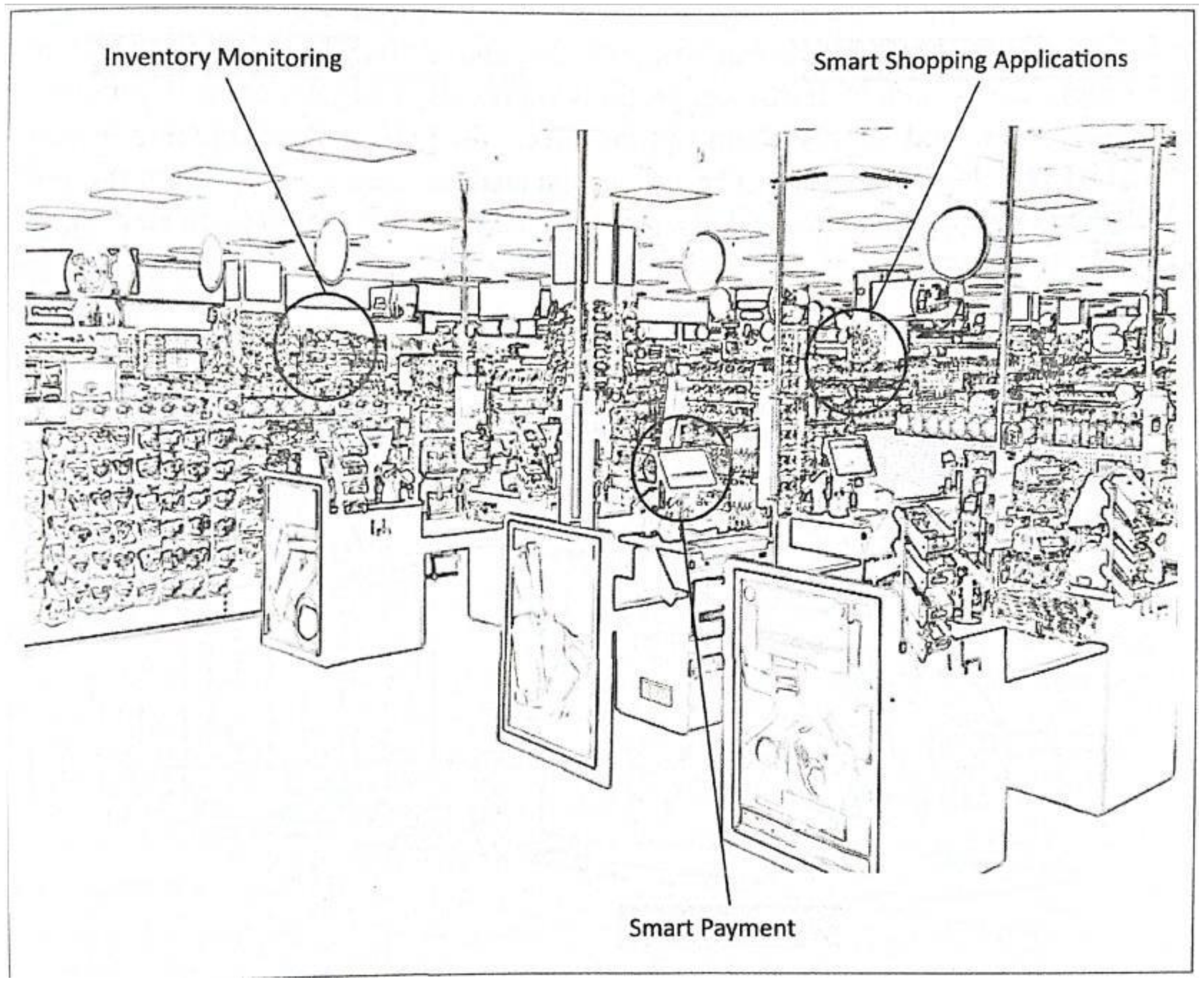
Energy (1/2)


Smart Grid
Renewable Energy Integration
Prognostic Health Management

Energy (2/2)

- Smart Grids
  - Collect data regarding electricity generation, consumption, storage (conversion of energy into other forms), distribution, equipment health data
  - Control the consumption of electricity
  - Remotely switch off supply
- Renewable Energy Systems
  - Measure the electrical variables
  - Measure how much the power is fed into the grid
- Prognostics
  - Predict performance of machines or energy systems
    - By collect and analyze the data from sensors

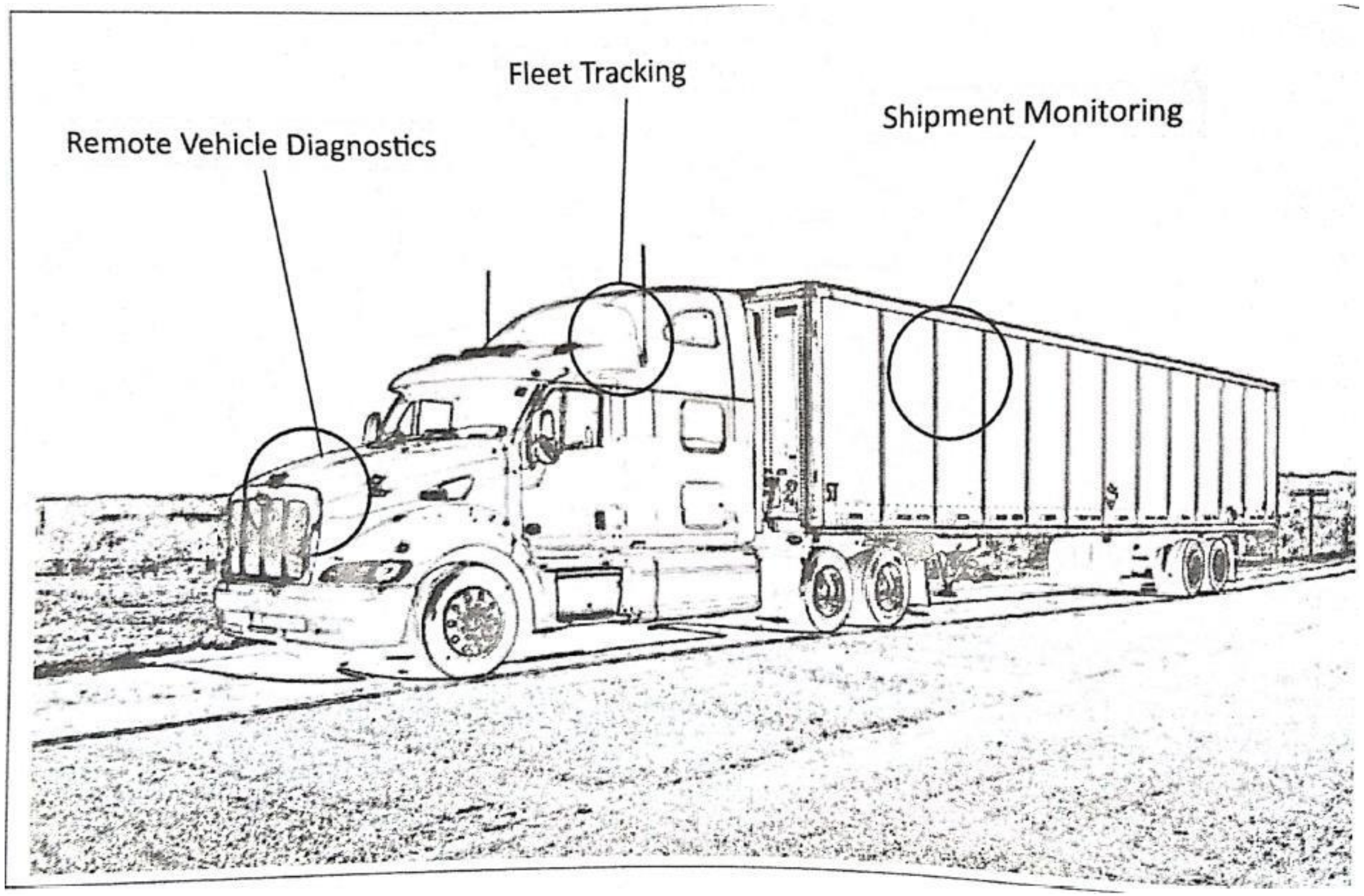
Retail (1/2)


Inventory Monitoring
Smart Shopping Applications
Smart Payment

Retail (2/2)

- Inventory Management
  - Monitoring the inventory by the RFID readers
  - Tracking the products
- Smart Payments
  - Use the NFC
    - Customers store the credit card information in their NFC-enabled
- Smart Vending Machines
  - Allow remote monitoring of inventory levels
  - Elastic pricing of products
  - Contact-less payment using NFC
  - Send the data to the cloud for predictive maintenance
    - The information of inventory levels
    - The information of the nearest machine in case a product goes out of stock in a machine

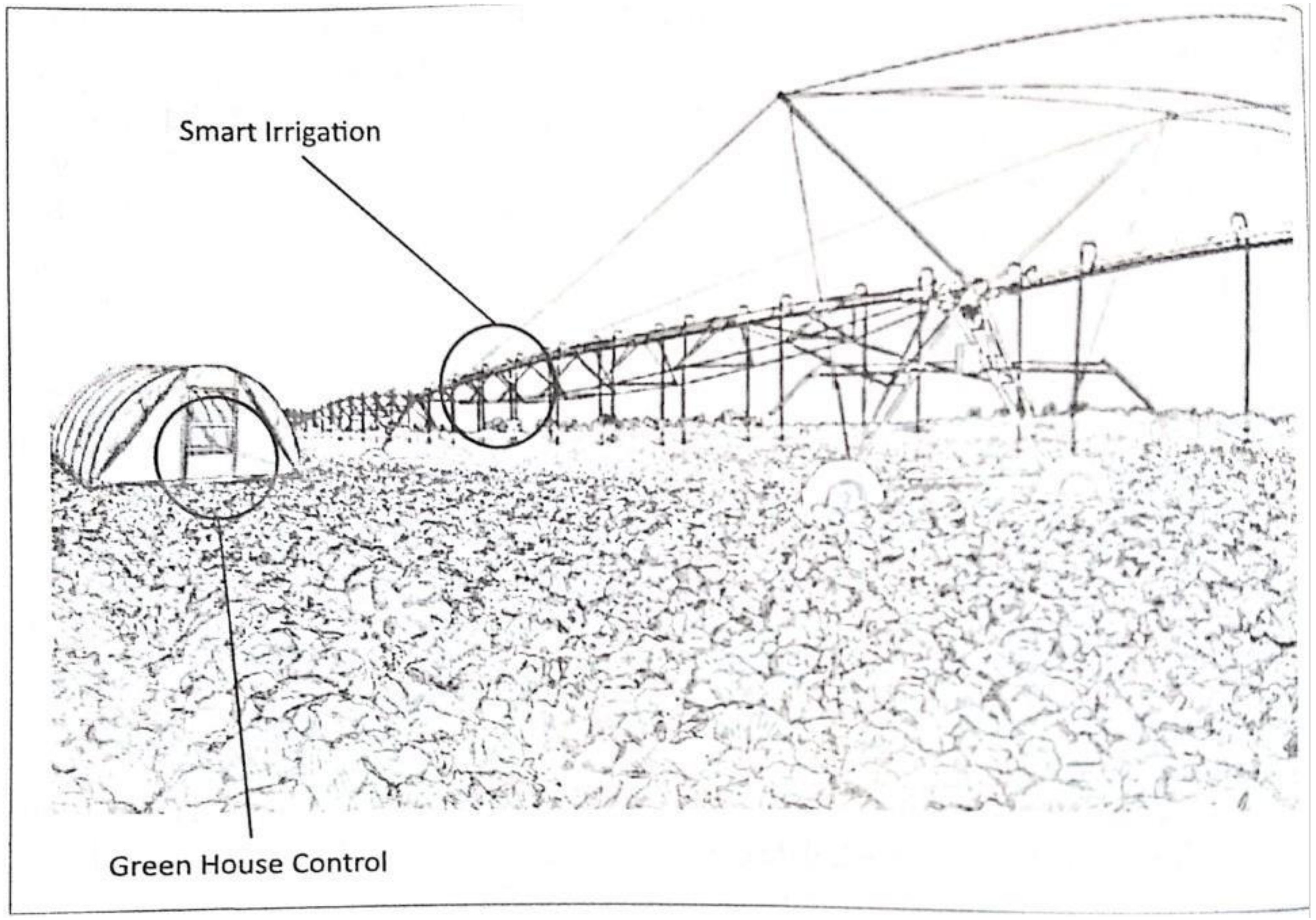
Logistics (1/2)



Logistics (2/2)

- Route Generation & Scheduling
  - Generate end-to-end routes using combination of route patterns
  - Provide route generation queries
  - Can be scale up to serve a large transportation network
- Fleet Tracking
  - Track the locations of the vehicles in real-time
  - Generate alerts for deviations in planned routes
- Shipment monitoring
  - Monitoring the conditions inside containers
  - Using sensors (temperature, pressure, humidity)
  - Detecting food spoilage
- Remote Vehicle Diagnostics
  - Detect faults in the vehicle
  - Warn of impending faults
  - IoT collects the data on vehicle (speed, engine RPM, coolant temperature)
  - Generate alerts and suggest remedial actions

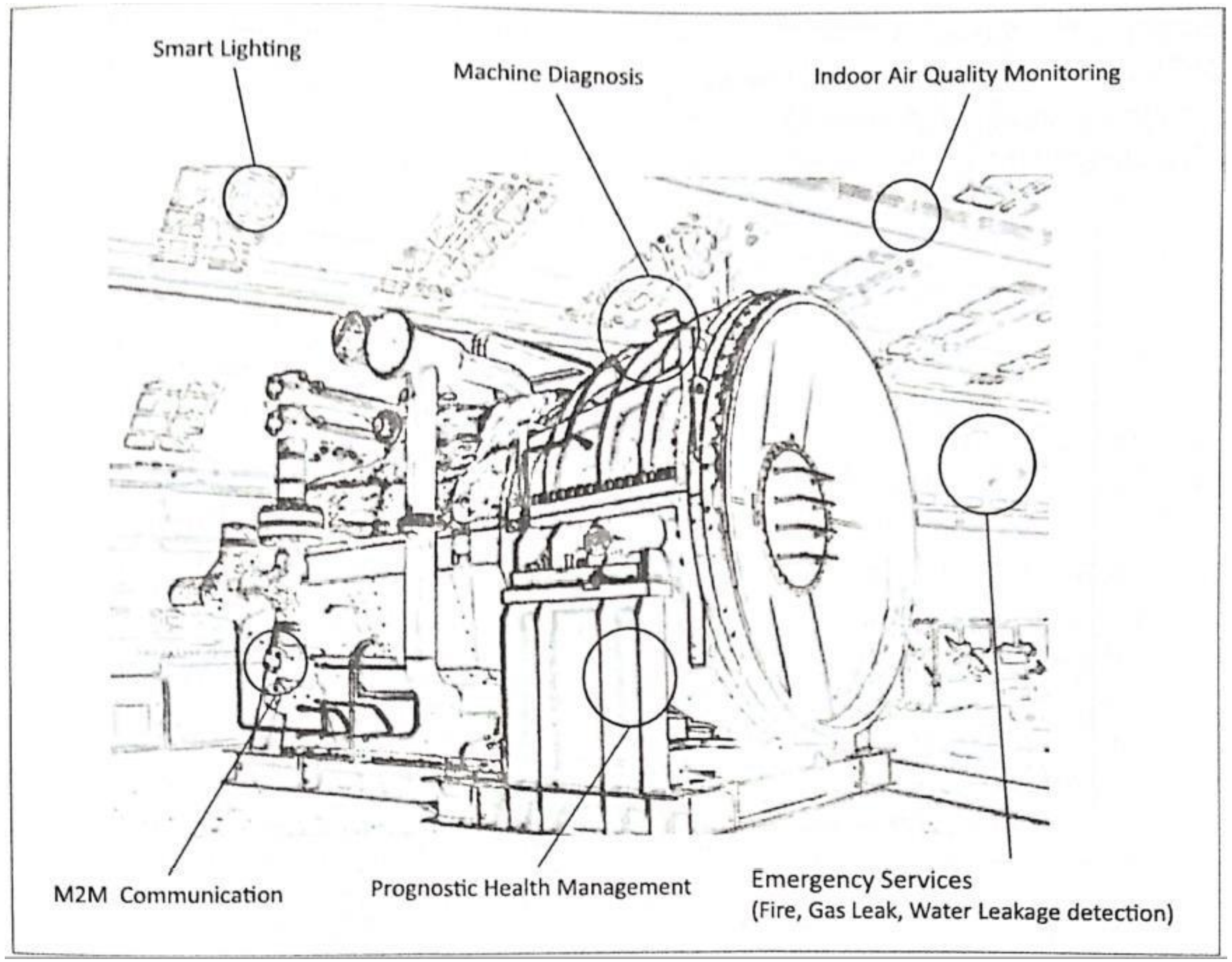
Agriculture (1/2)



Agriculture (2/2)

- Smart Irrigation
  - Use sensors to determine the amount of moisture in the soil
  - Release the flow of water
    - Using predefined moisture levels
  - Water Scheduling
- Green House Control
  - Automatically control the climatological conditions inside a green house
    - Using several sensors to monitor
    - Using actuation devices to control
      - Valves for releasing water and switches for controlling fans
  - Maintenance of agricultural production

Industry (1/2)


Smart Lighting
Machine Diagnosis
Indoor Air Quality Monitoring
M2M Communication
Prognostic Health Management
Emergency Services
(Fire, Gas Leak, Water Leakage detection)

Industry (2/2)

- Machine Diagnosis
  - Sensors in machine monitor the operating conditions
    - For example: temperature & vibration levels
  - Collecting and analyzing massive scale machine sensor data
    - For reliability analysis and fault prediction in machines
- Indoor Air Quality Monitoring
  - Use various gas sensors
    - To monitor the harmful and toxic gases (CO, NO, NO_2, etc.)
  - Measure the environmental parameters to determine the indoor air quality
    - Temperature, humidity, gaseous pollutants, aerosol

Health & Lifestyle

- Health & Fitness Monitoring
  - Collect the health-care data
    - Using some sensors: body temperature, heart rate, movement (with accelerometers), etc.
  - Various forms : belts and wrist-bands
- Wearable electronic
  - Assists the daily activities
    - Smart watch
    - Smart shoes
    - Smart wristbands

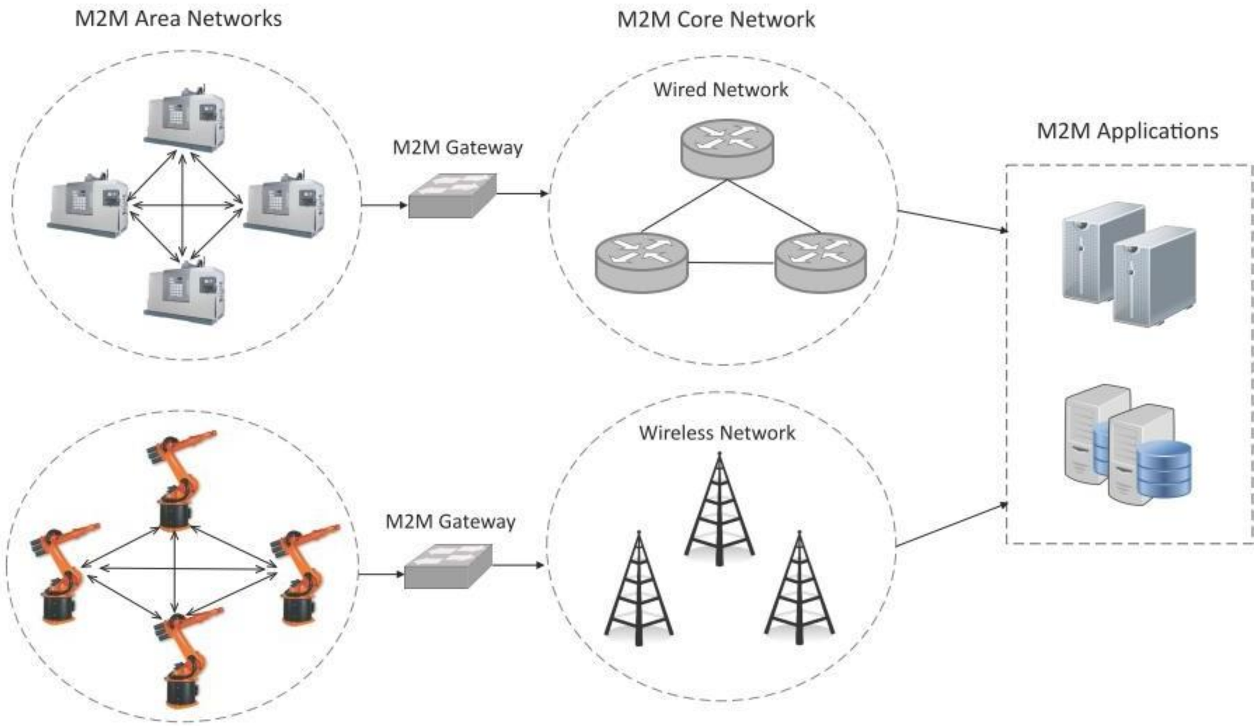
IoT & M2M

Outline

- M2M
- Differences and Similarities between M2M and IoT
- SDN and NFV for IoT

Machine-to-Machine (M2M)

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

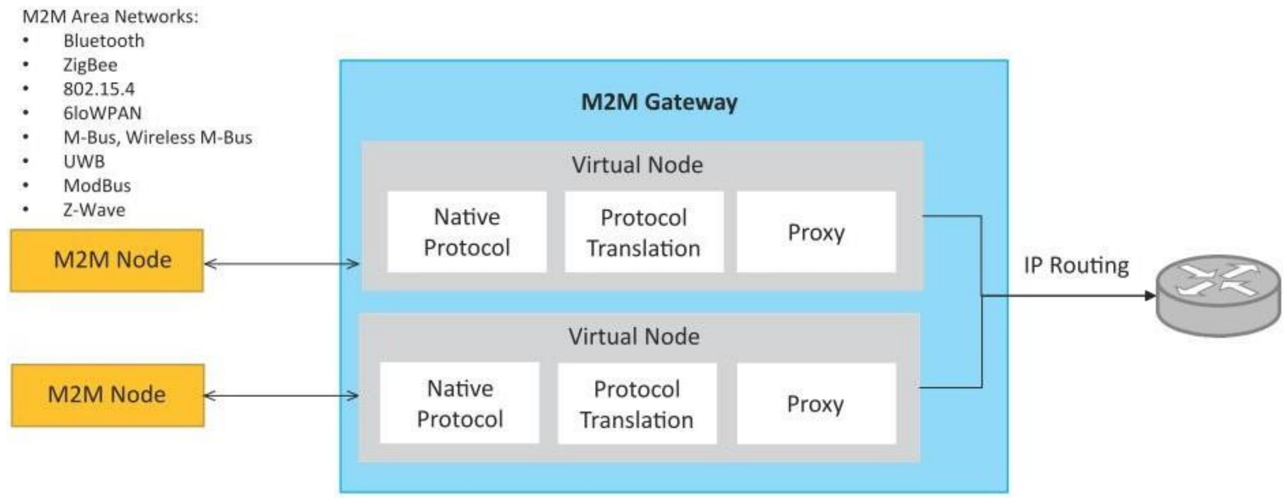


Machine-to-Machine (M2M)

- An M2M area network comprises of machines (or M2M nodes) which have embedded hardware modules for sensing, actuation and communication.
- Various communication protocols can be used for M2M local area networks such as ZigBee, Bluetooh, ModBus, M-Bus, Wirless M-Bus, PowerLine Communication (PLC), 6LoWPAN, IEEE 802.15.4, etc.
- The communication network provides connectivity to remote M2M area networks.
- The communication network can use either wired or wireless networks (IP-based).
- While the M2M area networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based networks.

M2M gateway

- Since non-IP based protocols are used within M2M area networks, the M2M nodes within one network cannot communicate with nodes in anexternal network.
- To enable the communication between remote M2M area networks, M2M gateways are used.



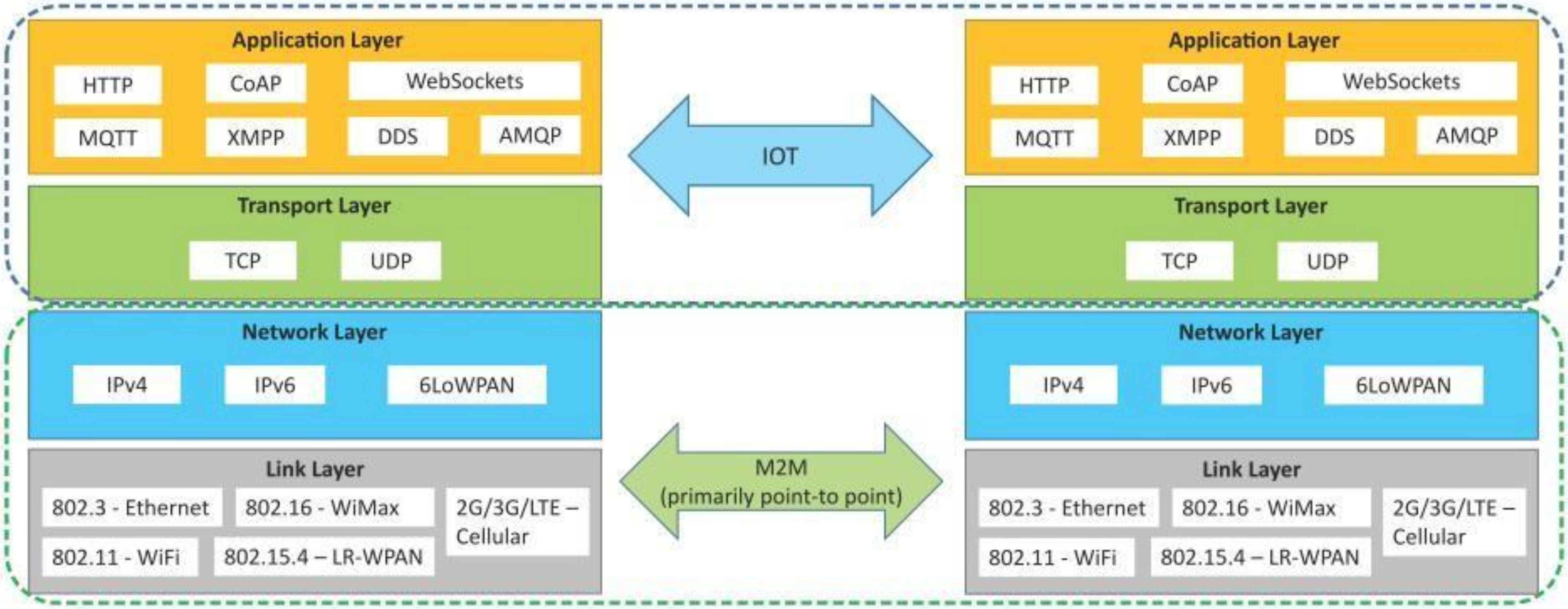
Difference between IoT and M2M

- Communication Protocols
  - M2M and IoT can differ in how the communication between the machines ordevices happens.
  - M2M uses either proprietary or non-IP based communication protocols for communication within the M2M area networks.
- Machines in M2M vs Things in IoT
  - The "Things" in IoT refers to physical objects that have unique identifiers andcan sense and communicate with their external environment (and user applications) or their internal physical states.
  - M2M systems, in contrast to IoT, typically have homogeneous machine typeswithin an M2M area network.

Difference between IoT and M2M

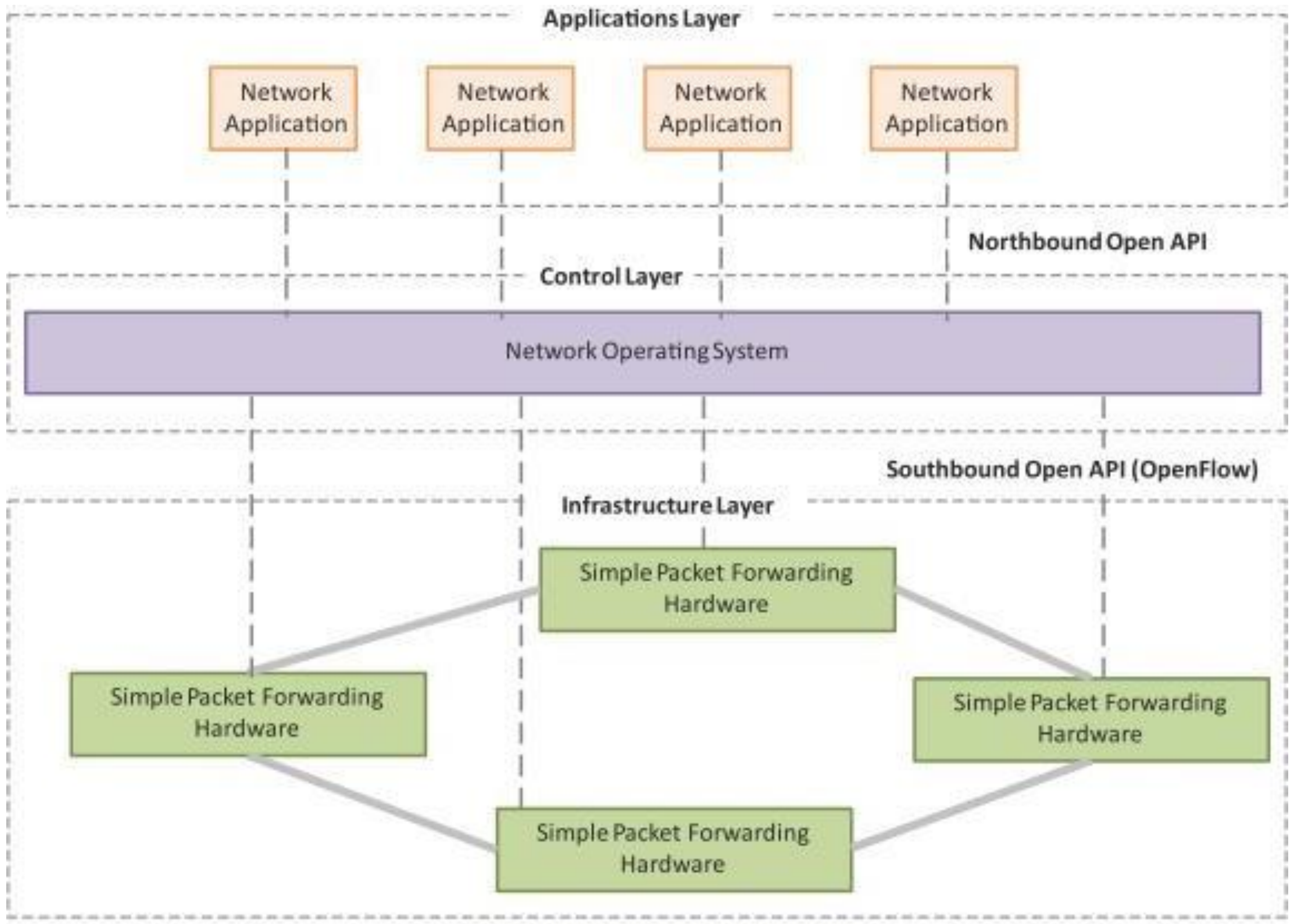
- Hardware vs Software Emphasis
  - While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.
- Data Collection & Analysis
  - M2M data is collected in point solutions and often in on-premises storage infrastructure.
  - In contrast to M2M, the data in IoT is collected in the cloud (can be public, private or hybrid cloud).
- 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-premisis 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.

Communication in IoT vs M2M



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 and make confi guration, management and provisioning simpler.
- The underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional networks.

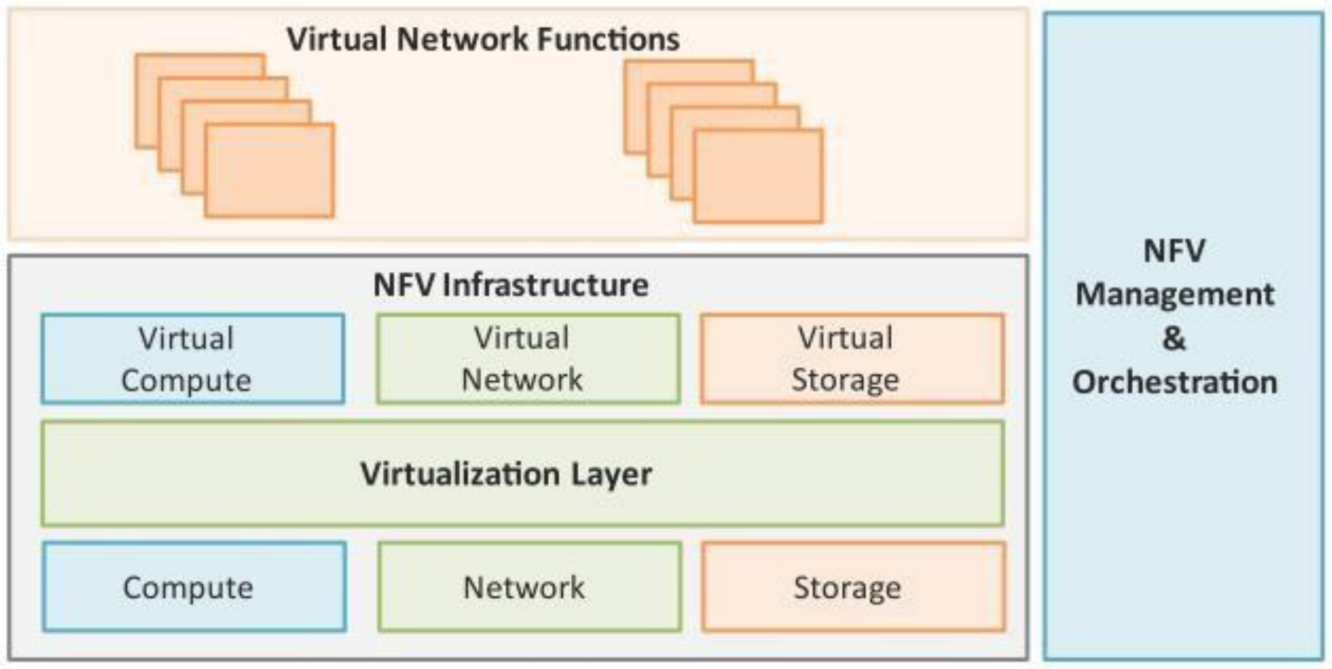


Key elements of SDN

- Centralized Network Controller
  - With decoupled control and data planes and centralized network controller, the network administrators can rapidly configure the network.
- Programmable Open APIs
  - SDN architecture supports programmable open APIs for interface betweenthe SDN application and control layers (Northbound interface).
- Standard Communication Interface (OpenFlow)
  - SDN architecture uses a standard communication interface between the control and infrastructure layers (Southbound interface).
  - OpenFlow, which is defined by the Open Networking Foundation (ONF) isthe broadly accepted SDN protocol for the Southbound interface.

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.

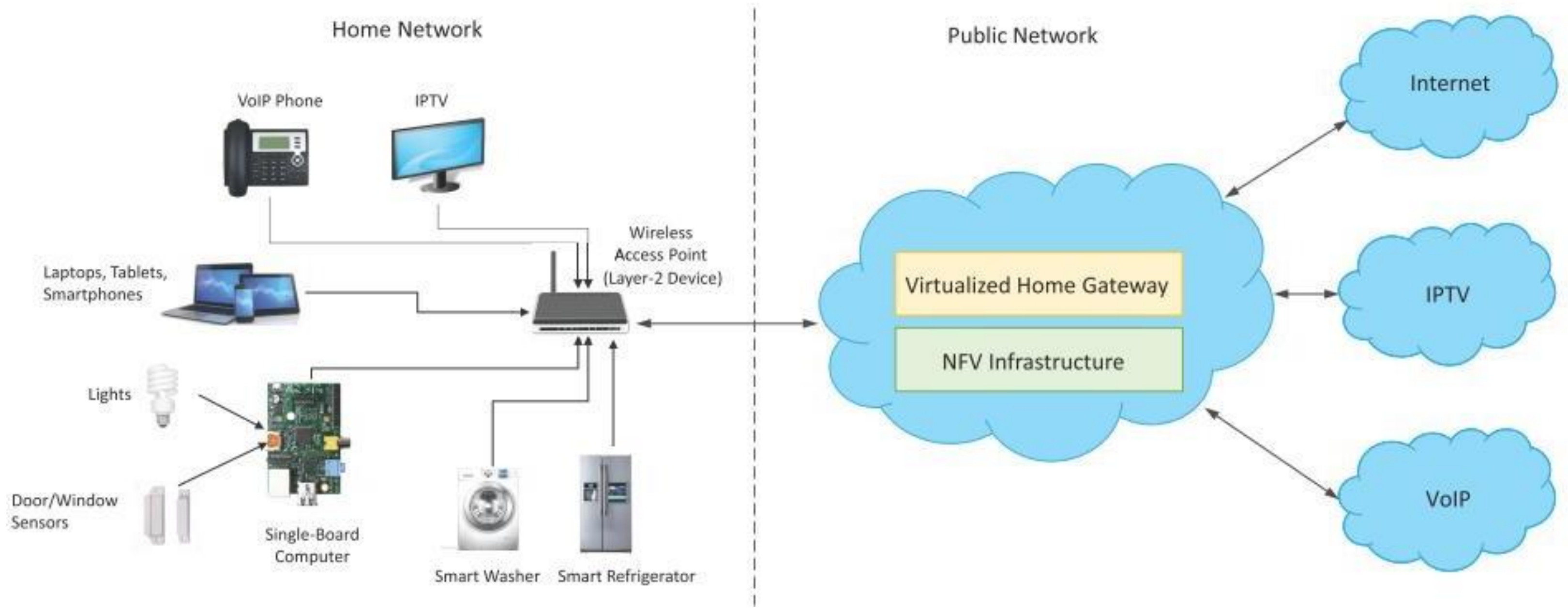


Key elements of NFV

- Virtualized Network Function (VNF):
  - VNF is a software implementation of a network function which is capable of running over the NFV Infrastructure (NFVI).
- NFV Infrastructure (NFVI):
  - NFVI includes compute, network and storage resources that are virtualized.
- 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.

NFV Use Case

- NFV can be used to virtualize the Home Gateway. The NFV infrastructure in the cloud hosts a virtualized Home Gateway. The virtualized gateway provides private IP addresses to the devices in the home. The virtualized gateway also connects to network services such as VoIP and IPTV.



IoT System Management with NETCONF-YANG

Outline

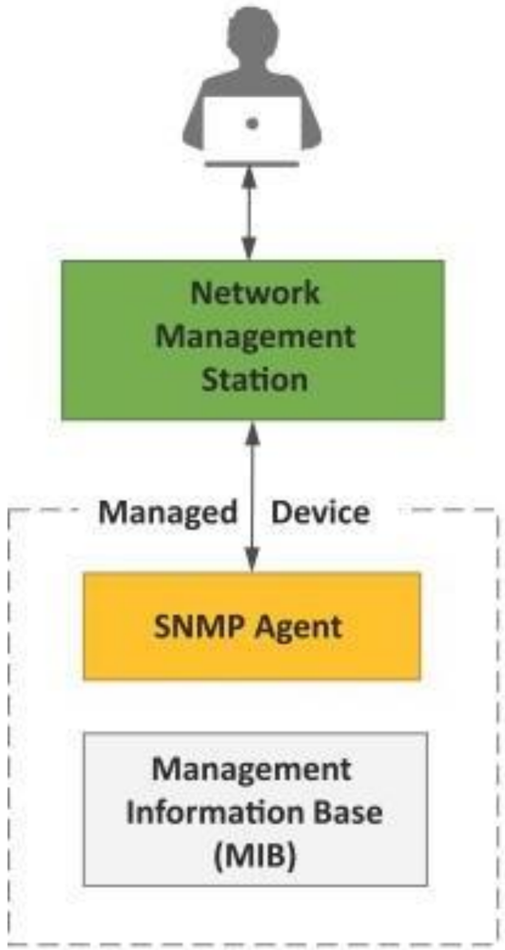
- Need for IoT Systems Management
- SNMP
- Network Operator Requirements
- NETCONF
- YANG
- IoT Systems Management with NETCONF-YANG

Need for IoT Systems Management

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

Simple Network Management Protocol (SNMP)

- SNMP is a well-known and widely used network management protocol that allows monitoring and configuring network devices such as routers, switches, servers, printers, etc.
- SNMP component include
  - Network Management Station (NMS)
  - Managed Device
  - Management Information Base (MIB)
  - SNMP Agent that runs on the device



Limitations of SNMP

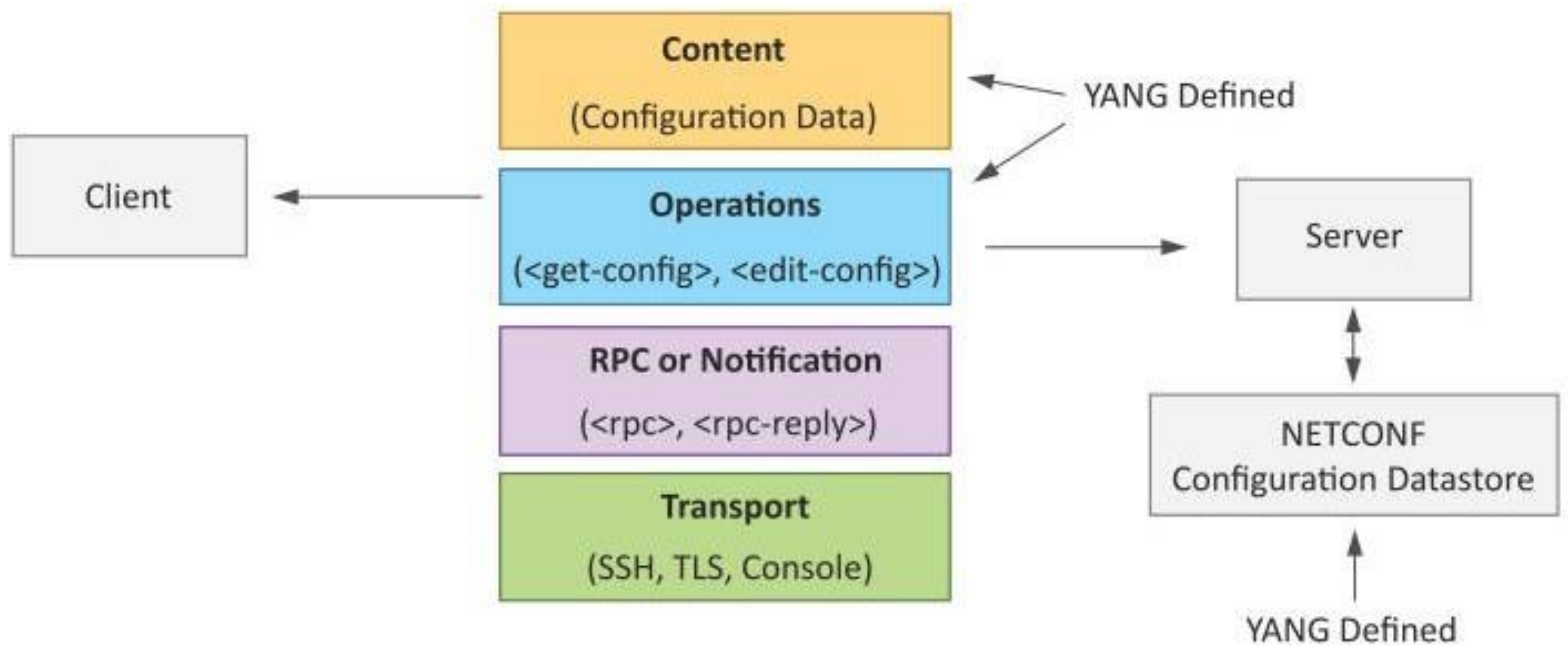
- SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device.
- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.
- MIBs often lack writable objects without which device configuration is not possible using SNMP.
- It is difficult to differentiate between configuration and state data in MIBs.
- Retrieving the current configuration from a device can be difficult with SNMP.
- Earlier versions of SNMP did not have strong security features.

Network Operator Requirements

- Ease of use
- Distinction between configuration and state data
- Fetch configuration and state data separately
- Configuration of the network as a whole
- Configuration transactions across devices
- Configuration deltas
- Dump and restore configurations
- Configuration validation
- Configuration database schemas
- Comparing configurations
- Role-based access control
- Consistency of access control lists:
- Multiple configuration sets
- Support for both data-oriented and task-oriented access control

NETCONF

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



NETCONF

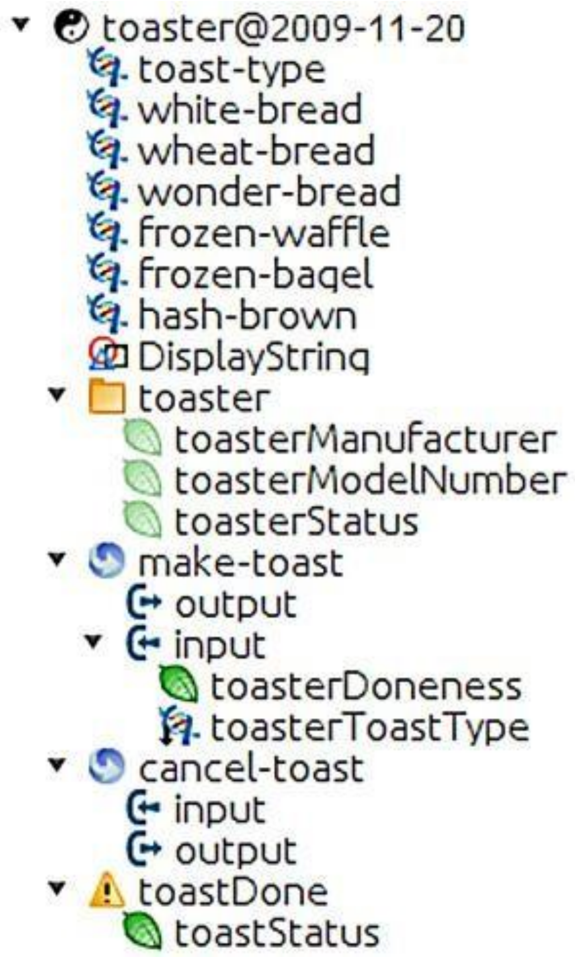
- NETCONF works on SSH transport protocol.
- Transport layer provides end-to-end connectivity and ensure reliable delivery of messages.
- NETCONF uses XML-encoded Remote Procedure Calls (RPCs) for framing request and response messages.
- The RPC layer provides mechanism for encoding of RPC calls and notifications.
- NETCONF provides various operations to retrieve and edit configuration data from network devices.
- The Content Layer consists of configuration and state data which is XML-encoded.
- The schema of the configuration and state data is defined in a data modeling language called YANG.
- NETCONF provides a clear separation of the configuration and state data.
- The configuration data resides within a NETCONF configuration datastore on the server.

YANG

- YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol
- YANG modules contain the definitions of the configuration data, state data, RPC calls that can be issued and the format of the notifications.
- YANG modules defines the data exchanged between the NETCONF client and server.
- A module comprises of a number of 'leaf' nodes which are organized into a hierarchical tree structure.
- The 'leaf' nodes are specified using the 'leaf' or 'leaf-list' constructs.
- Leaf nodes are organized using 'container' or 'list' constructs.
- A YANG module can import definitions from other modules.
- Constraints can be defined on the data nodes, e.g. allowed values.
- YANG can model both configuration data and state data using the 'config' statement.

YANG Module Example

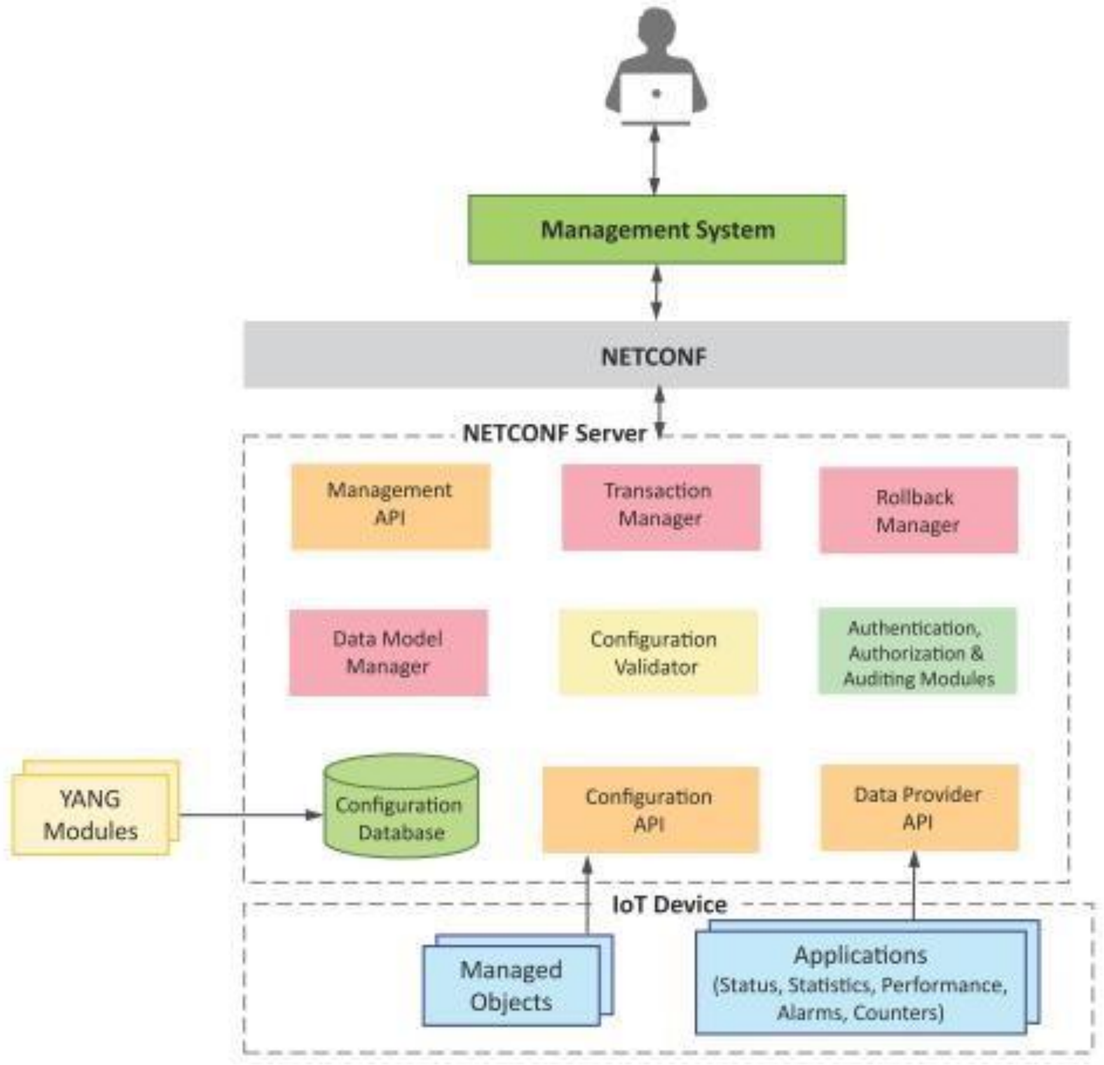
- This YANG module is a YANG version of the toasterMIB
- The toaster YANG module begins with the header information followed by identity declarations which define various bread types.
- The leaf nodes (‘toasterManufacturer’, ‘toasterModelNumber’ and oasterStatus’) are defined in the ‘toaster’ container.
- Each leaf node definition has a type and optionallya description and default value.
- The module has two RPC definitions (‘make-toast’ and ‘cancel-toast’).



```
▾ toaster@2009-11-20
    toast-type
    white-bread
    wheat-bread
    wonder-bread
    frozen-waffle
    frozen-bagel
    hash-brown
    DisplayString
  ▾ toaster
      toasterManufacturer
      toasterModelNumber
      toasterStatus
  ▾ make-toast
      output
    ▾ input
        toasterDoneness
        toasterToastType
  ▾ cancel-toast
      input
      output
  ▾ toastDone
      toastStatus
```

IoT Systems Management with NETCONF-YANG

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

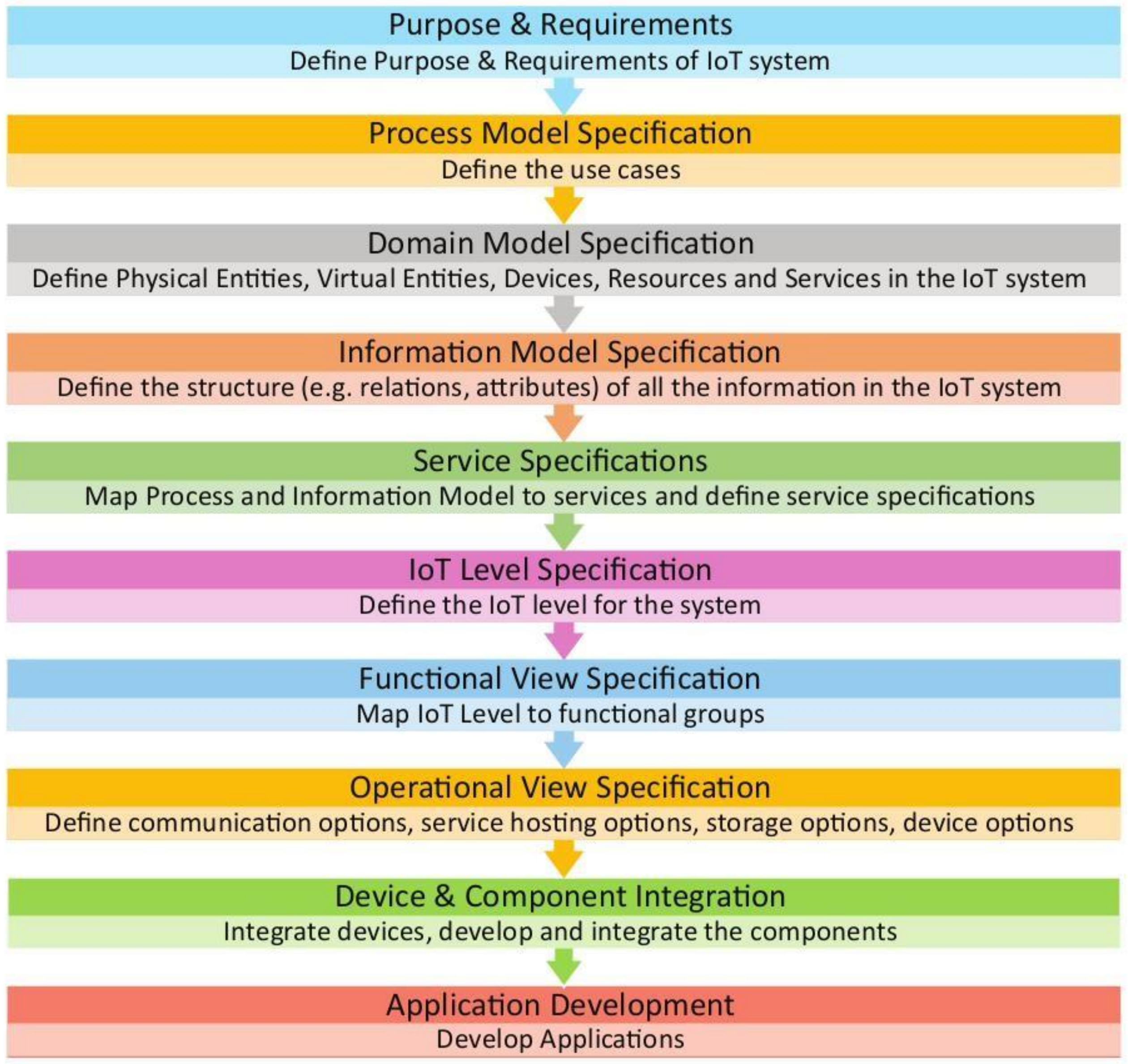


IoT Design Methodology

Outline

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

IoT Design Methodology - Steps


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

Step 1: Purpose & Requirements Specification

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

Step 2: Process Specification

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

Step 3: Domain Model Specification

- The third step in the IoT design methodology is to define the DomainModel.
- The domain model describes the main concepts, entities and objects in the domain of IoT system to be designed.
- Domain model defines the attributes of the objects and relationships between objects.
- Domain model provides an abstract representation of the concepts, objects and entities in the IoT domain, independent of anyspecific technology or platform.
- With the domain model, the IoT system designers can get an understanding of the IoT domain for which the system is to be designed.

Step 4: Information Model Specification

- The fourth step in the IoT design methodology is to define the Information Model.
- Information Model defines the structure of all the information in the IoT system, for example, attributes of Virtual Entities, relations, etc.
- Information model does not describe the specifics of how the information is represented or stored.
- To define the information model, we first list the Virtual Entities defined in the Domain Model. Information model adds more details to the Virtual Entities by defining their attributes and relations.

Step 5: Service Specifications

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

Step 6: IoT Level Specification

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

Step 7: Functional View Specification

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

Step 8: Operational View Specification

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

Step 9: Device & Component Integration

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

Step 10: Application Development

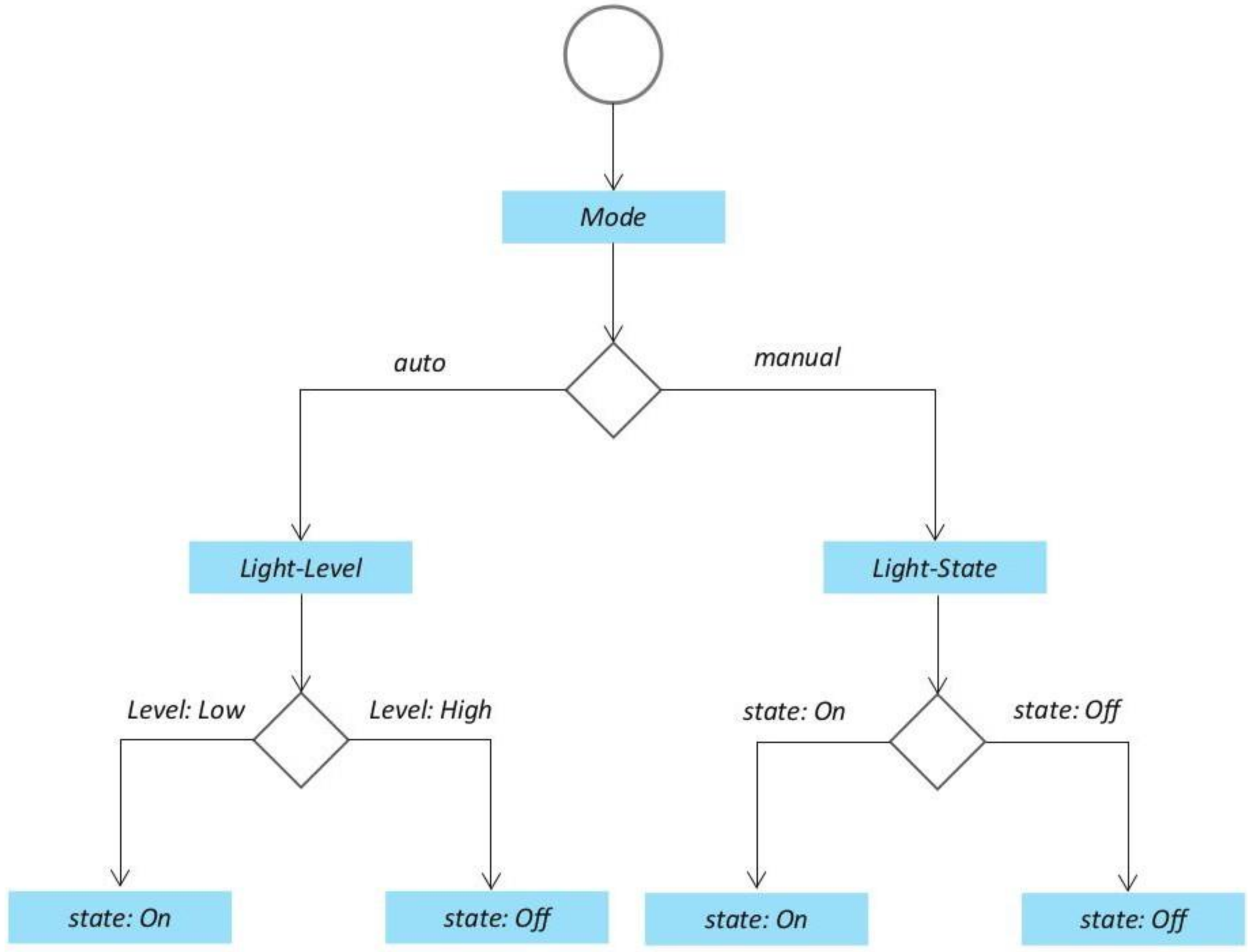
- The final step in the IoT design methodology is to develop the IoT application.

Home Automation Case Study

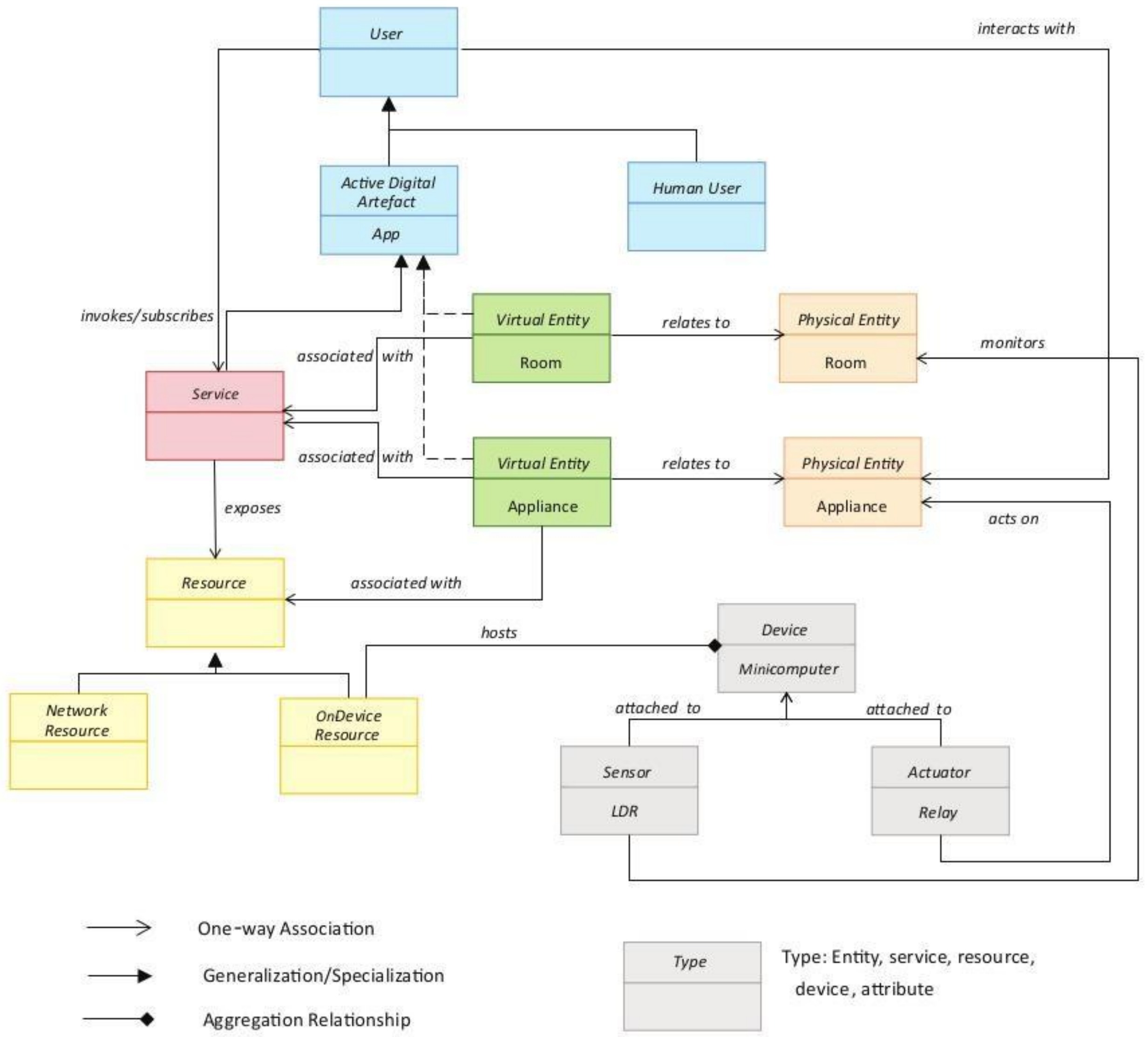
Step:1 - Purpose & Requirements

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

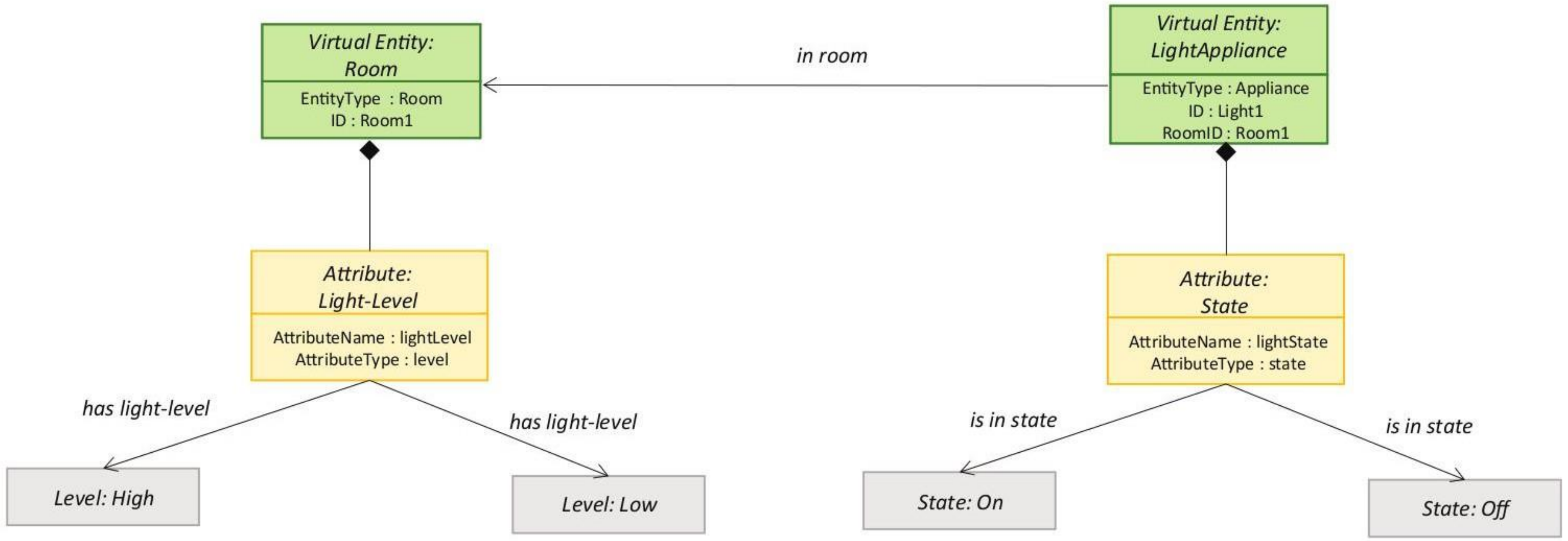
Step:2 - Process Specification



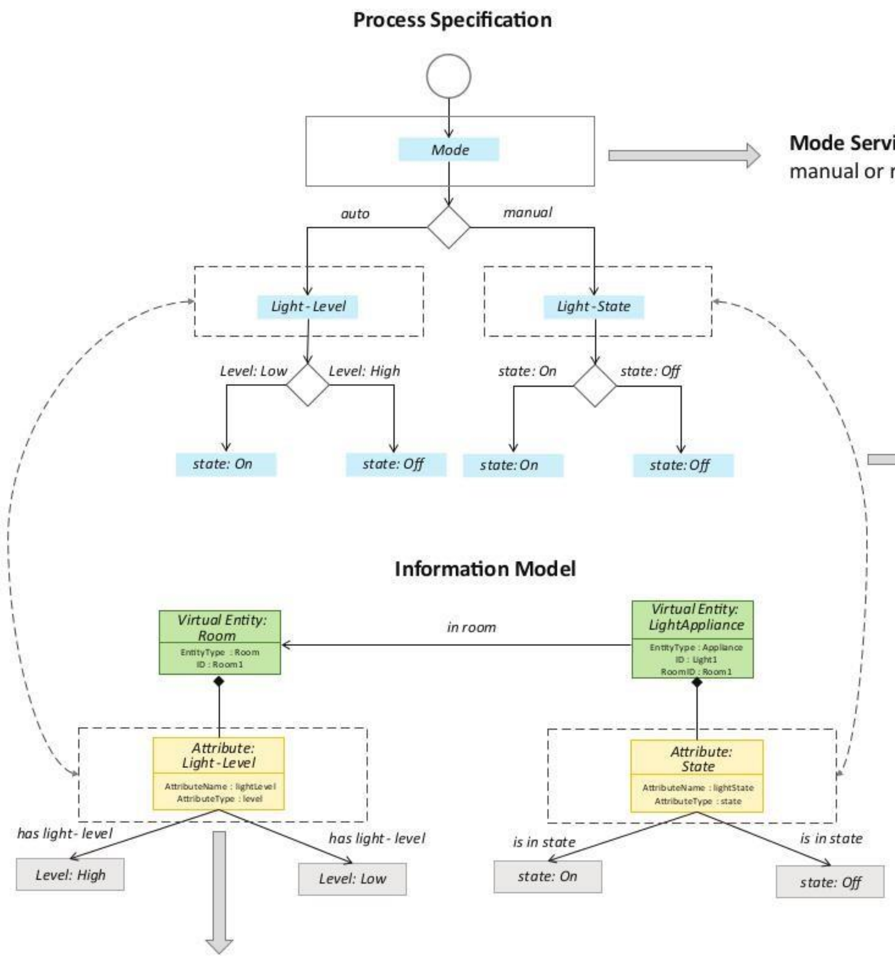
Step 3: Domain Model Specification



Step 4: Information Model Specification



Step 5: Service Specifications

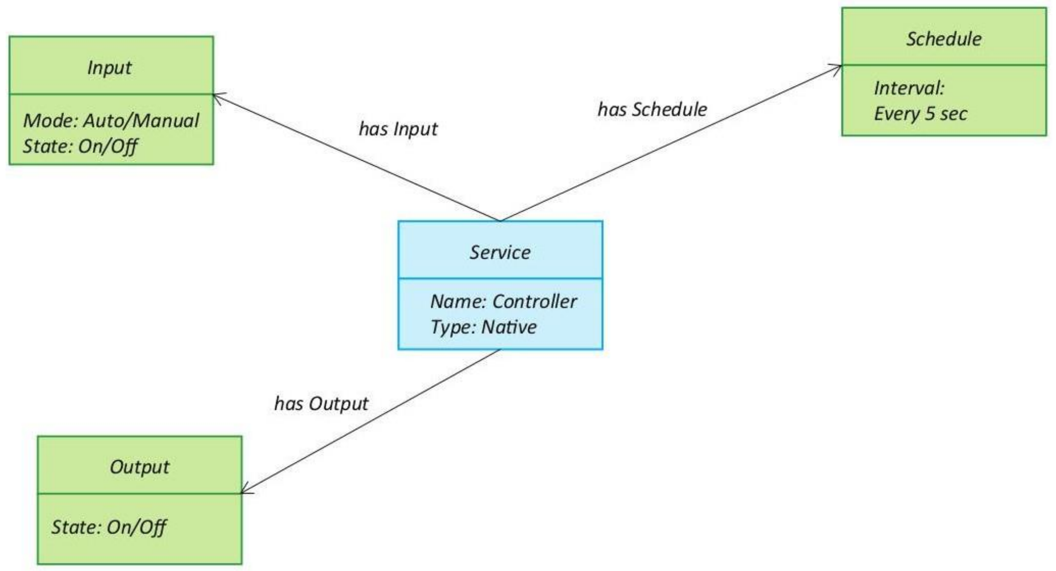


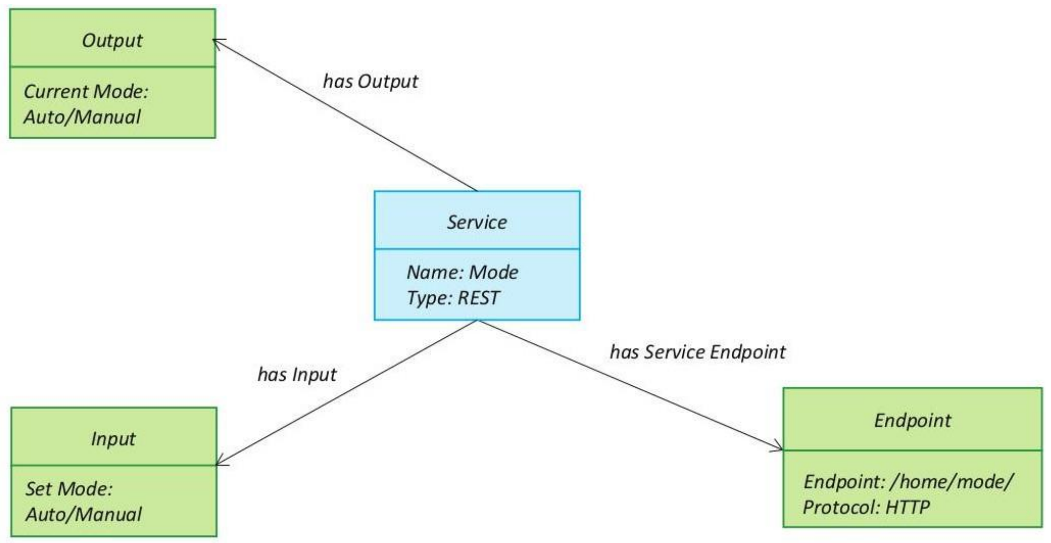
Mode Service: Sets mode to auto or manual or retrieves the current mode

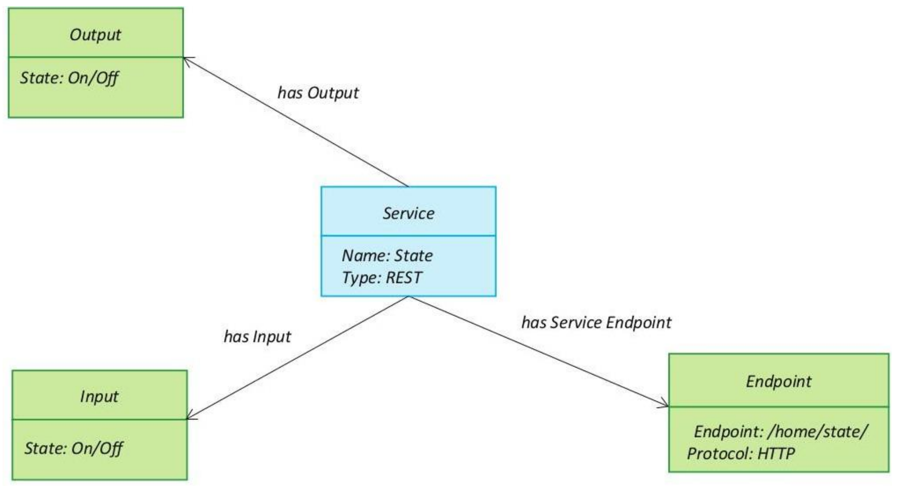
State Service: Sets the light appliance state to on/off or retrieves the current light state

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

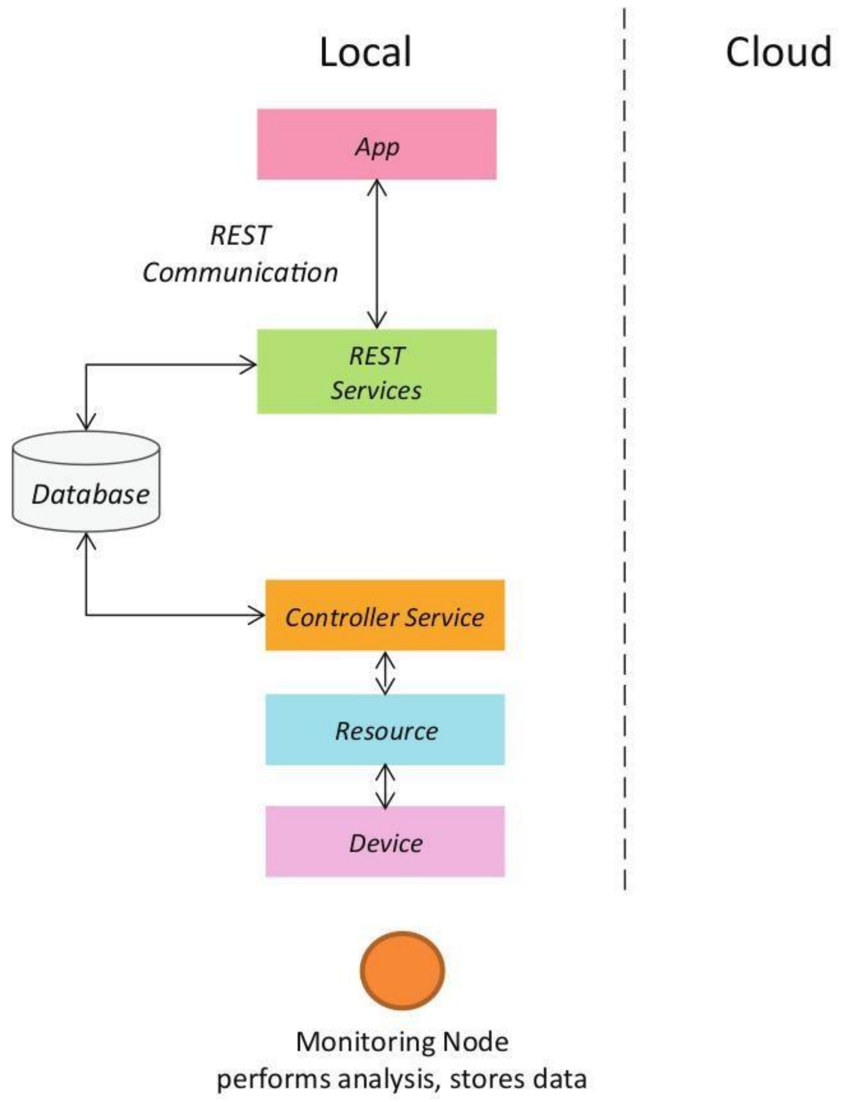
Step 5: Service Specifications



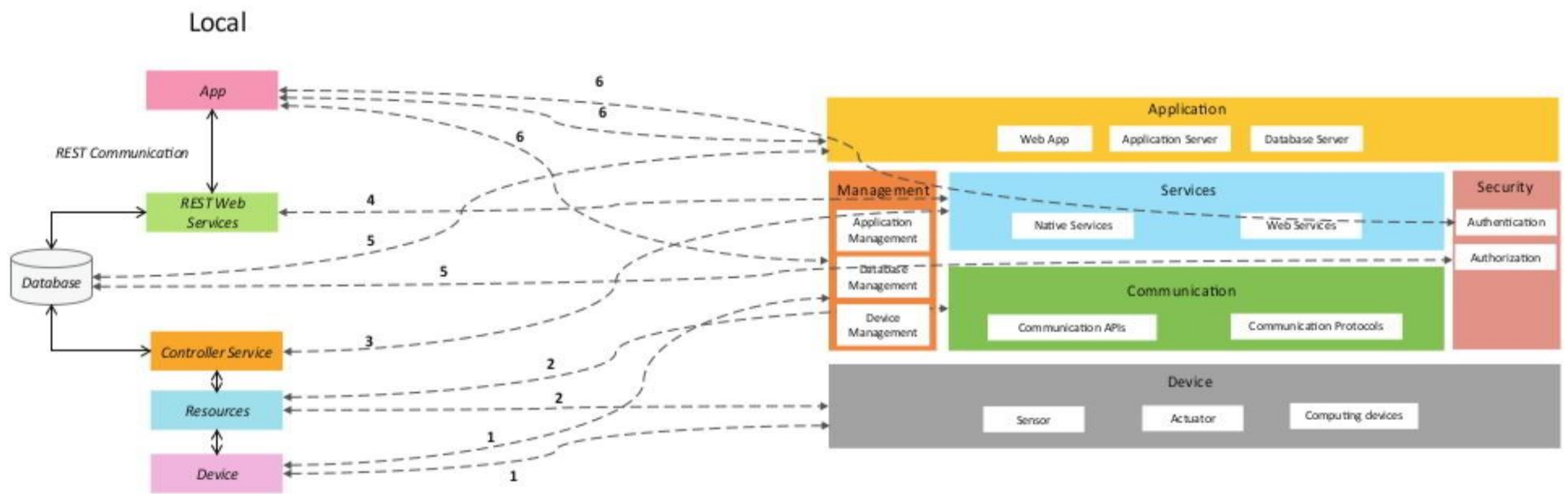




Step 6: IoT Level Specification



Step 7: Functional View Specification



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

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

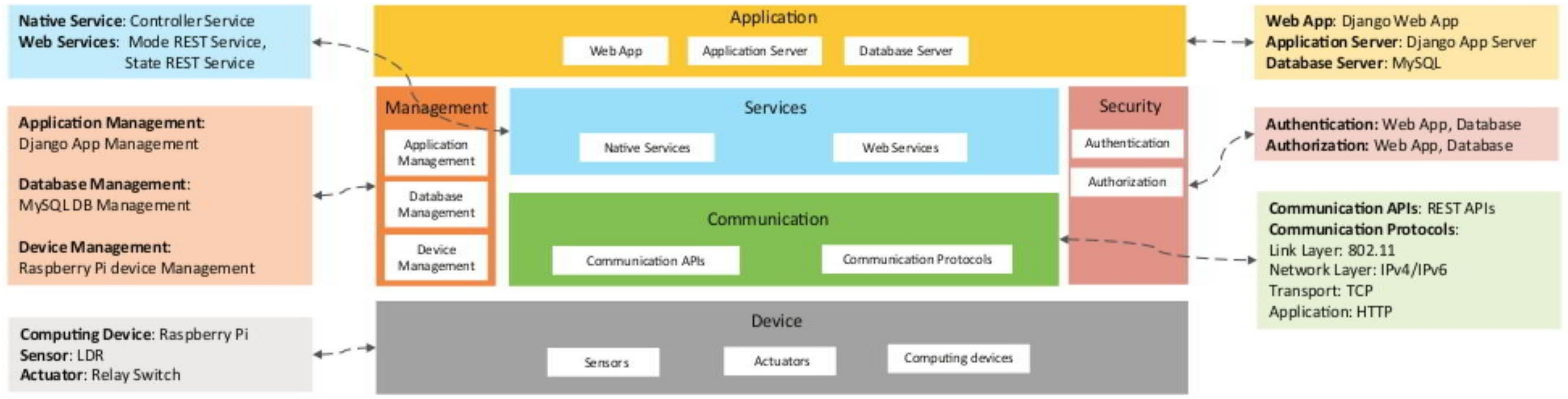
3. Controller service maps to the Services FG (native service). Web Services map to Services FG (web services)

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

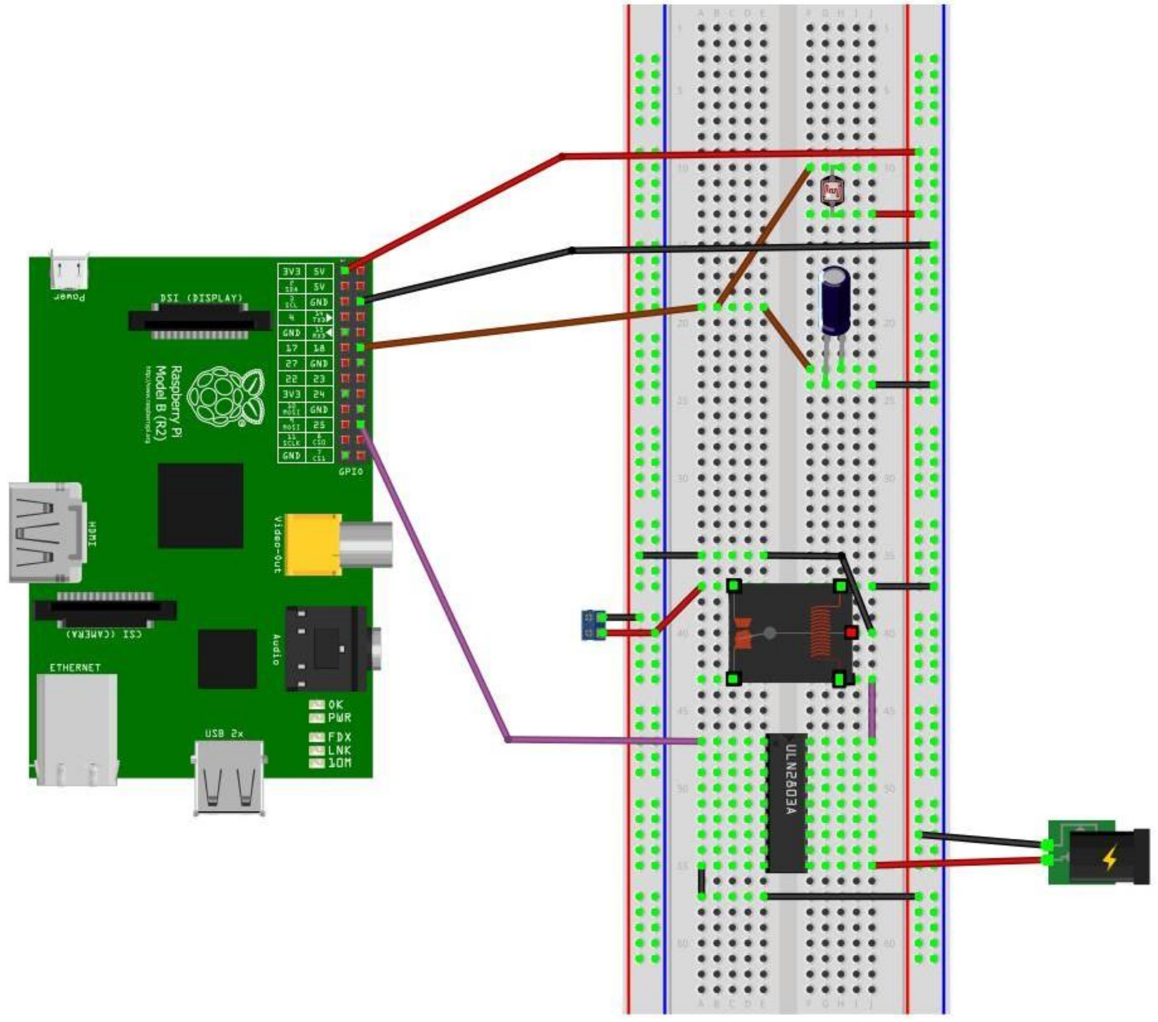
5. Database maps to the Management FG (database management) and Security FG (database security)

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

Step 8: Operational View Specification

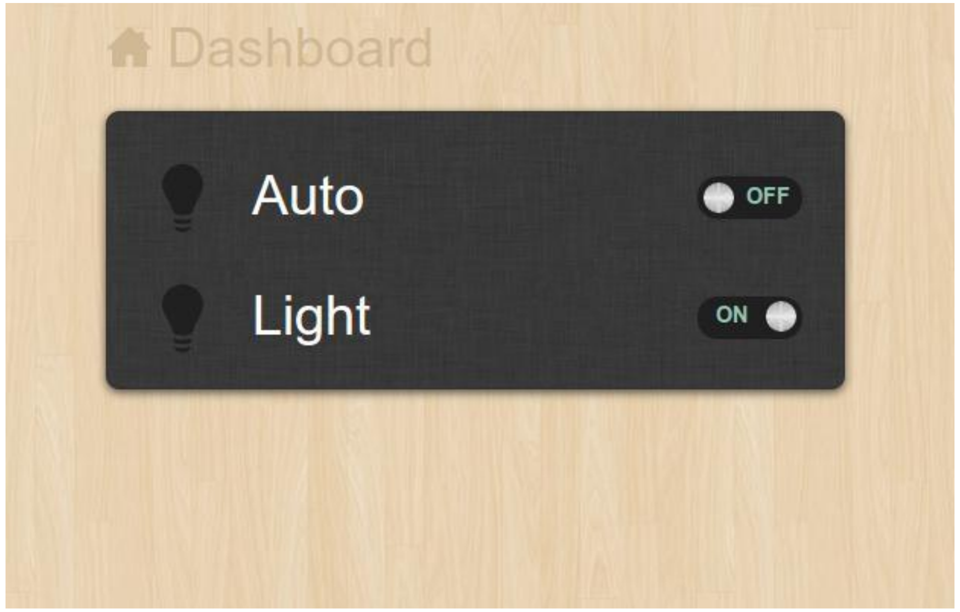


Step 9: Device & Component Integration



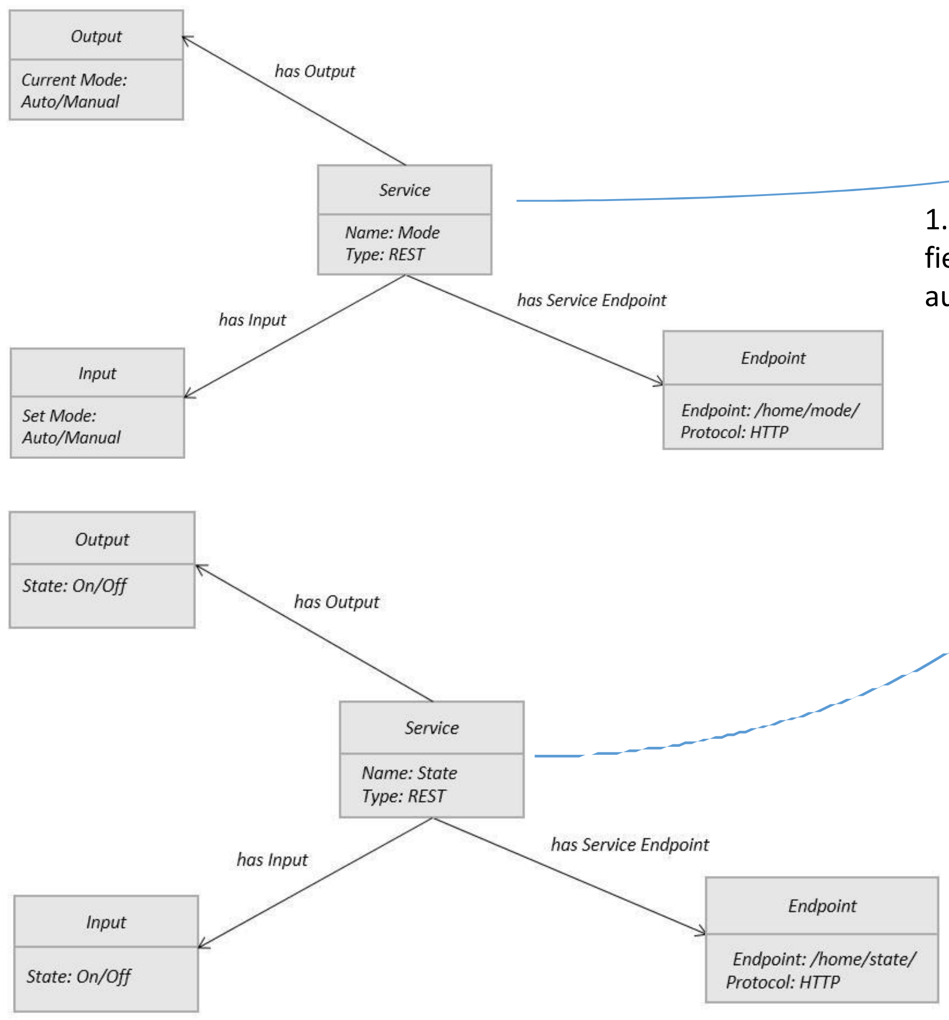
Step 10: Application Development

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



Implementation: RESTful Web Services

REST services implemented with Django REST Framework



1. Map services to models. Model fields store the states (on/off, auto/manual)

```
# Models – models.py
from django.db import models

class Mode(models.Model):
        name = models.CharField(max_length=50)

class State(models.Model):
        name = models.CharField(max_length=50)
```

2. Write Model serializers. Serializers allow complex data (such as model instances) to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types.

```
# Serializers – serializers.py
from myapp.models import Mode, State
from rest_framework import serializers

class ModeSerializer(serializers.HyperlinkedModelSerializer):
        class Meta:
                model = Mode
                fields = ('url', 'name')

class StateSerializer(serializers.HyperlinkedModelSerializer):
        class Meta:
                model = State
                fields = ('url', 'name')
```

Implementation: RESTful Web Services

```
# Models – models.py
from django.db import models

class Mode(models.Model):
        name = models.CharField(max_length=50)

class State(models.Model):
        name = models.CharField(max_length=50)
```

3. Write ViewSets for the Models which combine the logic for a set of related views in a single class.

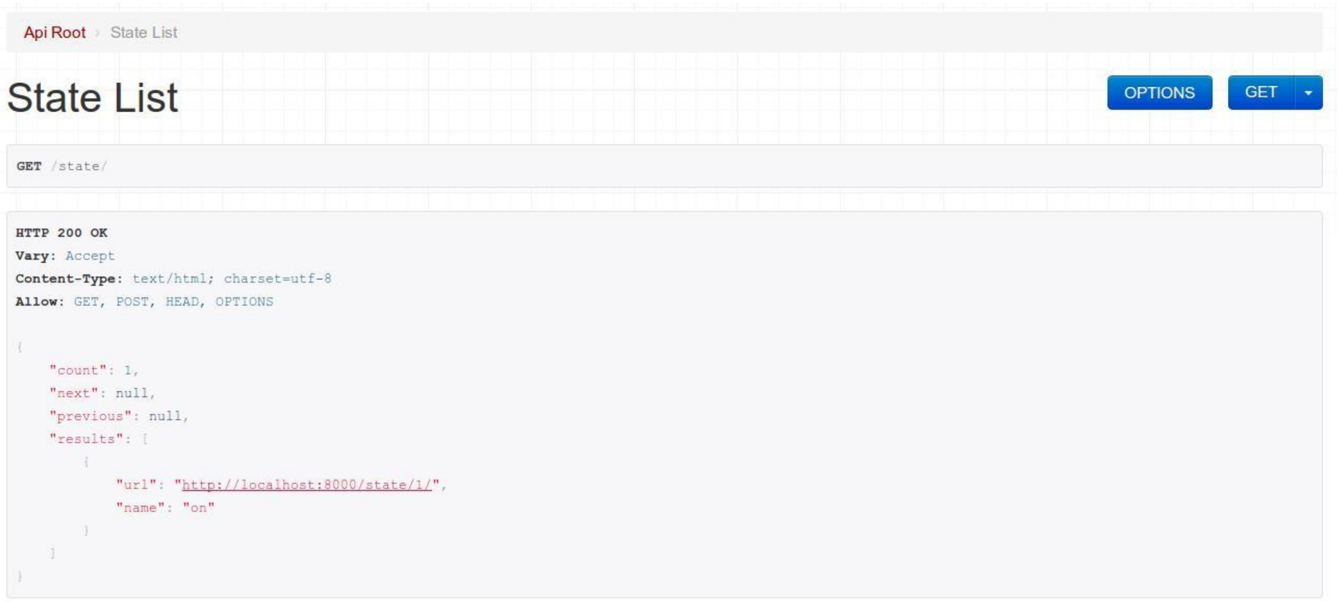
```
# Views – views.py
from myapp.models import Mode, State
from rest_framework import viewsets
from myapp.serializers import ModeSerializer, StateSerializer

class ModeViewSet(viewsets.ModelViewSet):
        queryset = Mode.objects.all()
        serializer_class = ModeSerializer

class StateViewSet(viewsets.ModelViewSet):
        queryset = State.objects.all()
        serializer_class = StateSerializer
```

4. Write URL patterns for the services.
Since ViewSets are used instead of views, we can automatically generate the URL conf by simply registering the viewsets with a router class.
Routers automatically determining how the URLs for an application should be mapped to the logic that deals with handling incoming requests.

```
# URL Patterns – urls.py
from django.conf.urls import patterns, include, url
from django.contrib import admin
from rest_framework import routers
from myapp import views
admin.autodiscover()
router = routers.DefaultRouter()
router.register(r'mode', views.ModeViewSet)
router.register(r'state', views.StateViewSet)
urlpatterns = patterns('',
   url(r'^', include(router.urls)),
   url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework')),
   url(r'^admin/', include(admin.site.urls)),
   url(r'^home/', 'myapp.views.home'),
)
```

Implementation: RESTful Web Services

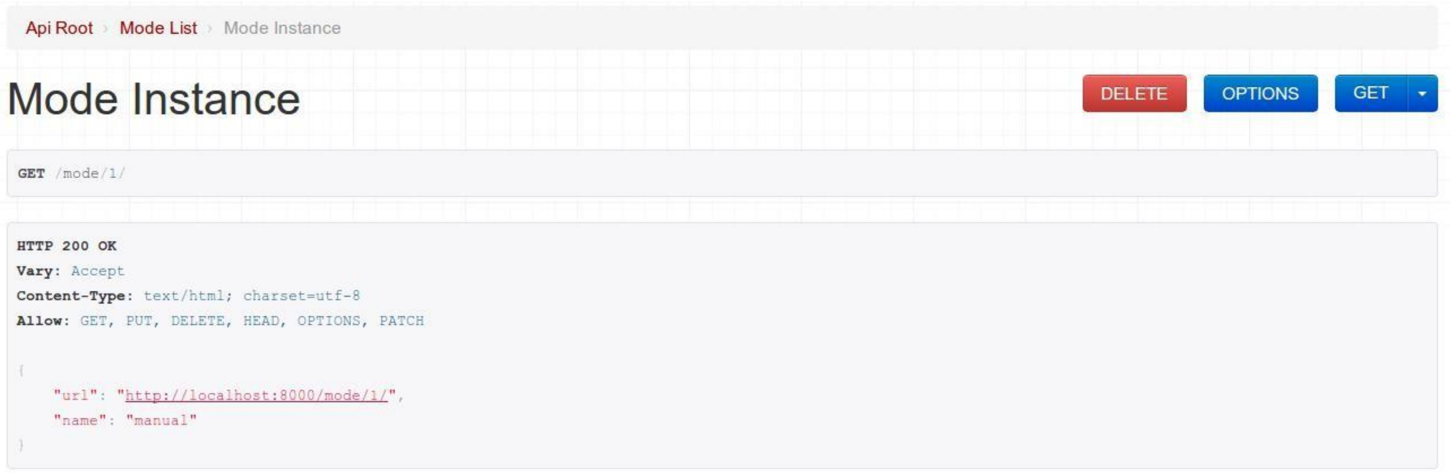
Screenshot of browsable State REST API



```
GET /state/
```

```
HTTP 200 OK
Vary: Accept
Content-Type: text/html; charset=utf-8
Allow: GET, POST, HEAD, OPTIONS

{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "url": "http://localhost:8000/state/1/",
            "name": "on"
        }
    ]
}
```

Screenshot of browsable Mode REST API

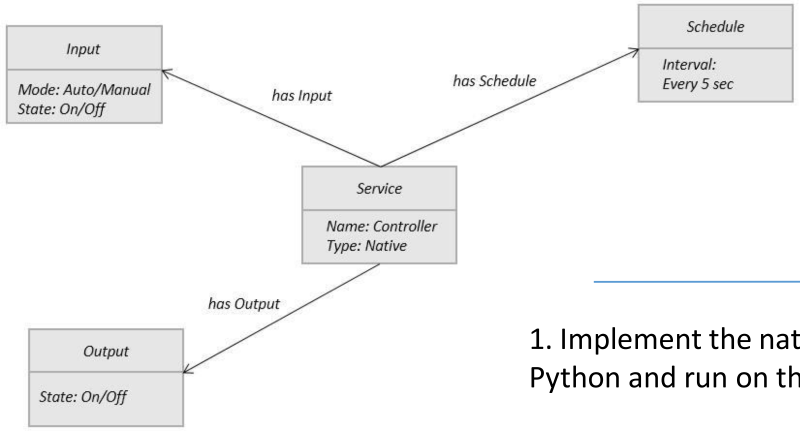


```
GET /mode/1/
```

```
HTTP 200 OK
Vary: Accept
Content-Type: text/html; charset=utf-8
Allow: GET, PUT, DELETE, HEAD, OPTIONS, PATCH

{
    "url": "http://localhost:8000/mode/1/",
    "name": "manual"
}
```

Implementation: Controller Native Service

Native service deployed locally



1. Implement the native service in Python and run on the device

```
#Controller service
import RPi.GPIO as GPIO
import time
import sqlite3 as lite
import sys

con = lite.connect('database.sqlite')
cur = con.cursor()

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

def readldr(PIN):
        reading=0
        GPIO.setup(PIN, GPIO.OUT)
        GPIO.output(PIN, GPIO.LOW)
        time.sleep(0.1)
        GPIO.setup(PIN, GPIO.IN)
        while (GPIO.input(PIN)==GPIO.LOW):
                reading=reading+1
        return reading

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

def switchOffLight(PIN):
        GPIO.setup(PIN, GPIO.OUT)
        GPIO.output(PIN, GPIO.LOW)
```

```
def runAutoMode():
        ldr_reading = readldr(LDR_PIN)
        if ldr_reading < threshold:
                switchOnLight(LIGHT_PIN)
                setCurrentState('on')
        else:
                switchOffLight(LIGHT_PIN)
                setCurrentState('off')

def runManualMode():
        state = getCurrentState()
        if state=='on':
                switchOnLight(LIGHT_PIN)
                setCurrentState('on')
        elif state=='off':
                switchOffLight(LIGHT_PIN)
                setCurrentState('off')

def getCurrentMode():
        cur.execute('SELECT * FROM myapp_mode')
        data = cur.fetchone()          #(1, u'auto')
        return data[1]

def getCurrentState():
        cur.execute('SELECT * FROM myapp_state')
        data = cur.fetchone()          #(1, u'on')
        return data[1]

def setCurrentState(val):
        query='UPDATE myapp_state set name="'+val+'"'
        cur.execute(query)

while True:
        currentMode=getCurrentMode()
        if currentMode=='auto':
                runAutoMode()
        elif currentMode=='manual':
                runManualMode()
        time.sleep(5)
```

Implementation: Application

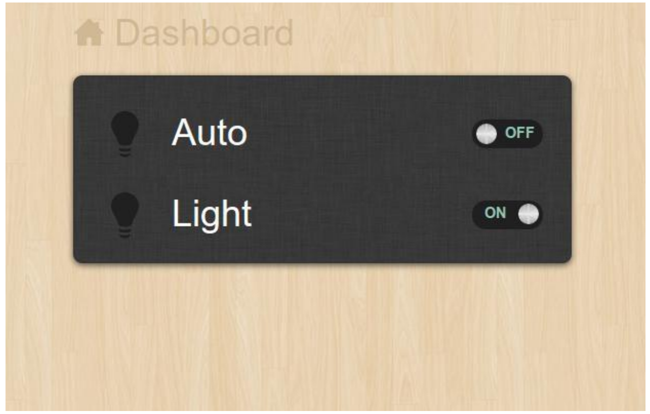
1. Implement Django Application View

```
# Views – views.py
def home(request):
    out=‘’
        if 'on' in request.POST:
                values = {"name": "on"}
                r=requests.put('http://127.0.0.1:8000/state/1/', data=values, auth=(‘username', ‘password'))
                result=r.text
                output = json.loads(result)
                out=output['name']
        if 'off' in request.POST:
                values = {"name": "off"}
                r=requests.put('http://127.0.0.1:8000/state/1/', data=values, auth=(‘username', ‘password'))
                result=r.text
                output = json.loads(result)
                out=output['name']
        if 'auto' in request.POST:
                values = {"name": "auto"}
                r=requests.put('http://127.0.0.1:8000/mode/1/', data=values, auth=(‘username', ‘password'))
                result=r.text
                output = json.loads(result)
                out=output['name']
        if 'manual' in request.POST:
                values = {"name": "manual"}
                r=requests.put('http://127.0.0.1:8000/mode/1/', data=values, auth=(‘username', ‘password'))
                result=r.text
                output = json.loads(result)
                out=output['name']

        r=requests.get('http://127.0.0.1:8000/mode/1/', auth=(‘username', ‘password'))
        result=r.text
        output = json.loads(result)
        currentmode=output['name']
        r=requests.get('http://127.0.0.1:8000/state/1/', auth=(‘username', ‘password'))
        result=r.text
        output = json.loads(result)
        currentstate=output['name']
        return render_to_response('lights.html',{'r':out, 'currentmode':currentmode, 'currentstate':currentstate},
context_instance=RequestContext(request))
```

Implementation: Application

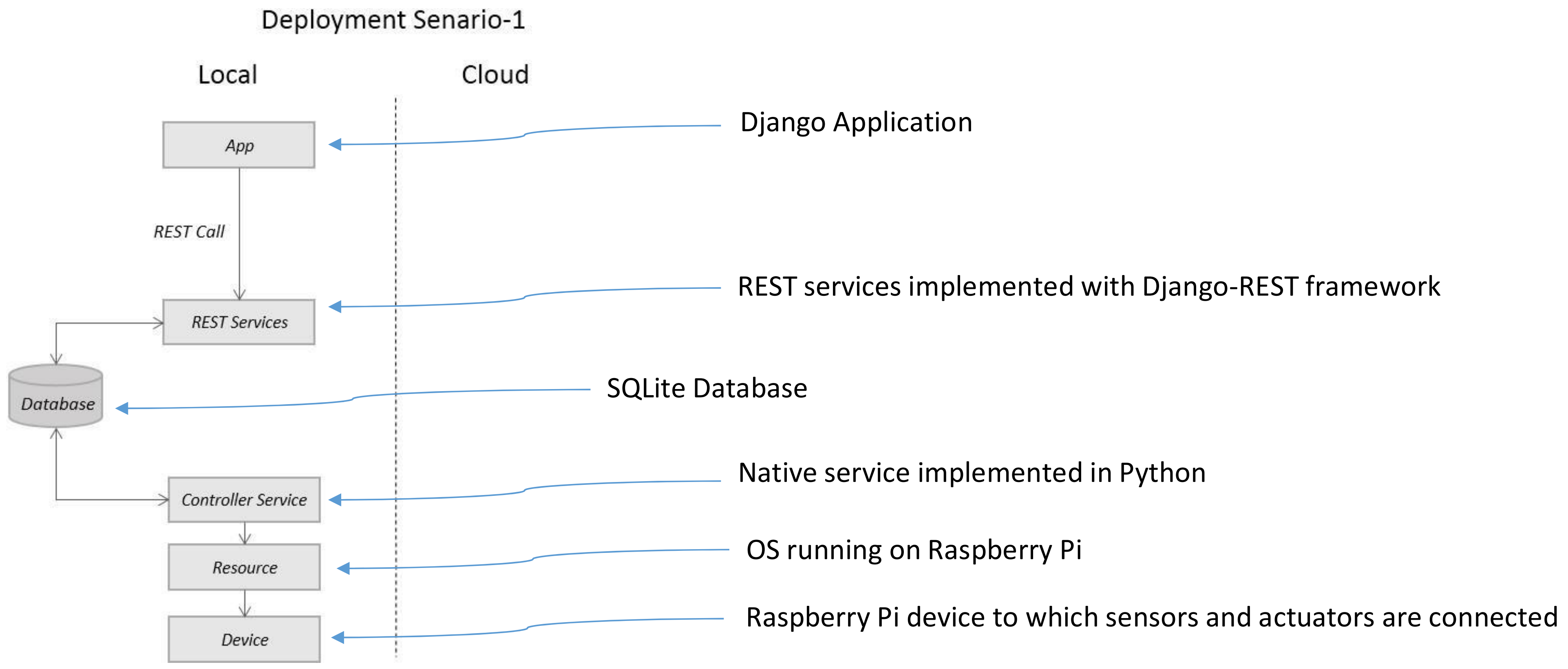
2. Implement Django Application Template

```
<div   class="app-content-inner">
<fieldset>
<div class="field clearfix">
<label class="input-label icon-lamp" for="lamp-state">Auto</label>
<input id="lamp-state" class="input js-lamp-state hidden" type="checkbox">
{% if currentmode == 'auto' %}
<div class="js-lamp-state-toggle ui-toggle " data-toggle=".js-lamp-state">
{% else %}
<div class="js-lamp-state-toggle ui-toggle js-toggle-off" data-toggle=".js-lamp-state">
{% endif %}
<span class="ui-toggle-slide clearfix">
<form id="my_form11" action="" method="post">{% csrf_token %}
<input name="auto" value="auto" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-off">OFF</strong></a>
</form>
<strong class="ui-toggle-handle brushed-metal"></strong>
<form id="my_form13" action="" method="post">{% csrf_token %}
<input name="manual" value="manual" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-on">ON</strong></a>
</form></span>
</div></div>
<div class="field clearfix">
<label class="input-label icon-lamp" for="tv-state">Light</label>
<input id="tv-state" class="input js-tv-state hidden" type="checkbox">
{% if currentstate == 'on' %}
<div class="js-tv-state-toggle ui-toggle " data-toggle=".js-tv-state">
{% else %}
<div class="js-tv-state-toggle ui-toggle js-toggle-off" data-toggle=".js-tv-state">
{% endif %}
{% if currentmode == 'manual' %}
<span class="ui-toggle-slide clearfix">
<form id="my_form2" action="" method="post">{% csrf_token %}
<input name="on" value="on" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-off">OFF</strong></a>
</form>
<strong class="ui-toggle-handle brushed-metal"></strong>
<form id="my_form3" action="" method="post">{% csrf_token %}
<input name="off" value="off" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-on">ON</strong></a>
</form>
</span>
{% endif %}
{% if currentmode == 'auto' %}
{% if currentstate == 'on' %}
<strong  class="ui-toggle-on">&nbsp;&nbsp;&nbsp;&nbsp;ON</strong>
{% else %}
<strong  class="ui-toggle-on">&nbsp;&nbsp;&nbsp;&nbsp;OFF</strong>
{% endif %}{% endif %}
</div>
</div>
</fieldset></div></div></div>
```



Finally - Integrate the System

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


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