UNIT-I

Concepts

Introduction to Internet of Things, Physical Design of IoT, Logical Design of IoT, IoT Enabling Technologies, IoT Levels. M2M

Introduction

Definition:

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 environment. Internet of Things (IoT) comprises things that have unique identities and are connected to the Internet

Applications of IoT:

- Home
- Cities
- Environment
- Energy
- Retail
- Logistics
- Agriculture
- Industry
- Health & lifestyle

Characteristics of IoT:

- **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

- **Self-Configuring**

  IoT devices may have self-configuring capability, allowing a large number of devices to work together to provide certain functionality like setup networking, fetch latest software upgrades

- **Interoperable Communication Protocols**

  IoT devices may support a number of interoperable communication protocols and can communicate with other devices and also with the infrastructure

- **Unique Identity**

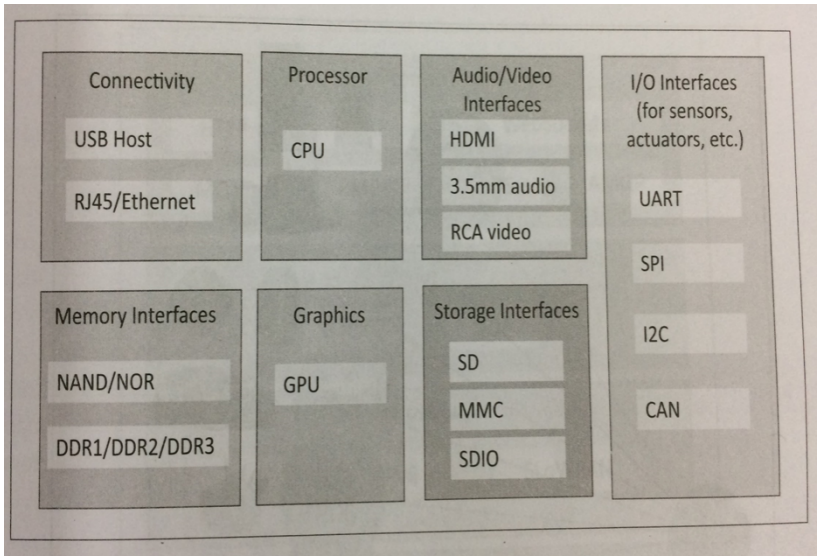
  Each IoT device has a unique identity and a unique identifier (such as an IP address or a URI)

- **Integrated into Information Network**

  IoT devices are usually integrated into the information network that allows them to communicate and exchange data with other devices and systems

Physical Design of IoT

- The "Things" in IoT usually refers to IoT devices
- IoT devices 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
- It process the data either locally or send the data to centralized server



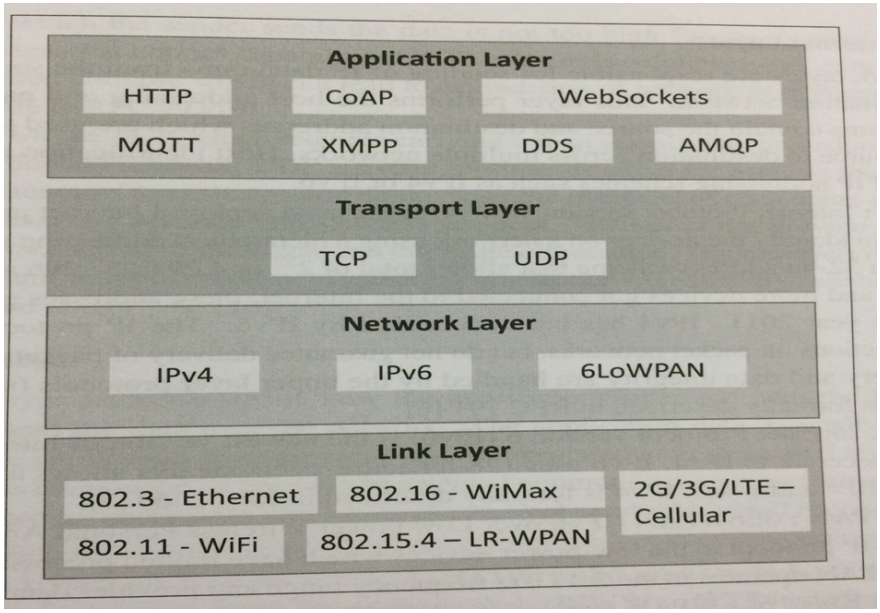
Generic block diagram of IoT Device

- IoT devices varied in types

Wearable Sensors, Smart watches, LED lights, automobiles, and industrial machines

- Almost all IoT devices generate data in some form or the other which when processed by data analytics systems leads to useful information to guide further actions locally or remotely

IoT Protocols



Link Layer: Link Layer protocol determines how the data is physically sent over the networks physical layer or medium.

- 802.3 Ethernet: 802.3 10BASE5 uses Coaxial cable
  802.3.i 10BASE-T Ethernet over Copper twisted-pair
  802.3.j 10BASE-F Ethernet over fiber optic connection
  802.3.ae 10Gbps Ethernet over fiber.

- 802.11 Wifi: 802.11a operate the 5 GHz
  802.11b operate in 2.5 GHz
  802.11.g operate in 2.5 GHz
  802.11n operate in 2.4/5 GHz
  802.11ac Operate in 5GHz
  802.11ad Operate in 60GHz

- 802.16 WiMax: IEEE 802.16 is a collection of broadband standards including extensive description for the link layer. WiMax standards provide data rate from 1.5 Mbps to 1Gbps.The recent update (802.16m) provides 100 Mbps for Mobile station and 1Gbps for fixed stations.

- 802.15.4 LR-WPAN: IEEE 802.15.4 is a collection of standards for low-rate wireless personal area network. These standards form the basis of specification for high level communication such as ZigBee. Data rate provides from 40Kbps to 250 Kbps.
- 2G/3G/4G-Mobile Communication: There are different generations of mobile communication standards including second generation (GSM,CDMA),third generation(UMTS,CDMA2000) , fourth generation (4G including LTE). Data rates ranging from 9.6 Kbps (2G) to 100Mbps(4G).

Network/Internet Layer

- **IPV4:**IPV4 uses 32 bit address scheme that allows 2^{32}
- **IPV6:** IPV6 uses 128 bit address scheme that allows total 2^{128} 3.4×10^{38} address
- **6LoWPAN:** IPV6 over Low Power Wireless Personal Area network brings IP protocol to the low-power devices. It operates in the 2.4 GHz 250 Kbps.

Transport Layer

Transport layer protocols provide end-to-end message transfer capability independent of the underlying network.

- TCP: Transmission Control Protocol(TCP) is the most widely used transport layer protocol, that is used by web browsers, email programs and file transfer. TCP is connection oriented and state-full protocol. While IP protocol deals with sending packets.TCP ensures reliable transmission of packets in-order.TCP also provides error detection capability so that duplicate packets can be discarded and lost packets are retransmitted.
- UDP: Unlike TCP, which requires carrying out an initial setup procedure, UDP is a connectionless protocol.UDP is useful for time-sensitive applications that have ver small data units to exchange and do not want the overhead of connection setup.UDP is a transaction oriented and stateless protocol.UDP does not provide guaranteed delivery, ordering of messages and duplication elimination.

Application Layer

Application layer protocol defines how the application interface with the lower layer protocols to send the data over the network. The application data, typically in files, is encoded by the application layer protocol and encapsulated in the transport layer protocol.

- HTTP:Hyper Text Transfer Protocol. The protocol follows a request-response model where a client sends request to a server using the HTTP commands. HTTP is a stateless protocol. HTTP uses URI to identify HTTP resources.
- CoAP: Constrained Application Protocol is an application layer protocol for machine-to-machine application(M2M),meant for constrained environment with constrained devices and constrained networks. Like HTTP, CoAP is a web transfer protocol and uses a request-response model.

- Web Socket:WebSocket protocol allows full-duplex communication over a single socket connection for sending message between client and server. WebSocket is based on TCP and allows stream of messages to be sent back and forth between the client and server while keeping the TCP connection open.
- MQTT: Message Queue Telemetry Transport is a light-weight messaging protocol based on the publish-subscribe model. MQTT uses a client-server architecture where the client connects to the server and publish message to topic on the server. The broker forwards the messages to the clients subscribed to topics.
- XMPP: Extensible Messaging and Presence Protocol is a protocol for real-time communication and streaming XML data between networks entities. XMPP power wide range of application including messaging , presence, data syndication, gaming, multi-party chatting and voice/video calls. XMPP allows sending small chunks of XML data from one network to another in near real-time.

- DDS: Data Distribution Service is data –centric middleware standard for device-to-device or machine-to-machine communication.DDS uses a publish-subscribe model where publishers create topic to which subscribers can subscribe. Publisher is an object responsible for data distribution and the subscriber is responsible for receiving published data.

- AMQP: Advanced Message Queuing Protocol is an open application layer protocol for business messaging. AMQP supports both point-to-point and publisher/subscriber models, routing and queuing. AMQP brokers receive messages from publishers and route them over connection to consumers. Publisher publishes the messages to exchanges which then distribute message copies to queues. Messages are either delivered by the broker to the consumer which have subscribed to the queues or the consumers can pull the message from the queues.

Logical Design of IoT

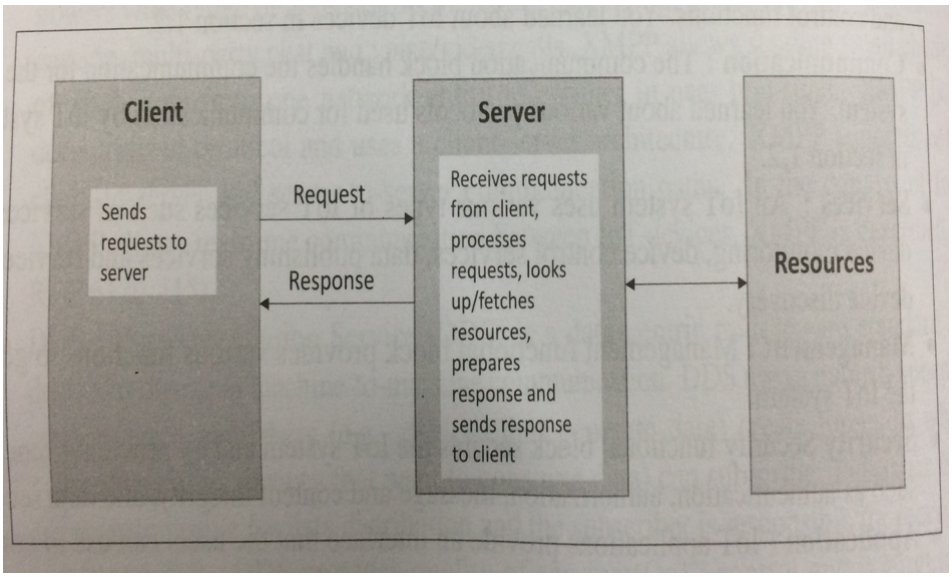
- IoT Functional Blocks
- IoT Communication Models
- IoT communication APIs

IoT Functional Block

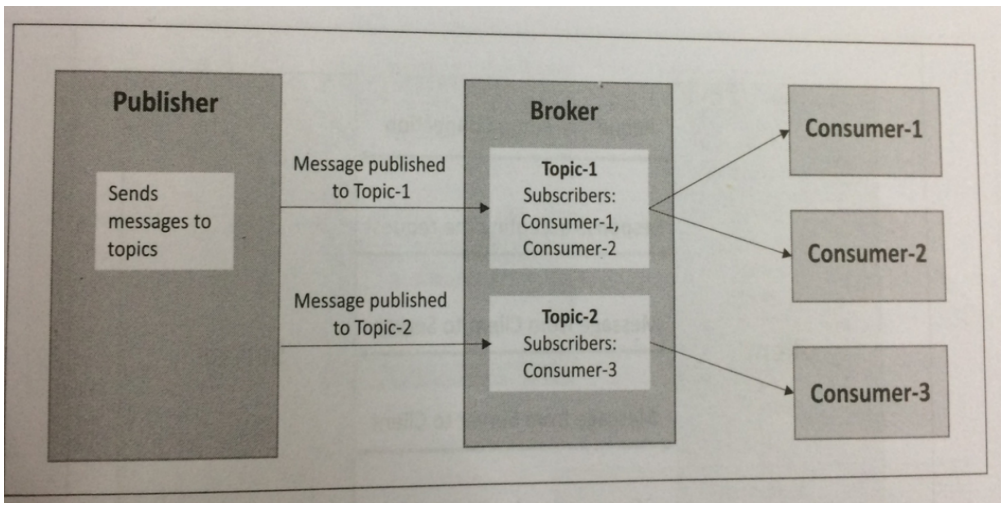
- An IoT system comprises of a number of functional blocks that provide the system the capabilities for identification, sensing, actuation, communication, and management
  - Device : provide sensing, actuating, monitoring and control functions.
  - Communication : The communication for the IoT systems.
  - Services : IoT system uses various types of services.
  - Management : Manages various functions in system.
  - Security : Secures the IoT system and provides security services.
  - Application :provides an interface that users can use to control IoT system.

IoT Communication Models

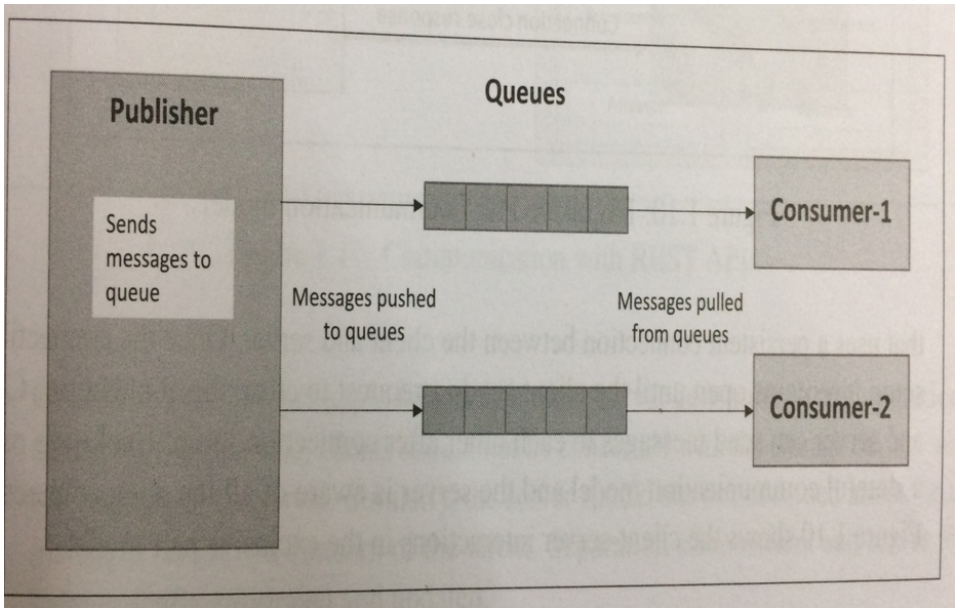
- Request – Response
- Publish – Subscribe
- Push-Pull
- Exclusive Pair



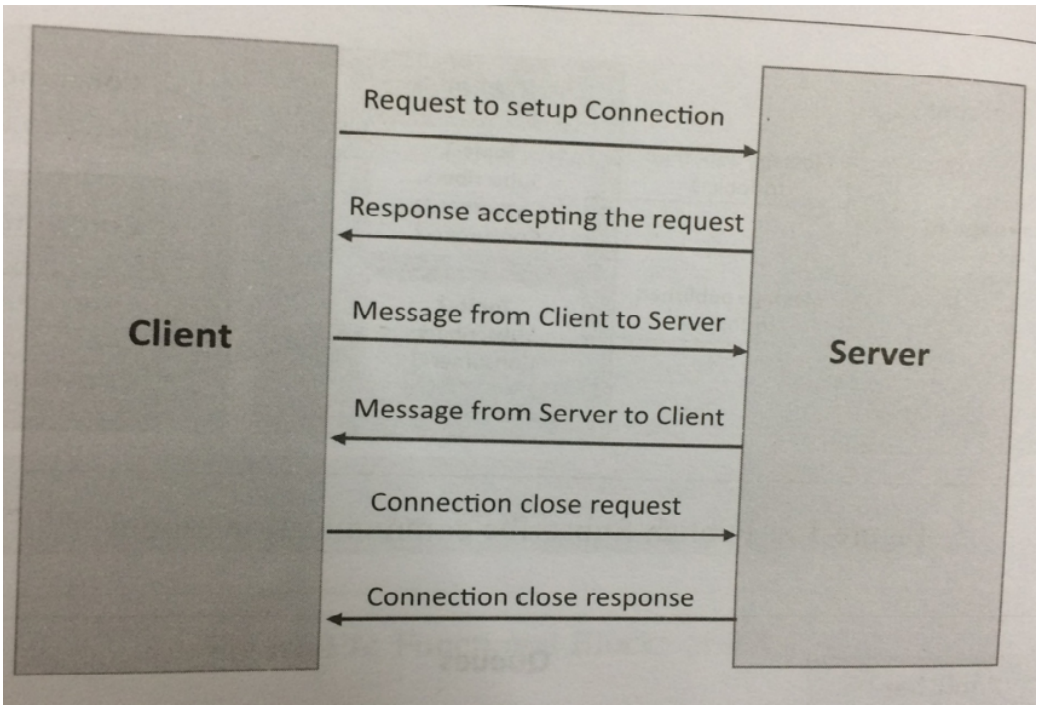
Request - Response communication model



Publish- Subscribe communication model



Push pull communication model



Exclusive pair communication model

IoT Communication APIs

- REST – based Communication APIs
- WebSocket – based Communication APIs

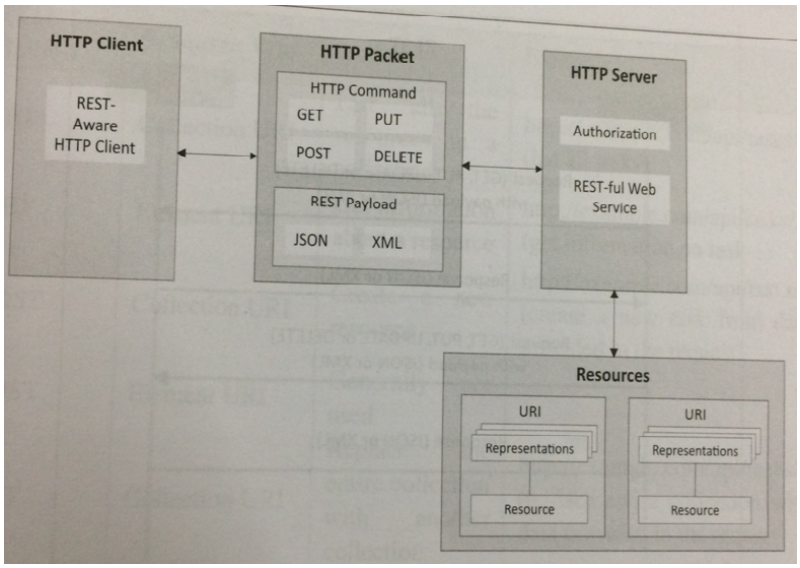
REST-based Communication APIs

- 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 system 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

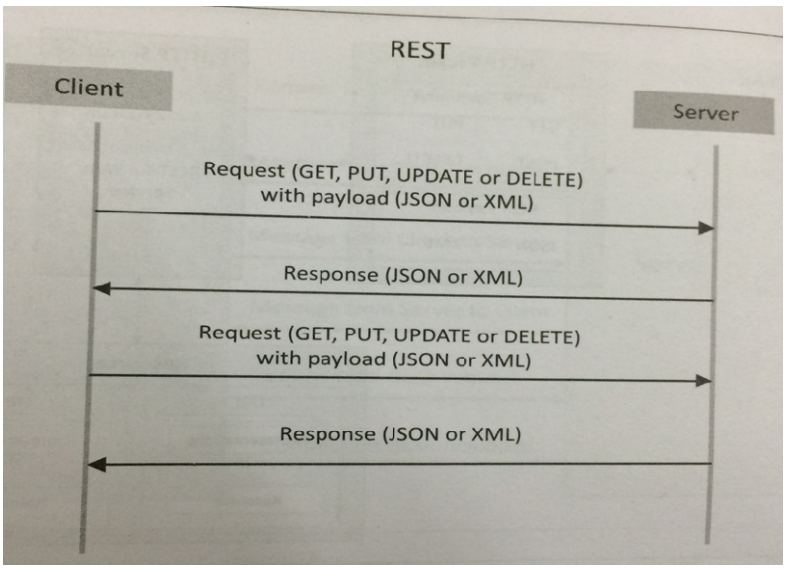
REST Architectural Constraints

- Client –Server
  - The principle behind the client-server constraints is the separation of concerns
- Stateless
  - Each request from client to server must contains all the information necessary to understand the request
  - The session state is kept entirely on the client
- Cache-able
  - Cache constraints requires that the data within a response to a request be implicitly or explicitly labeled as cache-able or non-cache-able
- Layered System
  - Layered system constraints, constraints the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting
- Uniform Interface
  - Interface constraints 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



Communication with REST APIs



Request Response model used by REST

HTTP Method	Resource Type	Action	Example
GET	Collection URI	List all the resources in a collection	http://example.com/api/tasks/ (list all tasks)
GET	Element URI	Get information about a resource	http://example.com/api/tasks/1/ (get information on task-1)
POST	Collection URI	Create a new resource	http://example.com/api/tasks/ (create a new task from data provided in the request)
POST	Element URI	Generally not used	
PUT	Collection URI	Replace the entire collection with another collection	http://example.com/api/tasks/ (replace entire collection with data provided in the request)
PUT	Element URI	Update a resource	http://example.com/api/tasks/1/ (update task-1 with data provided in the request)
DELETE	Collection URI	Delete the entire collection	http://example.com/api/tasks/ (delete all tasks)
DELETE	Element URI	Delete a resource	http://example.com/api/tasks/1/ (delete task-1)

Table 1.1: HTTP request methods and actions

WebSocket-based Communication APIs

- WebSocket APIs allow bi-directional, full duplex communication between clients and servers
- It follows the exclusive pair communication model
- WebSocket APIs reduce the network traffic and latency as there is no overhead for connection setup and termination requests for each message

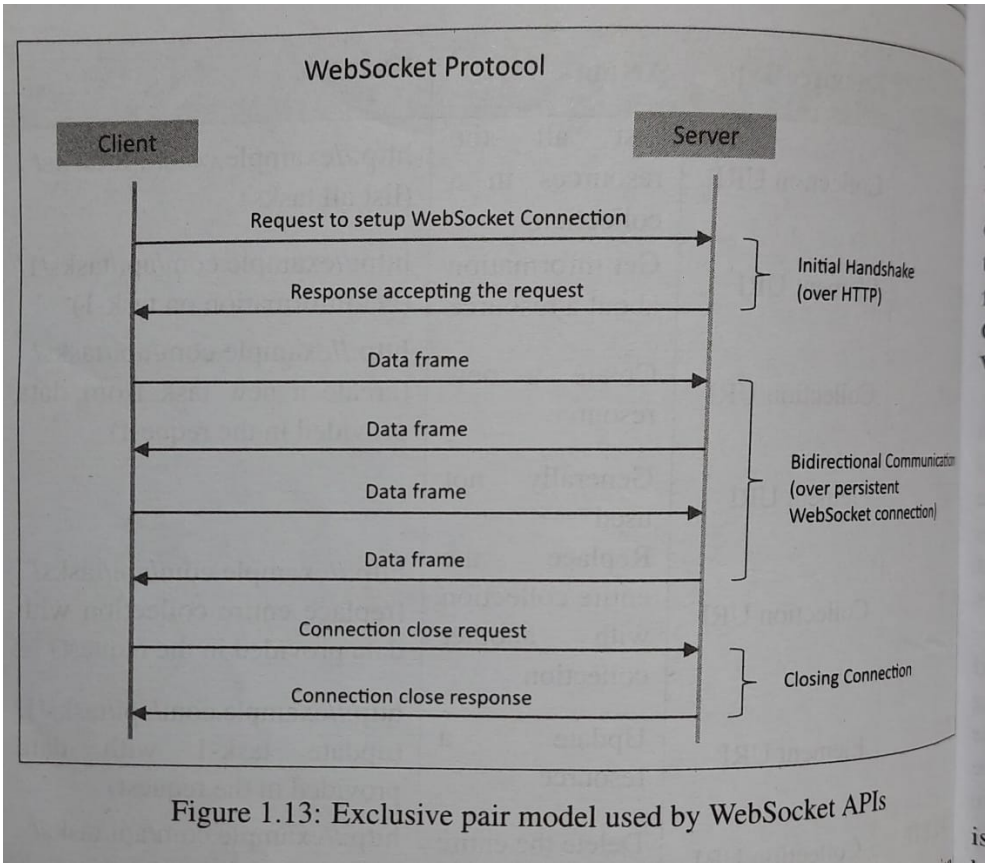


Figure 1.13: Exclusive pair model used by WebSocket APIs

IoT Enabling Technologies

- Wireless Sensor Networks
- Cloud Computing
- Big Data Analytics
- Communication Protocols
- Embedded Systems

Wireless Sensor Networks

- A Wireless Sensor Network (WSN) comprises of **distributed devices with sensors** which are used to monitor the environmental and physical conditions
- A WSN consist of a **number of end-nodes and routers and a coordinator** (collects the data from all the nodes)
- Eg : Weather monitoring system, Soil moisture monitoring system (at various location), Surveillance system etc..

Cloud Computing

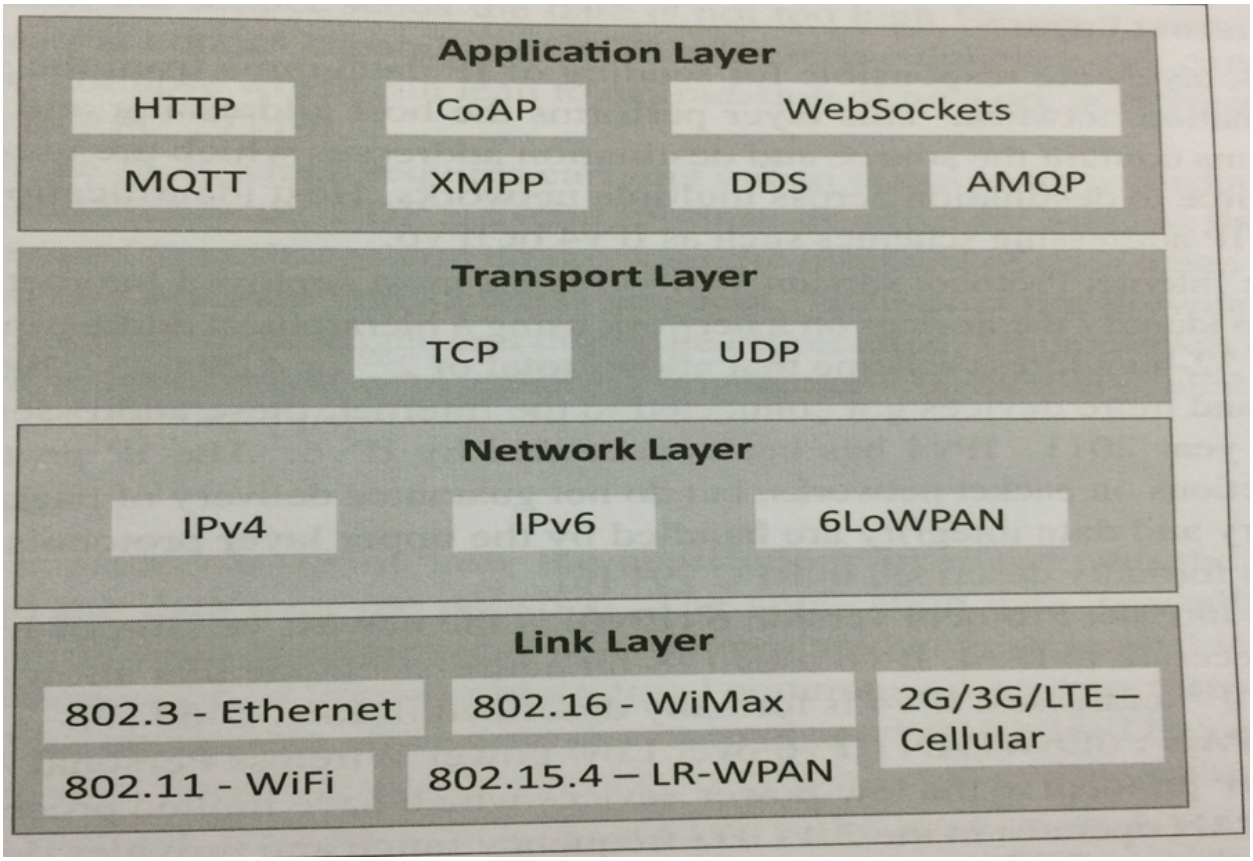
- Cloud computing is a **transformative computing paradigm** that involves delivering applications and services over the Internet
- Cloud computing involves provisioning of computing, networking and storage resources on demand and providing these resources as metered services to the users, in a "**pay as you go model**"
- **Cloud computing services are offered to users in different forms**
  - Infrastructure- as- a- Service (IaaS)
  - Platform-as-a-Service (PaaS)
  - Software-as-a-Service (SaaS)

Big Data Analytics

- Big data is a **collections of data sets whose volume, velocity or variety is so large** that it is difficult to store, manage, process and analyze the data using traditional databases and data processing tools
- Examples of big data generated by IoT systems
  - Weather Monitoring Stations
  - Machine sensor data from industrial systems
  - Health and fitness data
  - Location and tracking systems

Communication Protocols

- Communication protocols form the backbone of IoT systems and **enable network connectivity and coupling to applications**
- Communication protocols allow devices to exchange data over the network



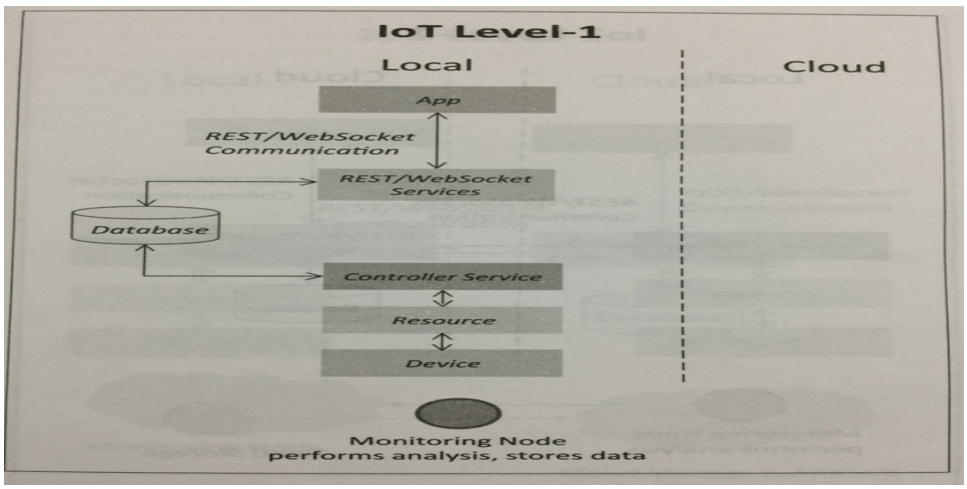
Embedded System

- An Embedded system is a system that has computer hardware and software embedded to **perform specific tasks**
- Embedded system range from low-cost **miniaturized devices** such as
  - Digital watch
  - Digital cameras
  - Vending machines
  - Appliances (washing machine) etc…

IoT Levels & Deployment Templates

- **Device** – An IoT device allows identification, remote sensing, actuating and remote monitoring capabilities
- **Resource** – Resources are software components on the IoT device for accessing, processing, and storing sensor information, or controlling actuators connected to the device

- **Controller Service** – It sends data from the device to the web service and receives commands from the application for controlling the device
- **Database** – Database can be either local or in the cloud and stores the data generated by the IoT
- **Web Service** – Web services serve as a link between the IoT device, application, database and analysis components
- **Analysis Component** – It is responsible for analyzing the IoT data and generate results
- **Application** – It is an interface that the users can use to control and monitor various aspects of the IoT system

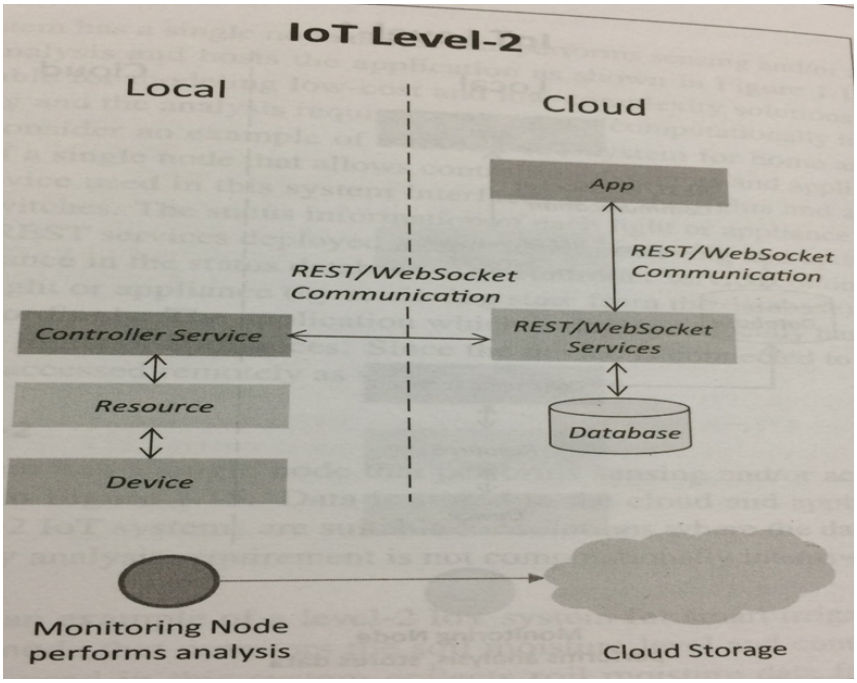


IoT Level -1

A Level -1 IoT system has a single node/device that performs sensing/or actuating, stores data, perform analysis and host 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 are not computationally intensive

Eg: Home automation

The system consists of a single node that allows controlling the lights and appliances in a home remotely. The device used in the system interfaces with the light and appliances using electronic relay switches. The status information of each light or appliance is maintained in the local database. The controller service continuously monitors the state of each light or appliance and triggers the relay switches accordingly.

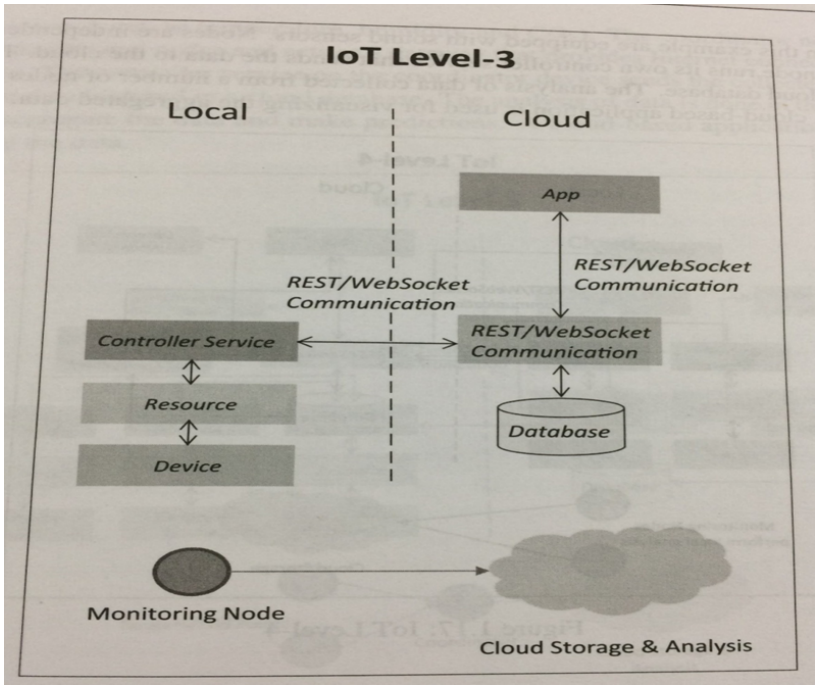


IoT Level -2

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 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 2 system for smart irrigation. The system consists of a single node that monitors the soil moisture level and controls the irrigation system. The device used in this system collects soil moisture data from sensors. The controller service continuously monitors the moisture levels. If the moisture level drops below a threshold, the irrigation system is turned on. The controller also sends the moisture data to the cloud. A cloud based application is used for visualizing the moisture level over a period of time, which can help in making decisions about iriigation schedule

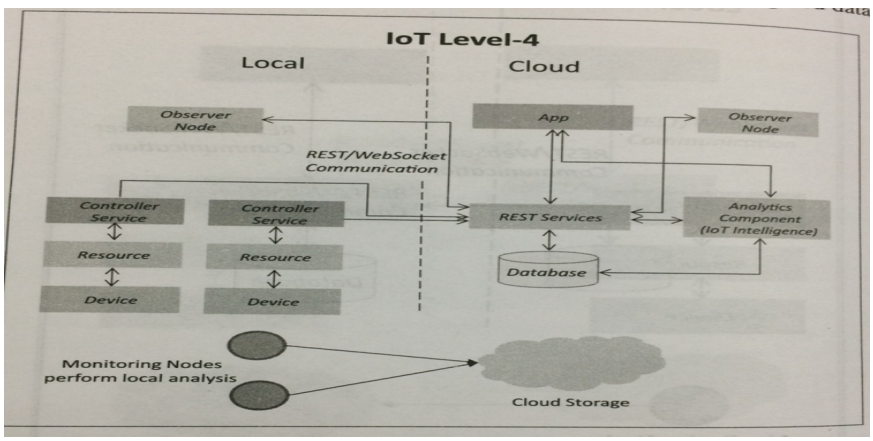


IoT Level -3

A Level 3 IoT system has a single node. Data is stored and analyzed in the cloud. Level 3 IoT systems are suitable for solutions where the data involved is big and the analysis requirements are computationally intensive

Example :

The system for tracking package handling. The system consists of a single node(for a package) that monitors the vibration levels for a package being shipped. The device in this system uses accelerometer and gyroscope sensors for monitoring vibration levels. The controller service sends the sensor data to the cloud in real-time. The data is stored in the cloud and also visualized using cloud based application. The analysis components in the cloud can trigger alters if the vibration levels become greater than a threshold

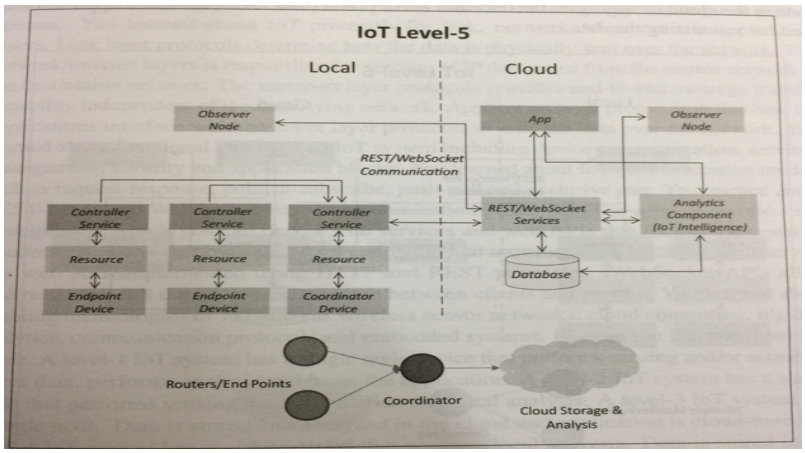


IoT Level -4

Level 4 system has multiple nodes that perform local analysis. Data is stored in the cloud. Level 4 contains local and cloud based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices. Observer nodes can process information and use it for various applications, however, observer nodes do not perform any control functions. Level 4 IoT systems are suitable for solutions where multiple nodes are required, the data involved is big and the analysis requirements are computationally intensive

Example:

Noise Monitoring. The system consists of multiple nodes placed in different locations for monitoring noise level in an area



IoT Level -5

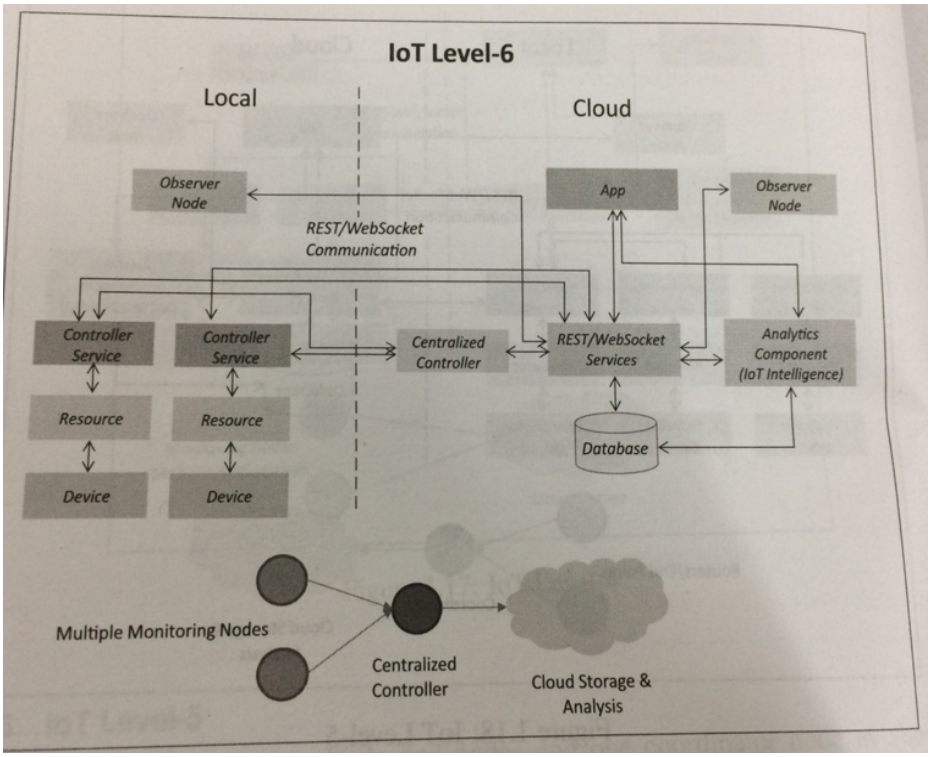
Level 5 IoT system has multiple end nodes and one coordinator node. The end nodes that perform sensing and/or actuation. Coordinator node collects data from the end nodes and sends to the cloud.

Data is stored and analyzed in the cloud

Level 5 systems are suitable for solutions based on Wireless sensor networks, in which the data involved is big and the analysis requirements are computational intensive

Example : Forest fire detection

The system consists of multiple nodes placed in different locations for monitoring temperature, humidity and carbon dioxide (CO2) level in the forest.



IoT Level -6

Level 6 system has multiple end nodes that perform sensing and/or actuation and send data to the cloud. Data is stored in the cloud

The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes

Example : Weather monitoring

M2M:

Machine-to-Machine(M2M) refers to networking of machines for the purpose of remote monitoring and control and data exchange. An M2M area network comprises of machine which have embedded hardware modules for sensing , actuating and communication. Various communication protocol can be used for M2M local area networks such as ZigBee, Bluetooth, ModBus, M-Bus, Power Line Communication, 6LoWPAN,IEEE 802.15.4 etc. These Communication protocols provide connectivity between M2M nodes with in an M2M area network.

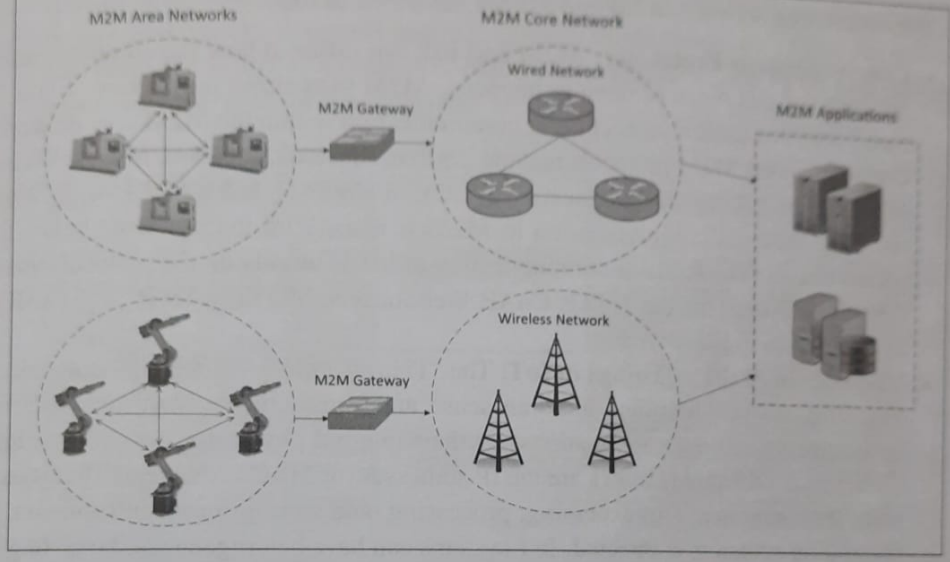


Figure 3.1: M2M system architecture

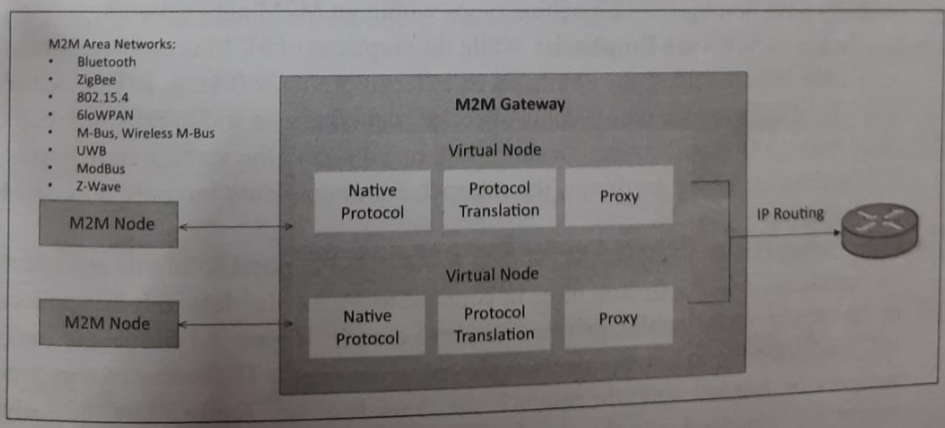


Figure 3.2: Block diagram of an M2M gateway

Difference between IoT and M2M:

Even both M2M and IoT involve networking of machines or devices, they differ in the underlying technologies , system architectures and types of applications.

The differences between M2M and IoT are described as follows

Communication Protocol

- M2M and IoT can differ in how the communication between the machine or devices happens. M2M uses either proprietary or non-IP based communication protocol for communication within the M2M area network.

- Commonly uses M2M protocol include ZigBee, Bluetooth, ModBus, M-Bus, Wireless M-Bus, Power Line Communication,,6LoWPAN,IEEE 802.15.4 Z-Wave, etc.,

- The focus of communication in M2M is usually on the protocols bellow the network layer.
- The focus of IoT is usually on the protocols above the network layer such as HTTP, CoAP, WebSockets, MQTT,XMPP,DDS,AMQP..etc.,

Machine in M2M vs Things in IoT

- The things in IOT refers to physical objects that have unique identifier and can sense and communicate with their external environment or their internal physical states. The unique identifiers for the things in IoT are the IP addresses.

Hardware Vs Software Emphasis

- While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software. IoT devices run specialized software for sensing data collection, data analysis and interfacing with the cloud through IP-based communication.

Data Collection and Analysis

- M2M data is collected in point solution and often in on-premises storage infrastructure. In contrast to M2M, the data in IoT is collected in the cloud.

Applications

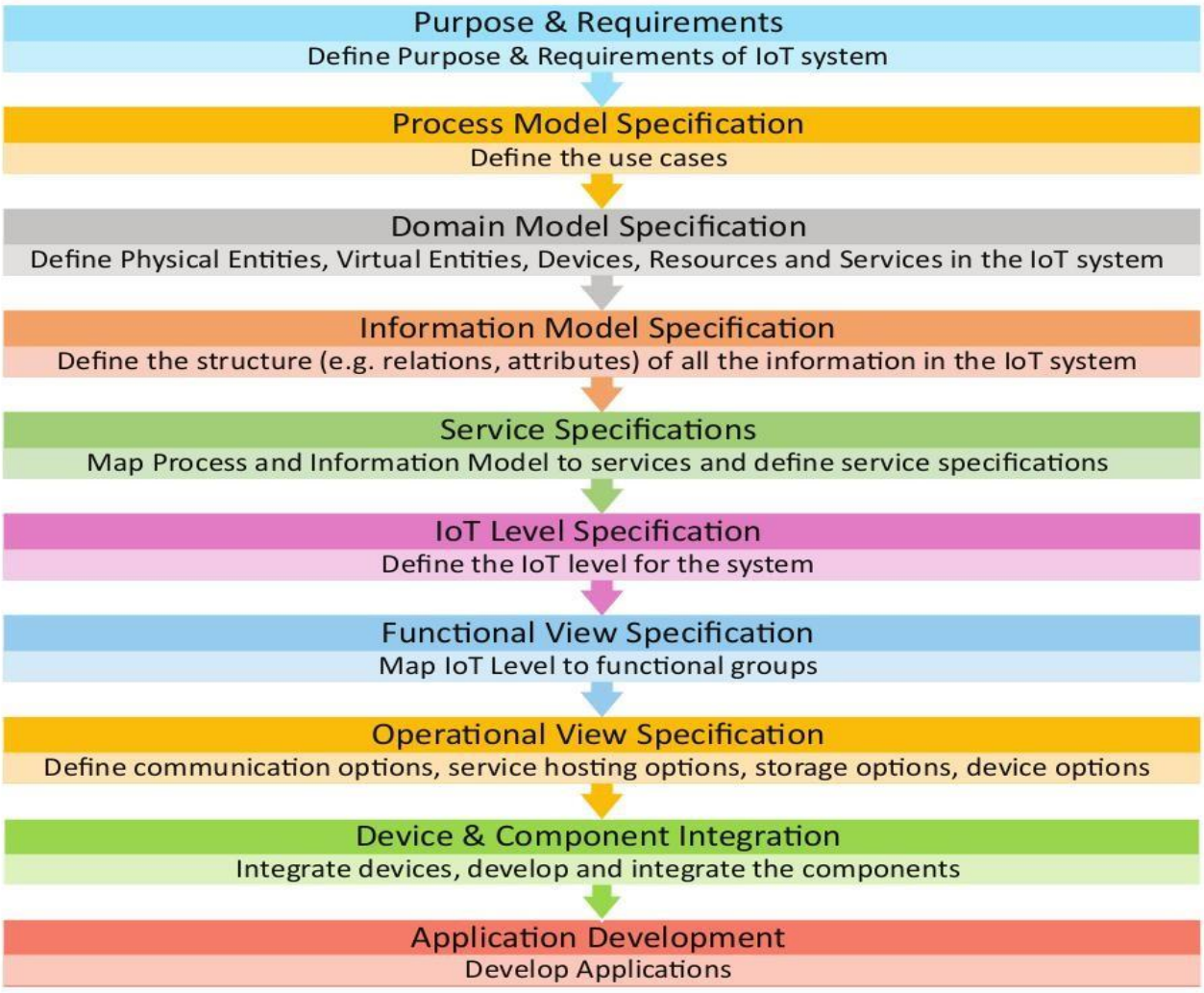
- M2M data is collected in point solution and can be accessed by on-premises application such as diagnosis application, service management applications and on-premises enterprise application.

UNIT-II

Concepts

IoT Design Methodology: Purpose & Requirements Specification, Process Specification, Domain Model Specification, Information Model Specification, Service Specification, IoT Level Specification, Functional View Specification, Operational View Specification, Device & Component Integration and Application Development

IoT Design Methodology:



Each of these steps is explained in the section that follows. To explain these steps, we uses the example of a smart IoT-based home automation system.

Purpose & Requirement Specification

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

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 specification.

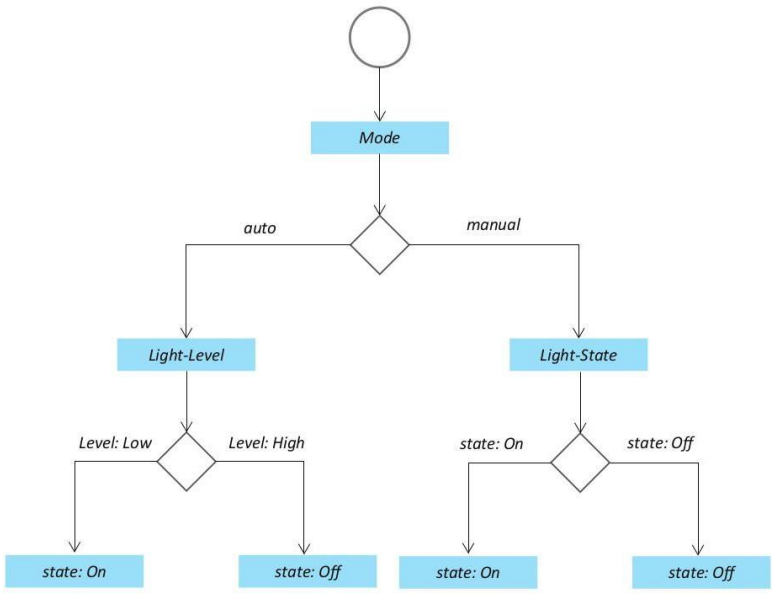


Figure 5.2 shows the process diagram for the home automation system.

- The process diagram shows the two modes of the system – auto and manual.
- In a process diagram, the circle denotes the start of a process, diamond denotes a decision box and rectangle denotes a state or attribute.
- When the auto mode is chosen, the system monitors the light level. If the light level is low, the system changes the state of the light to "on".
- Whereas, if the light level is high, the system changes the state of the light to "off". When the manual mode is chosen, the system checks the light to "on".
- Whereas, if the light state set by the user is "off", the system changes the state of light to "off".

Domain Model Specification

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

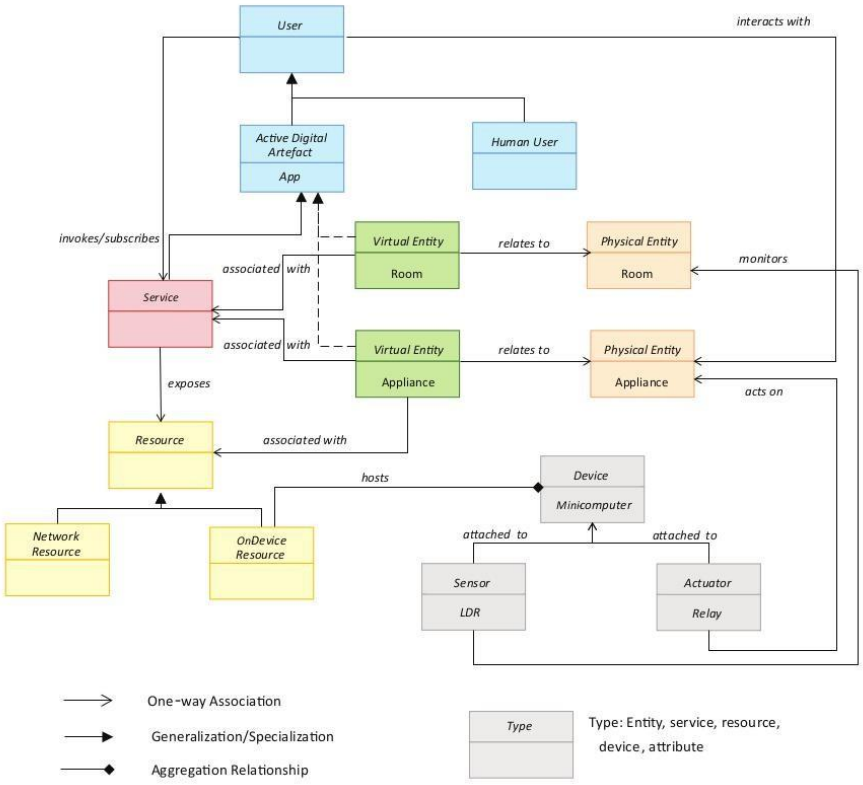


Figure 5.3 shows the domain model for the home automation system example.

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

Physical Entity :

- Physical Entity is a discrete and identifiable entity in the physical environment (e.g. a room, a light an appliance, a car, etc.).
- The IoT system provides information about the Physical Entity (using sensors) or performs actuation upon the Physical Entity (e.g., switching on a light).
- In the home automation example, there are two Physical Entities involved – one is the room in the home (of which the lighting conditions are to be monitored) and the other is the light appliance to be controlled.

Virtual Entity :

- Virtual Entity is a representation of the Physical Entity in the digital world.
- For each Physical Entity, there is a Virtual Entity in the domain model.
- In the home automation example, there is one Virtual Entity for the room to be monitored, another for the appliance to be controlled.

Device :

- Device provides a medium for interactions between Physical Entities and Virtual Entities.
- Devices are either attached to Physical Entities or placed near Physical Entities.
- Devices are used to gather information about Physical Entities (e.g. using actuators) or used to identify Physical Entities (e.g., using tags).
- In the home automation example, the device is a single –board mini computer which has light sensor and actuator (relay switch) attached to it.

Resource :

- Resources are software components which can be either "on-device" or "network-resources".
- On-device resources are hosted on the device and include software components that either provide information on or enable actuation upon the Physical Entity to which the device is attached.
- Network resources include the software components that are available in network (such as a database).
- In the home automation example, the on-device resource is the operating system that runs on the single-board minicomputer.

Service:

- Services provide an interface for interacting with the Physical Entity.
- Services access the resources hosted on the device or the network resources to obtain information about the Physical Entity or perform actuation upon the Physical Entity.
- In the home automation example, there are three services:

  (1) a service that sets mode to auto or manual, or retrieves the current mode:

  (2) a service that sets the light appliance state to on/off, or retrieves the current light state

  (3) a controller service that runs as a native service on the device.
- When in auto mode, the controller service monitors the light level and switches the light on/off and updates the status in the status database.
- When in manual mode, the controller service retrieves the current state from the database and switches the light on/off.
- The process of deriving the services from the process specification and information model is described in the later sections.

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.
- In the home automation example, there are two Virtual Entities – a Virtual Entity for the light appliance (with attribute –light state) and a Virtual Entity for the room (with attribute- light level).

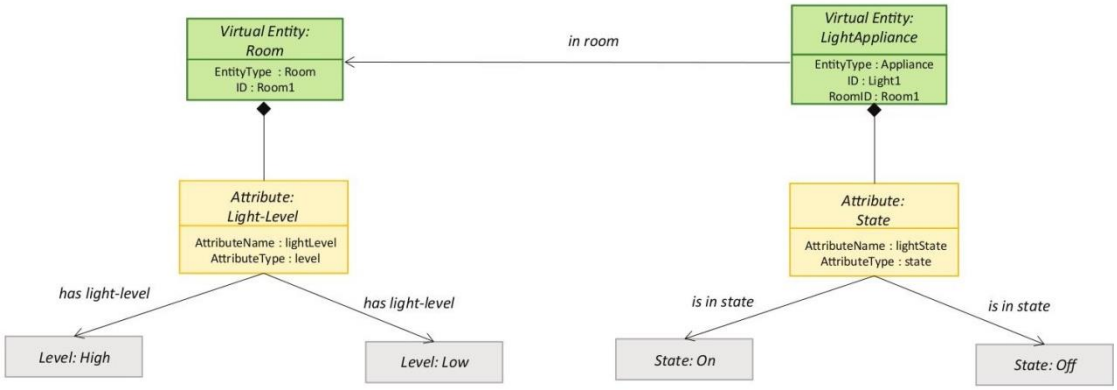


Figure 5.4 shows the information Model for the home automation system example.

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 types, service inputs/output, service endpoints, service schedules, service preconditions and service effects.
- You learned about the process specification and Information Model in the previous sections.

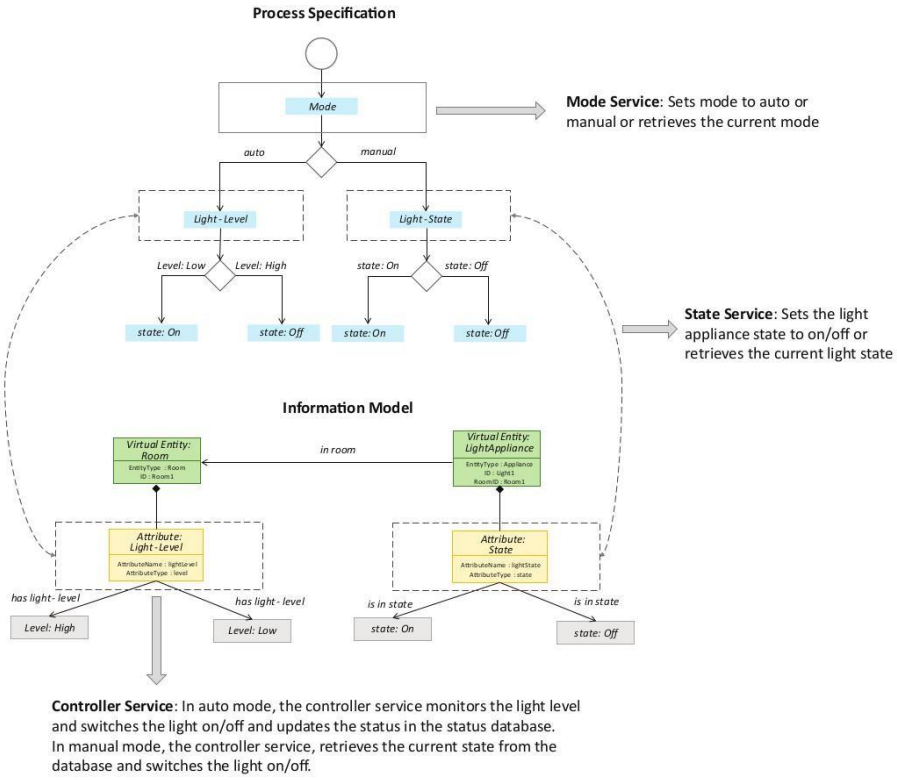
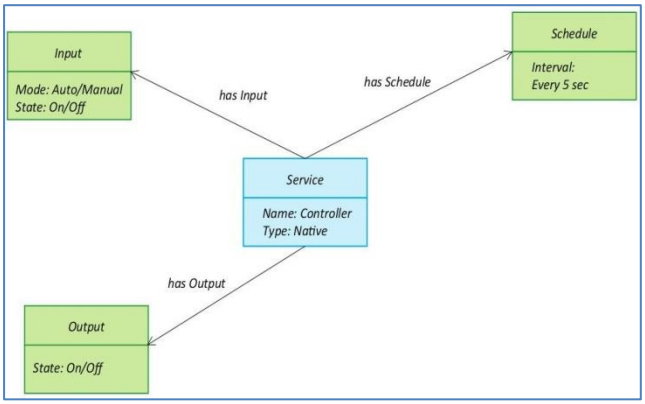
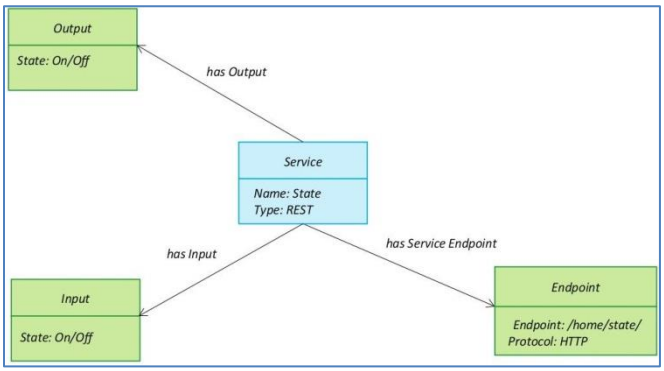


Figure 5.5 shows an example of deriving the services from the process specification and information model for the home automation IoT system.

- From the process specification and information model, we identify the states and attributes.
- For each state and attribute we define a service. These services either change the state or attribute values or retrieve the current values.
- For example, the Mode service sets mode to auto or manual or retrieves the current mode.
- The State service sets the light appliance state to on/off or retrieves the current light state.
- The Controller service monitors the light level in auto mode 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.





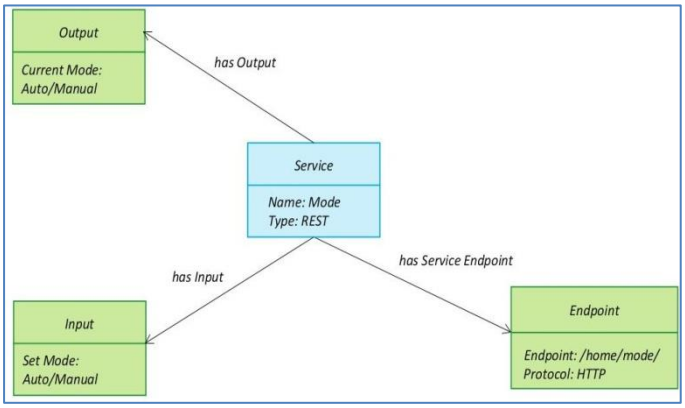


Figure 5.6, 5.7 and 5.8 show specifications of the controller, mode and state services of the home automation system.

- The Mode service is a RESTful web service that sets mode to auto or manual (PUT request), or retrieves the current mode (GET request) .
- The mode is update 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.

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

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 deployments levels.

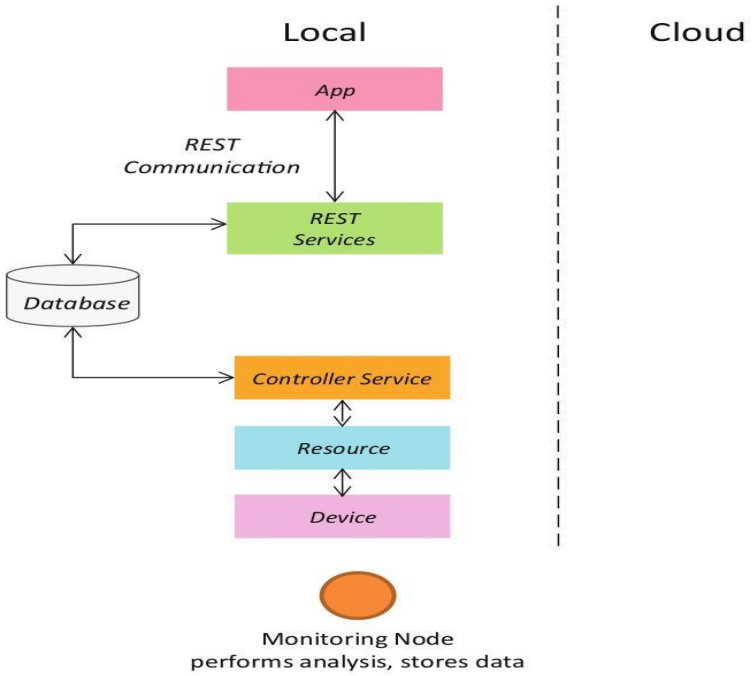


Figure 5.9 shows the deployment level of the home automation IoT system, which is level-1.

Functional View Specification,

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

The Functional Groups (FG) included in a Functional View include:

Device : The device FG contains devices for monitoring and control. In the home automation example, the device FG include a single board mini-computer, a light

Communication :

- The communication FG handles the communication for the IoT system. The communication FG includes the communication protocols that form the backbone of IoT systems and enable network connectivity.
- You learned about various link, network, transport and application layer protocols in Chapter-1. The communication FG also includes the communication APIs (such as REST and web Socket)that are used by the services and applications to exchange data over the network.
- In the home automation example the communication protocols include-802.11 (link layer), IPv4/IPv6 (network layer), TCP (transport layer), and HTTP (application layer). The communication API used in the home automation examples is a REST-based API.

Service :

- 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.
- In the home automation example, there are two REST services (mode and state service)and one native service (controller service).

Management :

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

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

Application : The application FG includes applications that provide an interface to the users to control and monitor various aspects of the IoT system. Applications also allow users to view the system status and the processed data.

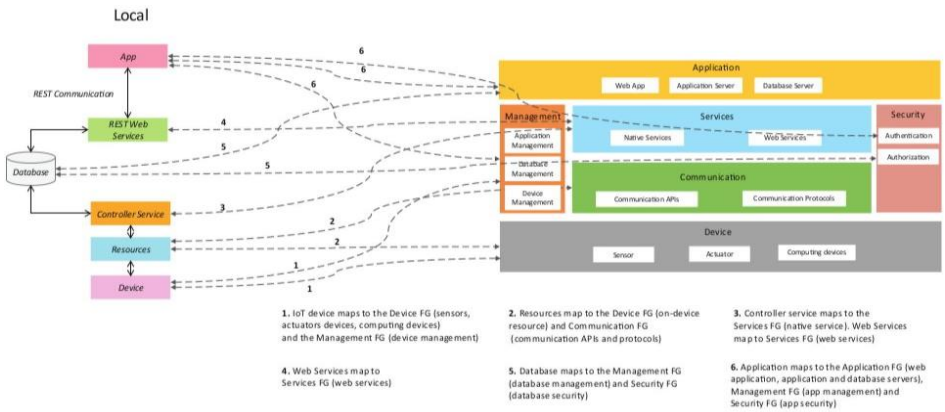


Figure 5.10 shows an example of mapping deployment level to functional groups for home automation IoT system.

- IoT device maps to the Device FG (device management). Resources map to the Device FG (on –device resource) and communication FG (communication APIs and protocols).Controller service maps to the Services FG (native service). Web services map to Services FG.
- Database maps to the Management FG (database management) and Security FG (database security). Application maps to the Application FG (web application, application and database servers), Management FG (app management) and Security FG (app security).

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, device options, application hosting options, etc.

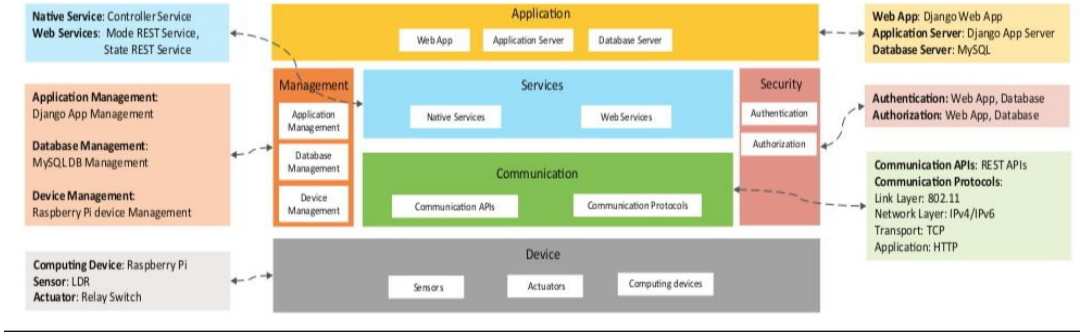


Figure 5.11 shows an example of mapping functional groups to operational view specifications for home automation IoT system.

Operational View specifications for the home automation example are as follows:

Devices:

- 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 device, implemented in Python and run as a native service.
2. Mode service – REST-ful web service, hosted on device, implemented with Django-REST Framework.
3. State service –REST -ful web service, hosted on 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

Authorization: Web App, Database

Management:

Application Management –Django App Management

Database Management – MySQL DB Management.

Device & Component Integration

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

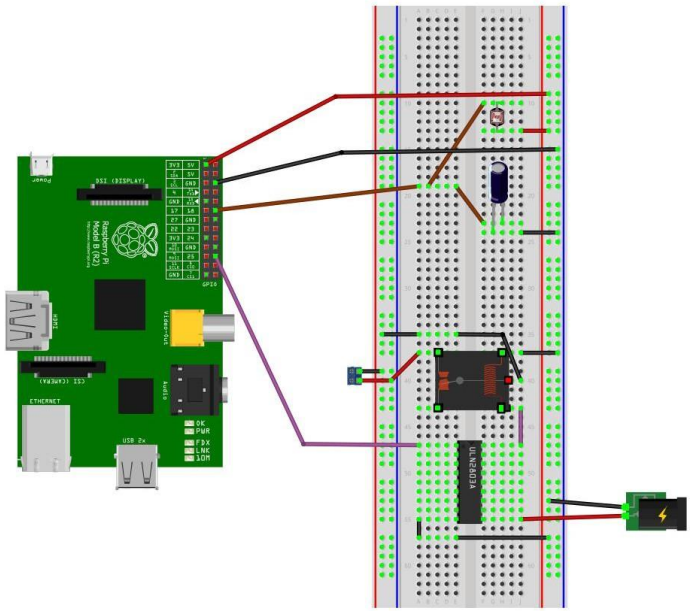


Figure 5.12 shows a schematic diagram of the home automation IoT system. The devices and components used in this example are Raspberry Pi mini computer, LDR sensor and relay switch actuator.

- A detailed description of Raspberry Pi board and how to interface sensors and actuators with the board is provided in later chapters.

Application Development

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

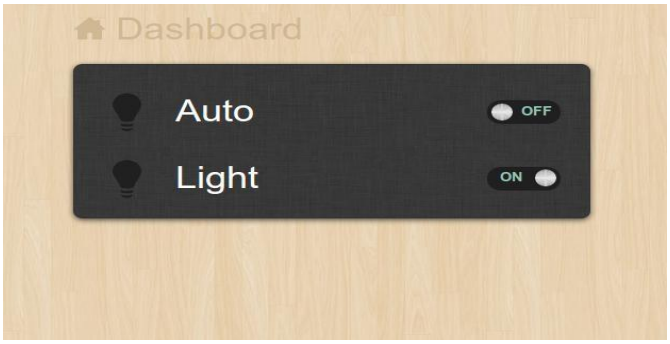


Figure 5.13 shows a screenshot of the home automation web application.

- The application has controls for the mode (auto on or auto off) and the light (on or off).
- In the auto mode, the IoT system controls the light appliance automatically based on the lighting conditions in the room.
- When auto mode is enabled the light control in the application is disabled and it reflects the current state of the light.
- When the auto mode is disabled, the light control is enabled and it is used for manually controlling the light.

UNIT - III: Prototyping Embedded Device with ARDUINO

Syllabus:

Sensors, Actuators, Embedded Computing Basics- Micro Controllers, System on Chips, Choosing your Platform, Arduino – Developing on the Arduino.

Before we get stuck into the ins and outs of microcontroller and embedded computer boards, let's address some of the electronics components that you might want to connect to them.

When it comes to thinking about the electronics, it's useful to split them into two main categories:

Sensors: Sensors are the ways of getting information *into* your device, finding out things about your surroundings.

Actuators: Actuators are the *outputs* for the device—the motors, lights, and so on, which let your device do something to the outside world.

- Within both categories, the electronic components can talk to the computer in a number of ways.

- The simplest is through digital I/O, which has only two states: a button can either be pressed or not; or an LED can be on or off. These states are usually connected via general-purpose input/output (GPIO) pins and map a digital 0 in the processor to 0 volts in the circuit and the digital 1 to a set voltage, usually the voltage that the processor is using to run (commonly 5V or 3.3V).

- Because computers are purely digital devices, you need a way to translate between the analogue voltages in the real world and the digital of the computer.

SENSORS

- Pushbuttons and switches, which are probably the simplest sensors, allow some user input.

- Potentiometers (both rotary and linear) and rotary encoders enable you to measure movement.

- Sensing the environment is another easy option.

- Light-dependent resistors (LDRs) allow measurement of ambient light levels, thermistors and other temperature sensors allow you to know how warm it is, and sensors to measure humidity or moisture levels are easy to build.

- Microphones obviously let you monitor sounds and audio, but piezo elements (used in certain types of microphones) can also be used to respond to vibration.

- Distance-sensing modules, which work by bouncing either an infrared or ultrasonic signal off objects, are readily available and as easy to interface to as a potentiometer.

ACTUATORS

- One of the simplest and yet most useful actuators is light, because it is easy to create electronically and gives an obvious output. Light-emitting diodes (LEDs) typically come in red and green but also white and other colours.

- RGB LEDs have a more complicated setup but allow you to mix the levels of red, green, and blue to make whatever colour of light you want. More complicated visual outputs also are available, such as LCD screens to display text or even simple graphics.

- More complicated again are motors. Stepper motors can be moved in *steps*, as the name implies. Usually, a fixed number of steps perform a full rotation.

- DC motors simply move at a given speed when told to. Both types of motor can be one-directional or move in both directions. Alternatively, if you want a motor that will turn to a given angle, you would need a servo.

- Although a servo is more controllable, it tends to have a shorter range of motion, often 180 or fewer degrees (whereas steppers and DC motors turn indefinitely).

- For all the kinds of motors that we've mentioned, you typically want to connect the motors to gears to alter the range of motion or convert circular movement to linear, and so on.

EMBEDDED COMPUTING BASICS

- Providing background is especially important because many of you may have little or no idea about what a microcontroller is.
- Although we've been talking about computing power getting cheaper and more powerful, you cannot just throw a bunch of PC components into something and call it an Internet of Things product.
- If you've ever opened up a desktop PC, you've seen that it's a collection of discrete modules to provide different aspects of functionality.
- It has a main motherboard with its processor, one or two smaller circuit boards providing the RAM, and a hard disk to provide the long-term storage.

- So, it has a lot of components, which provide a variety of general-purpose functionality and which all take up a corresponding chunk of physical space.

MICROCONTROLLERS

- Internet of Things devices take advantage of more tightly integrated and miniaturized solutions—from the most basic level of microcontrollers to more powerful system-on-chip (SoC) modules.

- These systems combine the processor, RAM, and storage onto a single chip, which means they are much more specialized, smaller than their PC equivalents, and also easier to build into a custom design.

- These microcontrollers are the engines of countless sensors and automated factory machinery. They are the last bastions of 8-bit computing in a world that's long since moved to 32-bit and beyond.

- The ubiquitous Arduino platform is based around Atmel's AVR ATmega family of microcontroller chips.
- The on-board inclusion of an assortment of GPIO pins and ADC circuitry means that microcontrollers are easy to wire up to all manner of sensors, lights, and motors.

- Because the devices using them are focused on performing one task, they can dispense with most of what we would term an operating system, resulting in a simpler and much slimmer code footprint than that of a SoC or PC solution.

SYSTEM-ON-CHIPS

- In between the low-end microcontroller and a full-blown PC sits the SoC (for example, the BeagleBone or the Raspberry Pi). Like the microcontroller, these SoCs combine a processor and a number of peripherals onto a single chip but usually have more capabilities.

- The processors usually range from a few hundred megahertz, nudging into the gigahertz for top-end solutions, and include RAM measured in megabytes rather than kilobytes.

- Storage for SoC modules tends not to be included on the chip, with SD cards being a popular solution.

- The greater capabilities of SoC mean that they need some sort of operating system to marshal their resources.

- A wide selection of embedded operating systems, both closed and open source, is available and from both specialized embedded providers and the big OS players, such as Microsoft and Linux.

- Again, as the price falls for increased computing power, the popularity and familiarity of options such as Linux are driving its wider adoption.

CHOOSING YOUR PLATFORM

- How to choose the *right* platform for your Internet of Things device is as easy a question to answer as working out the meaning of life.

- This isn't to say that it's an impossible question—more that there are almost as many
- answers as there are possible devices.

- The platform you choose depends on the particular blend of price, performance, and capabilities that suit what you're trying to achieve. And just because you settle on one solution, that doesn't mean somebody else wouldn't have chosen a completely different set of options to solve the same problem.

- Start by choosing a platform to prototype in. The following sections discuss some of the factors that you need to weigh—and possibly play off against each other—when deciding how to build your device.
    - Processor Speed
    - RAM

- Networking
- USB
- Power Consumption
- Interfacing with Sensors and Other Circuitry
- Physical Size and Form Factor

Processor Speed

- The processor speed, or clock speed, of your processor tells you how fast it can process the individual instructions in the machine code for the program it's running.

- Some processors may lack hardware support for floating-point calculations, so if the code involves a lot of complicated mathematics, a by-the-numbers slower processor with hardware floating-point support could be faster than a slightly higher performance processor without it.

- Generally, you will use the processor speed as one of a number of factors when weighing up similar systems. Microcontrollers tend to be clocked at speeds in the tens of MHz, whereas SoCs run at hundreds of MHz or possibly low GHz.

RAM

- RAM provides the working memory for the system. If you have more RAM, you may be able to do more things or have more flexibility over your choice of coding algorithm. If you're handling large datasets on the device, that could govern how much space you need.

- It is difficult to give exact guidelines to the amount of RAM you will need, as it will vary from project to project.

- However, microcontrollers with less than 1KB of RAM are unlikely to be of interest, and if you want to run standard encryption protocols, you will need at least 4KB, and preferably more.

- For SoC boards, particularly if you plan to run Linux as the operating system, we recommend at least 256MB.

Networking

- How your device connects to the rest of the world is a key consideration for Internet of Things products. Wired Ethernet is often the simplest for the user—generally plug and play—and cheapest, but it requires a physical cable.

- Wireless solutions obviously avoid that requirement but introduce a more complicated configuration.

- ***WiFi*** is the most widely deployed to provide an existing infrastructure for connections, but it can be more expensive and less optimized for power consumption than some of its competitors.

- Other short-range wireless can offer better power-consumption profiles or costs than WiFi but usually with the trade-off of lower bandwidth.

- ***ZigBee*** is one such technology, aimed particularly at sensor networks and scenarios such as home automation.

- The recent ***Bluetooth*** LE protocol (also known as Bluetooth 4.0) has a very low power-consumption profile similar to ZigBee's and could see more rapid adoption due to its inclusion into standard Bluetooth chips included in phones and laptops.

- There is, of course, the existing Bluetooth standard as another possible choice. And at the boringbut-very-cheap end of the market sit long-established options such as RFM12B which operate in the 434 MHz radio spectrum, rather than the 2.4 GHz range of the other options we've discussed.

- For remote or outdoor deployment, little beats simply using the mobile phone networks. For low-bandwidth, higher-latency communication, you could use something as basic as SMS; for higher data rates, you will use the same data connections, like 3G, as a Smartphone.

USB

- If your device can rely on a more powerful computer being nearby, tethering to it via USB can be an easy way to provide both power and networking.
- You can buy some of the microcontrollers in versions which include support for USB, so choosing one of them reduces the need for an extra chip in your circuit.

- Instead of the microcontroller presenting itself as a device, some can also act as the USB "host".

- This configuration lets you connect items that would normally expect to be connected to a computer—devices such as phones, for example, using the Android ADK, additional storage capacity, or WiFi dongles.

- Devices such as WiFi dongles often depend on additional software on the host system, such as networking stacks, and so are better suited to the more computer-like option of SoC.

Power Consumption

- Faster processors are often more power hungry than slower ones. For devices which might be portable or rely on an unconventional power supply (batteries, solar power) depending on where they are installed, power consumption may be an issue.

- Even with access to mains electricity, the power consumption may be something to consider because lower consumption may be a desirable feature.

- However, processors may have a minimal power-consumption sleep mode.
- This mode may allow you to use a faster processor to quickly perform operations and then return to low-power sleep.

- Therefore, a more powerful processor may *not* be a disadvantage even in a low-power embedded device.

Interfacing with Sensors and Other Circuitry

- In addition to talking to the Internet, your device needs to interact with something else—either sensors to gather data about its environment; or motors, LEDs, screens, and so on, to provide output.

- You could connect to the circuitry through some sort of peripheral bus—SPI and I2C being common ones—or through ADC or DAC modules to read or write varying voltages; or through generic GPIO pins, which provide digital on/off inputs or outputs.

- Different microcontrollers or SoC solutions offer different mixtures of these interfaces in differing numbers.

Physical Size and Form Factor

- The continual improvement in manufacturing techniques for silicon chips means that we've long passed the point where the limiting factor in the size of a chip is the amount of space required for all the transistors and other components that make up the circuitry on the silicon.

- Nowadays, the size is governed by the number of connections it needs to make to the surrounding components on the PCB.

- With the traditional through-hole design, most commonly used for homemade circuits, the legs of the chip are usually spaced at 0.1" intervals.

- Even if your chip has relatively few connections to the surrounding circuit—16 pins is nothing for such a chip—you will end up with over 1.5" (~4cm) for the perimeter of your chip.

- More complex chips can easily run to over a hundred connections; finding room for a chip with a 10" (25cm) perimeter might be a bit tricky!

- The limit to the size that each connection can be reduced to is then governed by the capabilities and tolerances of your manufacturing process.

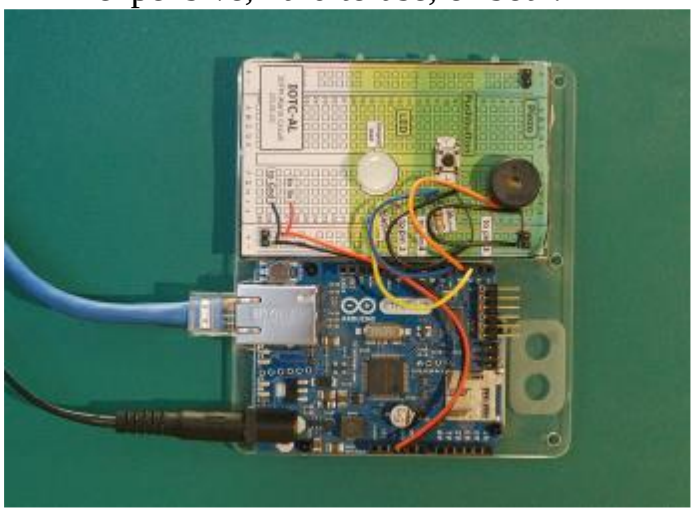
- Some surface-mount designs are big enough for home-etched PCBs and can be hand-soldered.
- Others require professionally produced PCBs and accurate pick-and-place machines to locate them correctly.

- All three chips pictured in the following figure provide identical functionality because they are all AVR ATmega328 microcontrollers. The one on the left is the through-hole package, mounted here in a socket so that it can be swapped out without soldering. The two others are surface mount, in two different packages, showing the reduction in size but at the expense of ease of soldering.

Fig: Through-hole versus surface-mount ATmega328 chips.

ARDUINO

- Without a doubt, the poster child for the Internet of Things, and physical computing in general, is the Arduino.
- These days the Arduino project covers a number of microcontroller boards, but its birth was in Ivrea in Northern Italy in 2005.
- A group from the Interaction Design Institute Ivrea (IDII) wanted a board for its design students to use to build interactive projects.
- An assortment of boards was around at that time, but they tended to be expensive, hard to use, or both.



An Arduino Ethernet board, plugged in, wired up to a circuit and ready for use.

- The "standard" Arduino board has gone through a number of iterations: Arduino NG, Diecimila, Duemilanove, and Uno.
- The Uno features an ATmega328 microcontroller and a USB socket for connection to a computer.
- It has 32KB of storage and 2KB of RAM, but don't let those meagre amounts of memory put you off; you can achieve a surprising amount despite the limitations.
- The Uno also provides 14 GPIO pins (of which 6 can also provide PWM output) and 6 10-bit resolution ADC pins.
- The ATmega's serial port is made available through both the IO pins, and, via an additional chip, the USB connector.

DEVELOPING ON THE ARDUINO

- More than just specs, the experience of working with a board may be the most important factor, at least at the prototyping stage. As previously mentioned, the Arduino is optimized for simplicity, and this is evident from the way it is packaged for use.

Developing on the Arduino we will consider below points.

- Integrated Development Environment(IDE)
- Pushing Code
- Operating System
- Language
- Debugging

Integrated Development Environment(IDE)

- You usually develop against the Arduino using the integrated development environment (IDE) that the team supply at http://arduino.cc.

- Although this is a fully functional IDE, based on the one used for the Processing anguage (http://processing.org/), it is very imple to use. Most Arduino projects consist of a single file of code, so you can think of the IDE mostly as a simple file editor.

- The controls that you use the most are those to check the code (by compiling it) or to push code to the board.

Pushing Code

- Connecting to the board should be relatively straightforward via a USB cable. Sometimes you might have issues with the drivers (especially on some versions of Windows) or with permissions on the USB port (some Linux packages for drivers don't add you to the dialout group), but they are usually swiftly resolved once and for good.
- After this, you need to choose the correct serial port (which you can discover from system logs or select by trial and error) and the board type (from the appropriate menus, you may need to look carefully at the labelling on your board and its CPU to determine which option to select).
- When your setup is correct, the process of pushing code is generally simple:

- first, the code is checked and compiled, with any compilation errors reported to you.
- If the code compiles successfully, it gets transferred to the Arduino and stored in its flash memory. At this point, the Arduino reboots and starts running the new code.

Operating System

- The Arduino doesn't, by default, run an OS as such, only the bootloader, which simplifies the code-pushing process described previously.
- When you switch on the board, it simply runs the code that you have compiled until the board is switched off again (or the code crashes).
- It is, however, possible to upload an OS to the Arduino, usually a lightweight real-time operating system (RTOS) such as FreeRTOS/DuinOS.
- The main advantage of one of these operating systems is their built-in support for multitasking.
- However, for many purposes, you can achieve reasonable results with a simpler task-dispatching library.
- If you dislike the simple life, it is even possible to compile code without using the IDE but by using the toolset for the Arduino's chip—for example, for all the boards until the recent ARM-based Due, the avr-gcc toolset.
- The avr-gcc toolset (www.nongnu.org/avr-libc/) is the collection of programs that let you compile code to run on the AVR chips used by the rest of the Arduino boards and flash the resultant executable to the chip. It is used by the Arduino IDE behind the scenes but can be used directly, as well.

Language

- The language usually used for Arduino is a slightly modified dialect of C++ derived from the Wiring platform.
- It includes some libraries used to read and write data from the I/O pins provided on the Arduino and to do some basic handling for "interrupts" (a way of doing multitasking, at a very low level).
- This variant of C++ tries to be forgiving about the ordering of code; for example, it allows you to call functions before they are defined.
- This alteration is just a nicety, but it is useful to be able to order things in a way that the code is easy to read and maintain, given that it tends to be written in a single file.

The code needs to provide only two routines:

- setup(): This routine is run once when the board first boots. You could use it to set the modes of I/O pins to input or output or to prepare a data structure which will be used throughout the program.
- loop(): This routine is run repeatedly in a tight loop while the Arduino is switched on. Typically, you might check some input, do some calculation on it, and perhaps do some output in response.

To avoid getting into the details of programming languages in this chapter, we just compare a simple example across all the boards—blinking a single LED:

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:

int led = 13;

// the setup routine runs once when you press reset:

void setup() {

// initialize the digital pin as an output.

pinMode(led, OUTPUT);

}

// the loop routine runs over and over again forever:

void loop() {

digitalWrite(led, HIGH); // turn the LED on

delay(1000); // wait for a second

digitalWrite(led, LOW); // turn the LED off

delay(1000); // wait for a second

}
```

- Reading through this code, you'll see that the setup() function does very little; it just sets up that pin number 13 is the one we're going to control (because it is wired up to an LED).
- Then, in loop(), the LED is turned on and then off, with a delay of a second between each flick of the (electronic) switch.
- With the way that the Arduino environment works, whenever it reaches the end of one cycle—on; wait a second; off; wait a second—and drops out of the loop() function, it simply calls loop() again to repeat the process.

Debugging

- Because C++ is a compiled language, a fair number of errors, such as bad syntax or failure to declare variables, are caught at compilation time.
- Because this happens on your computer, you have ample opportunity to get detailed and possibly helpful information from the compiler about what the problem is.
- Although you need some debugging experience to be able to identify certain compiler errors, others, like this one, are relatively easy to understand:

Blink.cpp: In function 'void loop()':Blink:21:

error:'digitalWritee' was not declared in this scope

On line 21, in the function **loop(),** we deliberately misspelled the call to **digitalWrite.**

- When the code is pushed to the Arduino, the rules of the game change, however. Because the Arduino isn't generally connected to a screen, it is hard for it to tell you when something goes wrong. Even if the code compiled
- successfully, certain errors still happen. An error could be raised that can't be handled, such as a division by zero, or trying to access the tenth element of a 9-element list. Or perhaps your program leaks memory and eventually just stops working. Or (and worse) a programming error might make the code continue to work dutifully but give entirely the wrong results.
- If Bubblino stops blowing bubbles, how can we distinguish between the following cases?

  Nobody has mentioned us on Twitter.

  The Twitter search API has stopped working.

  Bubblino can't connect to the Internet.

- Bubblino has crashed due to a programming error.
- Bubblino is working, but the motor of the bubble machine has failed.
- Bubblino is powered off.