Ut,

Ultrasonic Sensor:

An ultrasonic sensor is measures the distance of respective object by sending the wave of specific frequency. This sound wave is reflected after the collision with respective object and this wave is received by the ultrasonic receiver. Distance is measured by calculating sending and receiving time of this sound wave.

Distance = Sound speed x time taken /2.

Working

It consists of set of ultrasonic transmitter and receiver which are operated at same frequency. When anything or object comes into the area of covered circuit then its frequency sound reflected to receiver and alarm is triggered. This sensor circuit is very sensitive and it could be reset automatically or still in triggered until it is reset manually.

Types

- **Ultrasonic Proximity Sensors.**

A special type of sonic transducer is used in this sensor for alternate transmission and reception of sound wave. This sonic transducer emits the sonic waves which are reflected by an object and after this emission, this sensor switched in to receive mode.

- **Ultrasonic 2 Point Proximity ~~Sensors~~ Switches**

It consists of 2 points for switching, therefore it is called 2-point proximity switches. It is almost similar with standard sensor only differ the 2-touch set up key and this function is called Tech-in function. Its switches Sde1 & Sde2 could be easily programmed within the sensing range with the help of built in Tech-in button.

- **Ultrasonic Retro reflective Sensors:**

The operation of ultrasonic retro reflective sensor is similar with ultrasonic proximity sensor.

Only difference, in this sensor the distance between sensor to reflector is measured by measuring the propagation time. In this sensor, the stationary object could be used as a reflector and sensing distance (SD) could be adjust by adjusting the potentiometer resistance with in ultrasonic sensor.

- **Ultrasonic Through beam Sensors**

Unlike proximity and retro-reflective sensors these sensors separate the emitter and the receiver into separate housings. The emitter sends a continous signal, which is then picked up by the receiver. When an object disrupts the sonic beam, the receiver reacts and triggers an output.

Arduino

→ Arduino is an open-source prototyping platform used for building electronics project

→ It consists of a both a physical programmable circuit board and a software, or IDE that runs on your computer, where you can write and upload the code to the physical board.

→ It The Arduino board adapting to the new needs and challenges differentiating it from simple 8 bit boards to products for IoT applications, 3D printing
→ It can interact with buttons, LEDs, motors, speakers, GPS units, cameras, the internet and even your smartphone or your TV.

Features

1) IDE runs on every ~~platform~~ operating system (Mac, Linux and Windows).
2) It's based on a strong and well supported backened, the open source gcc toolchain and wrapped in Java, so bugs can be found and fixed.
3) There is a big community of smart people using and working on the IDE to keep it going strong.
4) There are numerous object wrapped libraries to do complex things.
5) The code runs directly on bare metal, with a well tested and up understood compiler.

6) It became a huge hit because of its analog to digital input.
7) It is easily affordable and there is no comprise with low quality board.

Arduino Variants

- Arduino Uno
- " NaNo
- " Lilypad
- " Mega 2560
- RedBoard.

Raspberry PI

→ Raspberry Pi is a low cost, credit card sized computer that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python.

→ It is a project initiated by Eben Upton and developed in UK by Raspberry Pi foundation in 2009.

→ It supports several programming languages like scratch, Python, Node.JS, C, C++, Java, Perl, HTML5, Javascript, JQuery etc.

- It is capable of doing everything that a desktop computer can do.

Features & Benefits

1) It's simple, open and easy to maintain and energy efficient.
2) Small in size and at the same time has all the functions of a laptop and a desktop.
3) It consumes very less power, only about five to seven watts of electricity.
4) Systems are noise free and is a perfect adaptive technology where it is able to display images or play videos at 1080p HD resolution.
5) It is very ~~com~~ affordable compared to branded computers that are commercially available.
6) It is armed with built in HDMI capable graphics.
7) It can be overclocked if there are performance problems with the application used.
8) The ability to store an SD card makes it easy to swap with other SD cards.

Lite Os

- It is a lightweight, open source IoT device and smartphone OS from the Chinese smartphone manufacturer Huawei.
- It is designed to have a low footprint, which saves space and reduces the load of the OS on the device.
- It supports smartphones, wearables, intelligent manufacturing applications, smart homes and Internet of Vehicles (IoV).
- It simplifies IoT device development and connectivity while focusing on enhancing user experience.
- The smallest Kernel (6KB) on the market offers fast-start and low power consumption features.

RIoT Os

- Open source Embedded O.S.
- It is designed for networked and memory constrained systems.
- Targeting on low power and IoT devices.
- Lightweight, limited processing time, small main memory.

- First developed by FU Berlin, INRIA and the HAW Hamburg in 1999.
- Written in ~~ASC~~ ANSI C.
- Based on a ~~microcontroller~~ microKernel architecture.

Features :-

- Modularity
  → Customization of the system's configuration.
  → Minimized Kernel's size.
  → Effects of bugs is limited in the module itself.
- Tickless Scheduler
  → It does not have a timer that fires periodically in order to emulate concurrent execution by switching threads continuously.
- Straight forward interrupt handler.
- Support various hardware vendors.
- Reliability and real time features.
  → zero latency interrupt handlers.
  → minimum context switching times with thread priorities.
- Support for full multithreading and C++.
- Full support for internet protocols on resource constrained system.

Contiki Os

→ It is an open source O.S for the IoT.
→ It connects tiny low-cost, low power microcontrollers to the Internet and provides powerful low power internet communication.
→ It supports full standard IPV6 and IPV4 along with the recent low power wireless standards: 6lowpan, RPL, CoAP.
→ It uses a minimalist design while still packing the common tools of modern Os.

Features

1) It comes with a rich set of features that are dev programmer friendly.
2) It can fit into 10 KB of RAM and 100KB of ROM.
3) It can run on devices such as 8051 SoC to ARM powered devices.
4) Ports are available on other platforms such as Arduino and Atmel.
5) It comes with much documentation apart from well documented code.

OS. Functions include:

1. Process management
2. Memory management
3. Communication management
4. File management.

Applications

1) There are several app. that come packaged as part of contiki like small web browser, Web server, calc., shell, email client, ftp etc.
2) Developers can find tools like Cooja simulator for app. development.
3) Power sensitive applications.

Tiny Os

- It is a free open source operating system.
- Designed for wireless sensor networks.
- Tiny os began as a collaboration between Univery of California, Berkely and Intel Research.
- An embedded operating system written in nesC language.
- It features a component based architecture.

Features

- Completely non blocking
- Programs are built out of software components.
- Tasks are non preemptive and run in FIFO order.
- Tiny OS code is statically linked.
- Power efficient as it makes the sensors sleep as soon as possible.
- Component based architecture allows frequent changes while still keeping the size of code minimum.
- Event based execution model means no user / Kernel boundary and hence supports high concurrency.

Models

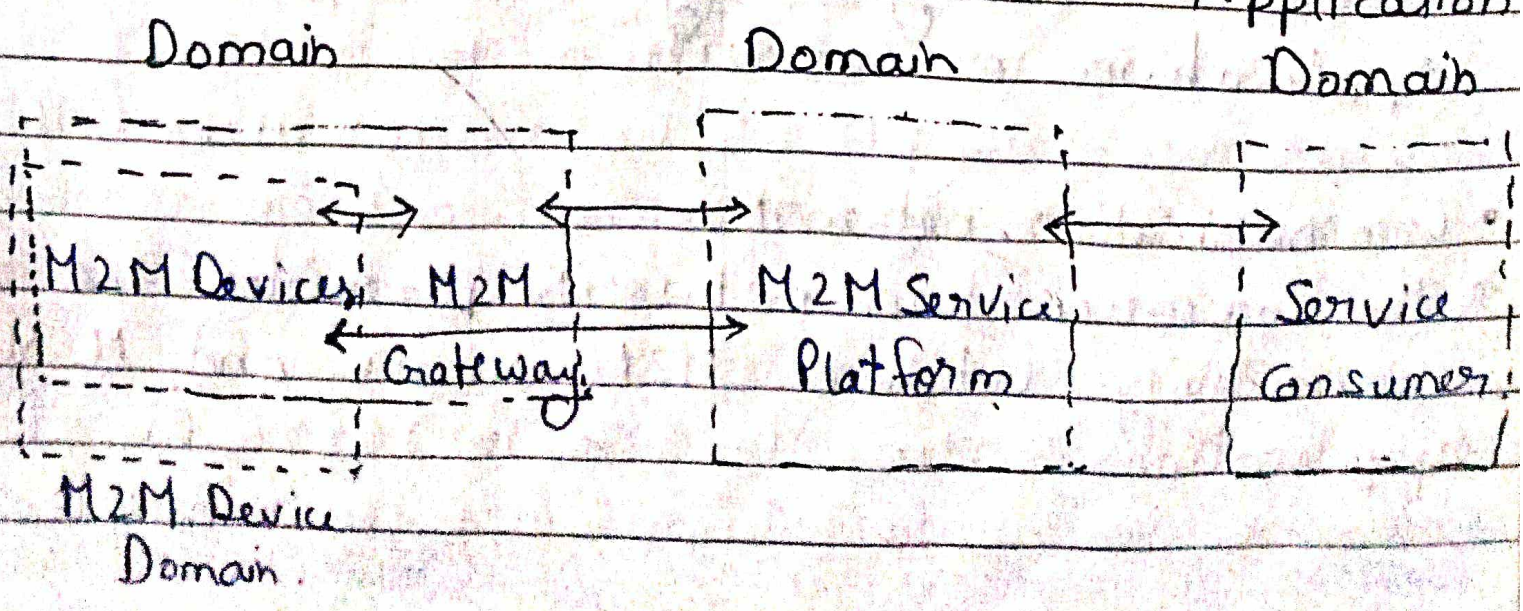
1) Data
2) Thread
3) Programming
4) Component
5) Network

M2M

- Machine to Machine (M2M) refers to the communication or exchange of data between to two or more machines without human interfacing or interaction.
- Communication in M2M may be wired or wireless systems.
- M2M uses a device such as sensor, RFID, meter, etc. to capture an 'events' like temp. inventory level, etc. that translates the captured event into meaningful information.

M2M System Architecture

1. M2M area networks
2. Communication networks
3. Application domains.
4. M2M gateways.



- M2M area networks

→ M2M network area consists of machines or M2M nodes which communicate with each other. The M2M nodes embedded with hardware modules such as sensors actuators and communication devices.

→ M2M uses communication protocol such as Zigbee, Bluetooth, Power line communication (PLC) etc.

→ M2M nodes communicate with in one network it can't communicate with external network node.

- M2M Gateways.

→ The Gateway module provides control and localization services for data collection

→ M2M communication network serves as infrastructure for realizing communication between M2M gateway & M2M end user application or server.

- Communication networks.

→ The communication network provides the connectivity between M2M nodes and M2M applications.

→ It uses wired or wireless network such as LAN, LTE, WiMAX, satellite communication etc.

• Application domains.

→ It contains the middleware layer where data goes through various app. services and is used by the specific business processing engines.

→ Applications may either target at end users, such as user of a specific M2M solution, or at other application providers to offer more refined building blocks by which they can build more sophisticated M2M solutions & services.

Difference between IoT and M2M

M2M	IoT
• Machine to machine communication and completely base hardware based	• Machine to machine, M to Sensors, or humans to machines and software based.
• It is a point to point communication and uses non IP protocols.	• It uses IP networks & protocols as the communication is multipoint.

• These devices don't rely on Internet. • Data can be stored locally.	• Devices required internet connections. • Data can be stored locally and also in cloud.
• Limited integration option devices must have corresponding communication standards	• Unlimited integration option, but requires a solutions that can manage all the communication.
• Unidirectional comm.	• Bidirectional comm.

Similarities between IoT & M2M

Both provide remote ~~access~~ access to machine data and both exchange info among machines without human intervention

Software Defined Networking (SDN)

• SDN is defined as the physical separation of the networking architecture of control plane from the data plane, and centralizes the network controller.

Basic Concepts of SDN

→ Separate control logic from hardware switches

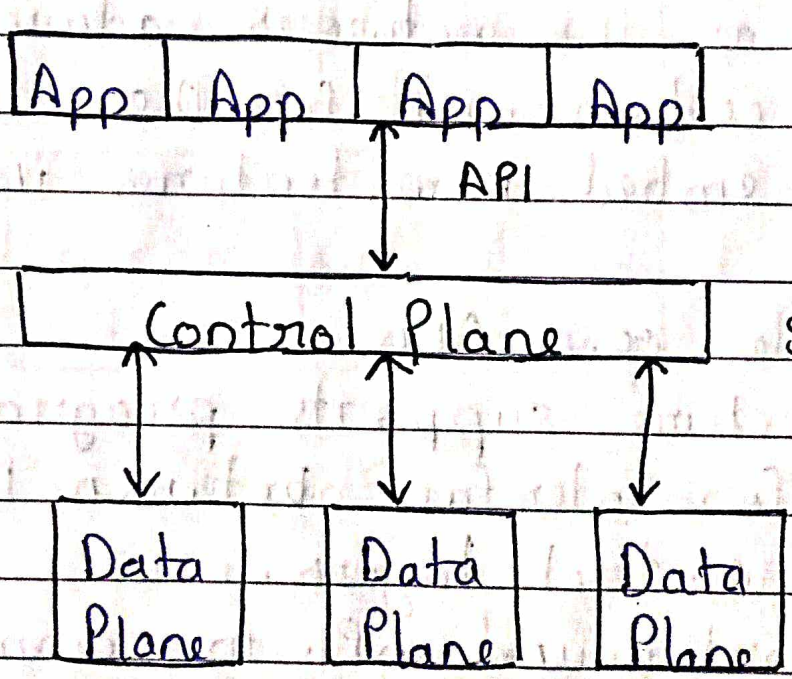
→ Define the control logic in a centralized

manner.

→ Control the entire network including individual switches.

→ Communication between the app., control, and data planes are done through APIs.

SDN Architecture



Key Components of SDN.

- Centralized Network Controller.
- Programable open APIs.
- Standard communication interface (Open Flow).

1) Centralized Network Controller.

→ With separated control plains data plane and centralized network controller, the network administrator can rapidly configure the network.

→ SDN application can be deployed through programmable APIs. which speeds up innovation as the network administrator no longer need to wait for the device vendors to embed new features in hardware.

2) Programmable open APIs

→ SDN architecture supports programmable open APIs for interface between the SDN app. and control layers.

→ SDN uses northbound APIs to communicate with the applications.

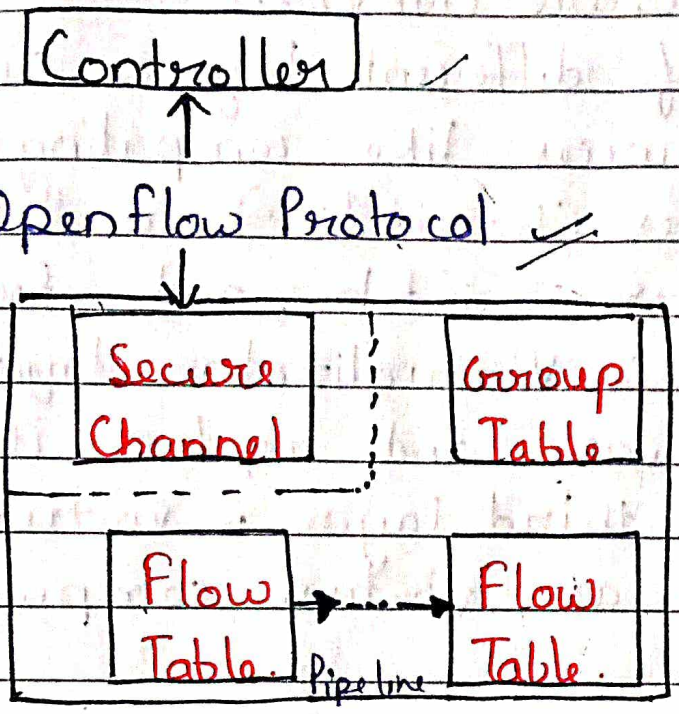
3) Open Flow

→ Standard communication interface between control layer and infrastructure layer.

→ It uses southbound APIs to relay information to the switches and routers below.

→ The controller manages the switch via open flow switch protocol where controller can add

update and delete flow entries in flow table.



OpenFlow Switch.

Network function Virtualization (NFV)

- Network functions virtualization (NFV) is the concept of replacing dedicated network appliances such as routers and firewalls with software running on general purpose CPUs or virtual machines, operating on standard servers.
- NFV provideas the infras infrastructure on which SDN can run. NFV and SDN are mutually beneficial to each other but not dependent.

Key elements of NFV

1. NFV infrastructure (NFVI):-

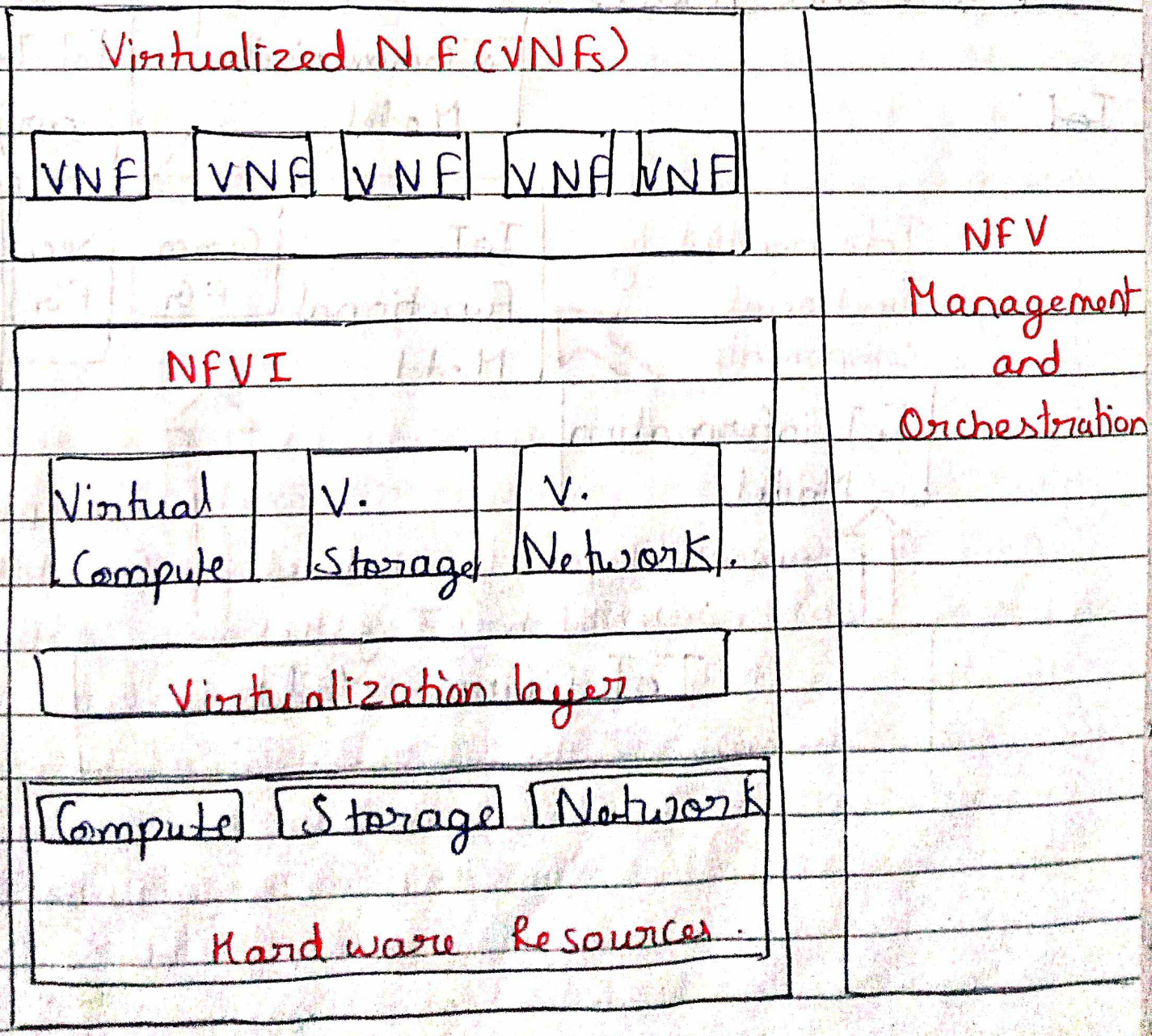
- NFVI consists of different layers such as hardware resources like computing resources, storage resources (hard-disc), and network resources (routers, switch, and firewalls)
- Second layer is Virtualization layer which separates hardware and replaces it with software and third layer is virtualized resources such as virtual compute, network and storage.

2. Virtualized network function (VNF)
VNF is a software implementation is a network function which is capable of running over the NFV infrastruce (NFVI).
Ex- vFirewall, vRouters.

3. NFV management and orchestration :-
It has three parts.
→ Virtualized infrastructure manager:-
It controls and manages network functions with NFVI resources and monitors virtualization layer.

→ VNf manager:- It manages the life cycle of VNf such as initialize, update, query, scale, terminate. etc.

→ Orchestrator:-
It manages the life cycle of network services which includes policy management performance measurement and monitoring.

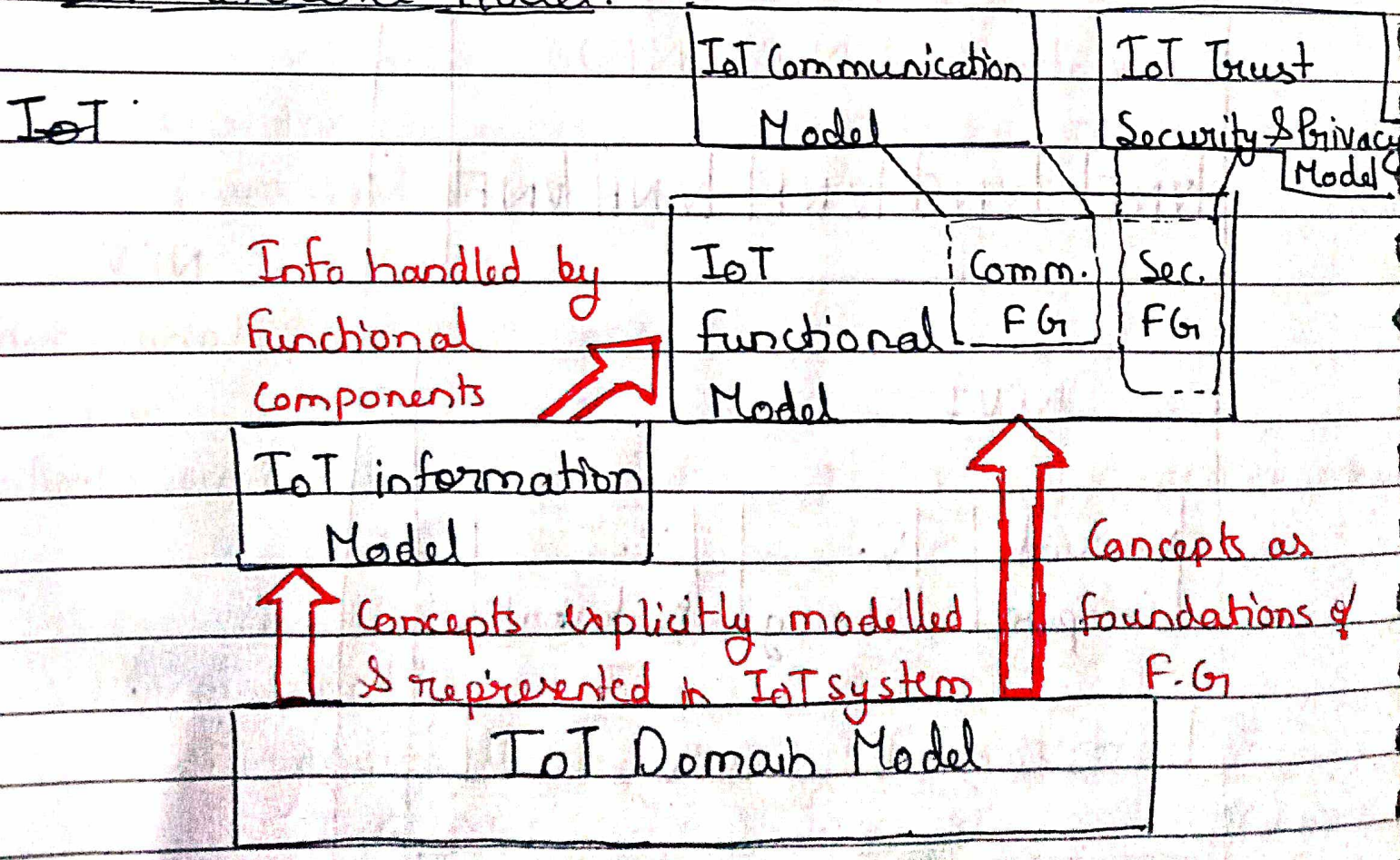


IoT Reference model and Architecture.

- An ARM consists of two main parts:
  1) a Reference model
  2) a Reference Architecture.

- A reference model describes the domain using a number of sub models.

1) IoT Reference Model.



IoT domain model

→ It captures the basic attributes of the main concepts and the relationship between these concepts.

→ Abstraction level of the IoT Domain model has been chosen in such a way that its concepts are independent of specific technologies and use cases.

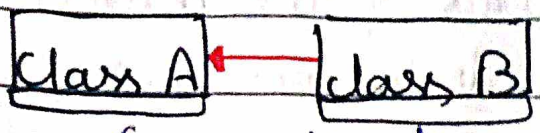
→ The idea is that these concepts are not expected to change much over the next decades or longer.

Three Kinds of Device types for the IoT Domain Model.

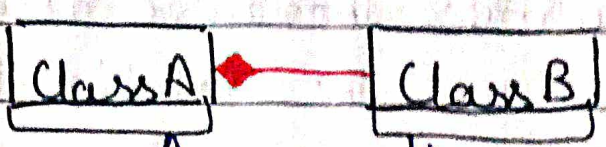
1. Sensors
2. Actuators.
3. Tags. - In general, identify the Physical Entity that they are attached to. It can be both devices or physical entities but not both, as the domain model shows.

Example: Tag as a device - Radio Frequency ID.
Tag as a P.E - Paper printed immutable barcode or Quick Response (QR) code.

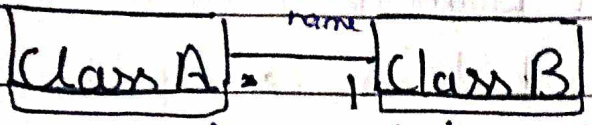
Model notation and Semantics



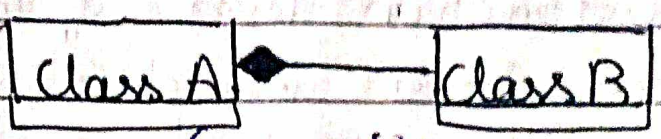
Generalization



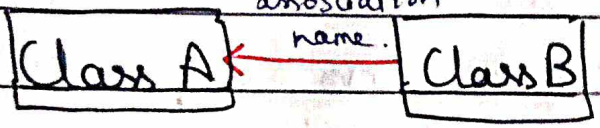
Aggregation



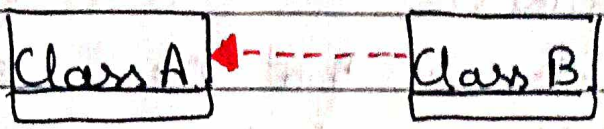
Association



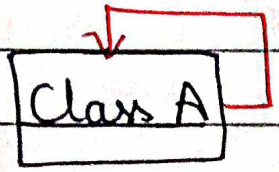
Composition



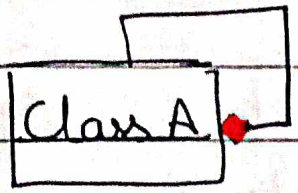
Directed Association



Realization.



Reflexive D.A



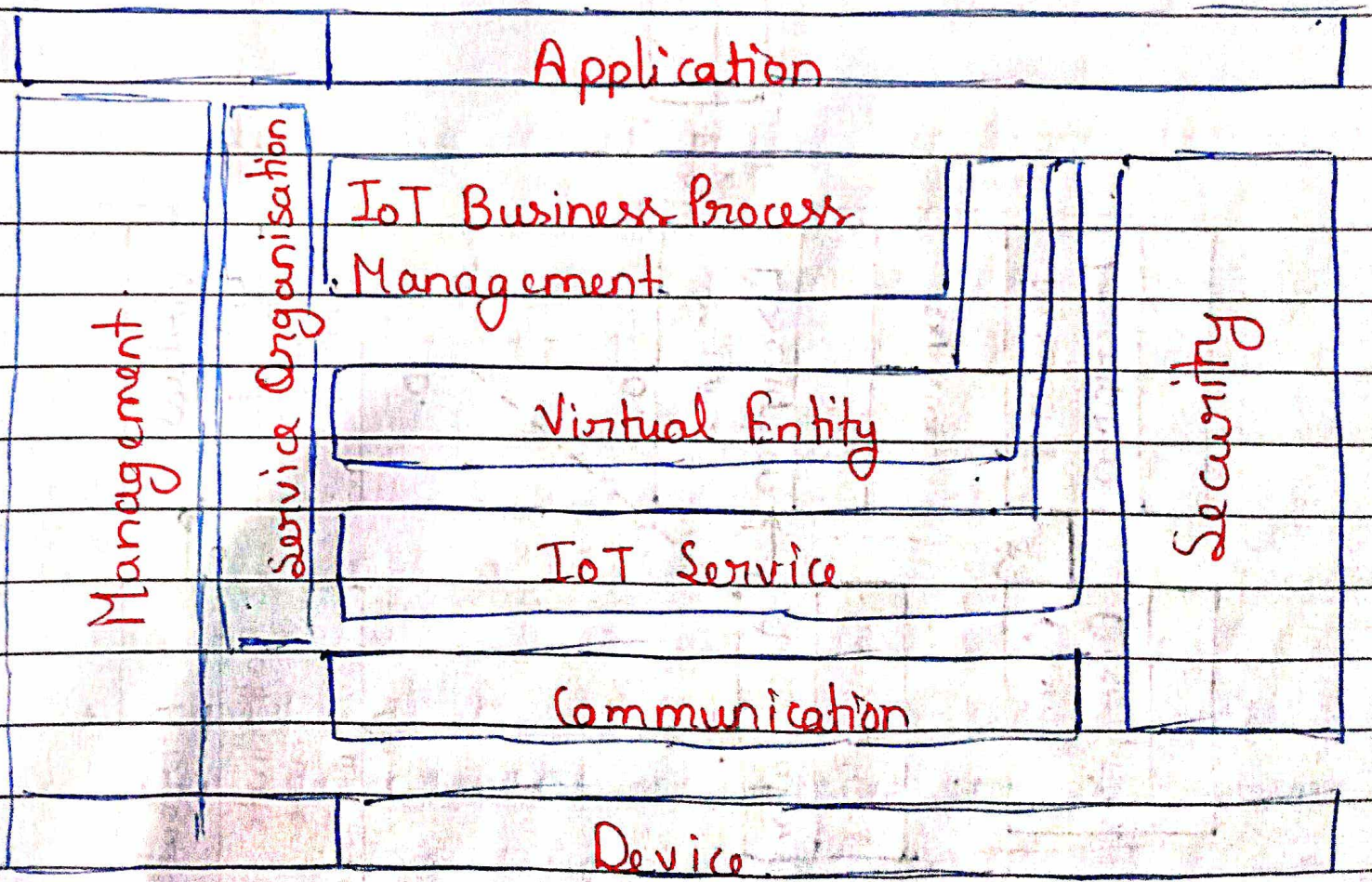
Reflexive Aggregation

IoT Information Model

Virtual entity in the IoT Domain Model is the 'thing' in the IoT, the IoT information model captures the details of a Virtual entity centric model. Similar to the IoT domain model, the IoT information Model is presented using Unified Modelling Language (UML) diagrams.

Functional model

>It aims at describing mainly the F.Gr. and their interaction with the ARM, while the Functional View of a Reference Architecture describes the functional components of a FGr, interfaces, and interactions between the components. The functional View is typically derived from the functional Model in conjunction with high-level requirements.



- Device functional Group.

→ The Device FGr contains all the possible functionalit hosted by the physical Devices that are used.

for increment the Physical Entities.
→ The Device functionality includes sensing actuation, processing, storage, and identification components, the sophistication of which depends on the Device capabilities.

- Communication functional group.

→ Comm. F Gr. consists abstracts all the possible communication mechanisms used by the relevant Devices in an actual system in order to transfer information to the digital world components or other Devices.

- IoT Service F.Gr.

It corresponds mainly to the Service class from the IoT Domain model, and contains single IoT services exposed by Resources hosted on Devices or in the network.

- Virtual Entity F.Gr.

→ It corresponds to the virtual entity class in the IoT Domain model
→ It contains the necessary functionality to manage associations between virtual Entities with themselves as well as between V.E and related IoT Services.

IoT Service Organization functional group.

→ Its purpose is to host all functional components that support the composition and of IoT and Virtual Entity services.

IoT Process Management F.G

→ It is a collection of functionalities that allows smooth integration of IoT related services with the business process.

Management F.G.

It includes the necessary functions for enabling fault and performance monitoring of the system, configuration for enabling the system to be flexible to changing user demands;

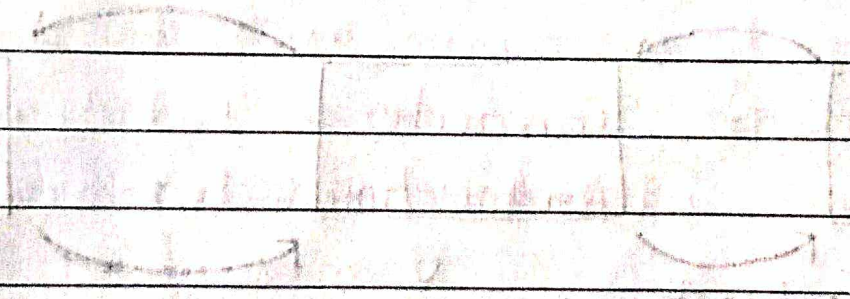
Security F.G.

It contains the functions that ensure the secure operation of the system as well as the manag ent of privacy. It components contains components for Authentication of Users, Authorisation of access to services by Users, secure communication between entities of the system such as Devices, Services, App.

Application F.Gr. - -

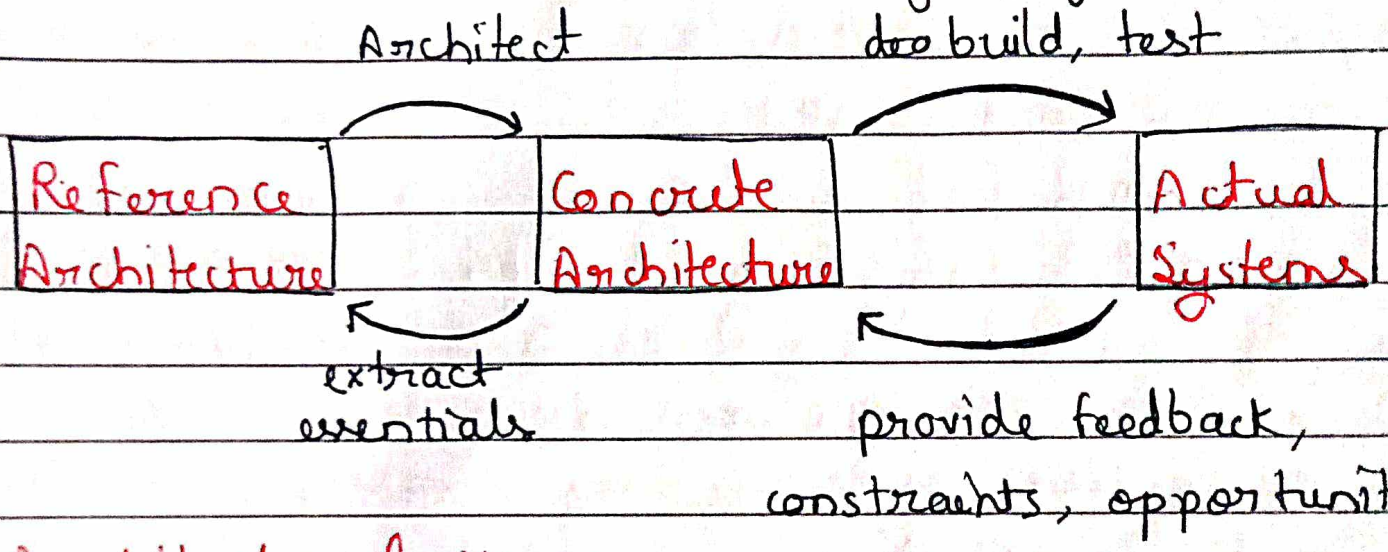
Communication Model.

It aims at defining the main communication paradigms for connecting elements, as defined in the IoT Domain Model.



IoT Reference Architecture

→ It is a starting point for generating concrete architectures and actual systems.
→ A reference architecture, serves as a guide for one or more concrete system architects.



- Architectural Views-

→ It is presented as set of architectural views. Views are useful for reducing the complexity of the Reference.
→ Views are used during the design and implementation phase of a concrete system architecture.
→ A view is composed of viewpoints which is a collection of patterns, templates and conventions for constructing one type of view.

- **Functional view**

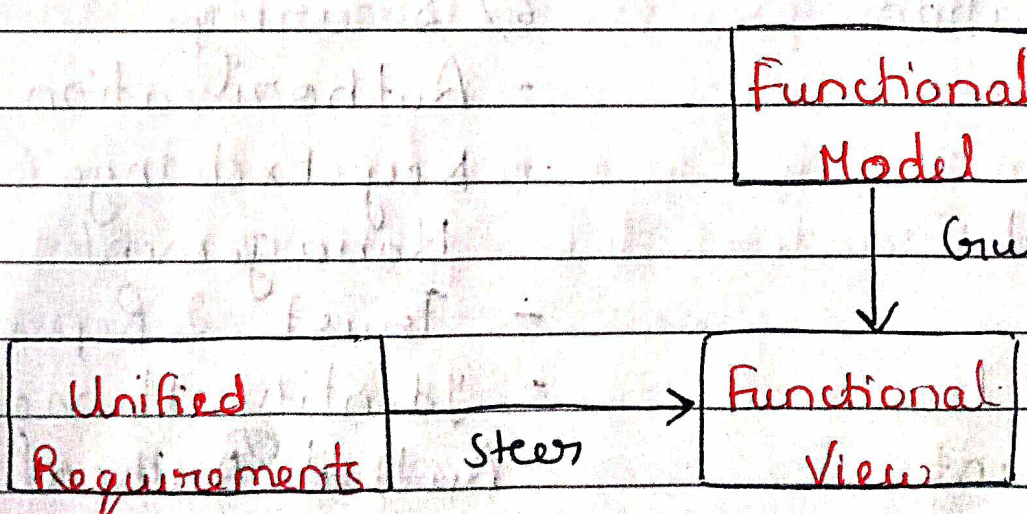
→ Describes & what the system does and its main functions.

→ The Unified Requirements are mapped to the diff. Functionality Groups of the IoT Functional Model.

→ Next, clusters of requirements of similar functionality are formed and a Functional Component for these requirements defined.

→ Thus the view points used for constructing IoT functional View are :-

1) Unified Requirements
2) IoT Functional Model



Functional view Process diag.

→ Once all functional components are defined the default function set, system use cases, sequence charts and interface definitions are made.

→ Following are the functional components for each of the functionality groups.

1) IoT Process Management
- Process Modelling
- Process Execution

2) Service Organisation
- Service Composition
- Service Orchestration
- Service Choreography

3) Virtual Entity
- VE Resolution
- VE & IoT Service Monitoring
- VE Service

4) IoT Service
- IoT Service
- IoT Service Resolution

5) Communication
- Hop to Hop "
- Network "
- End to End "

6) Security
- Authorization
- Key Exchange & Management
- Trust & Reputation
- Identity Management
- ~~Authent~~ Authentication

7) Management
- Configuration
- Fault
- Reporting
- Member
- State

Information View.

→ It describes the information that the system handles and the components that handle these information.

→ The pieces of information handled by an IoT system complying to an ARM such as the IoT A are the following:

- Virtual Entity context information i.e. attributes (simple or complex) as represented by parts of the IoT information model.
- IoT Service Output itself is another ~~impl~~ important part of information generated by an IoT system.
- Virtual Entity descriptions and its association with other Virtual entity.
- Resource descriptions - type of resources, identity, associated services and devices.
- Device descriptions like device capabilities
- Descriptions of Composed services like the model of how a complex service is composed of simpler

services.

Process

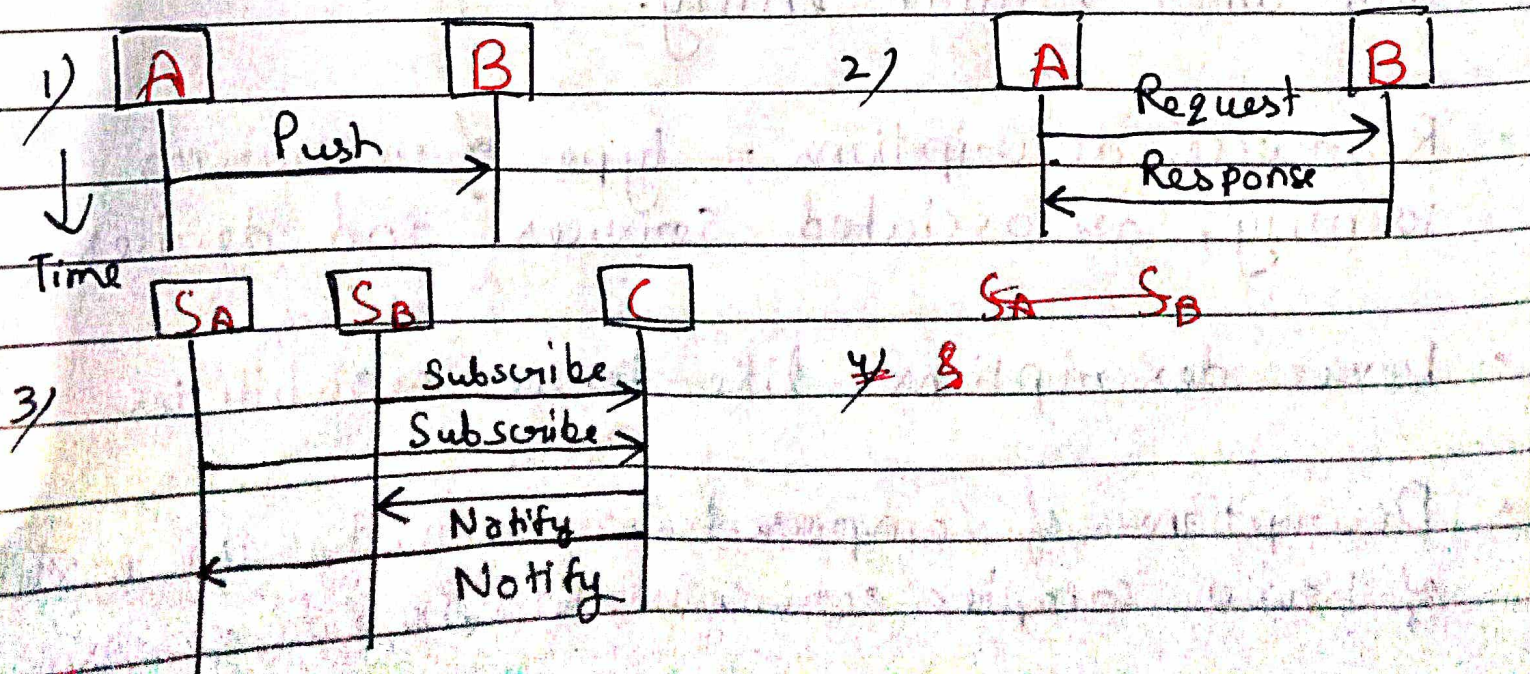
- IoT Business Model describes the steps of a business process utilizing other IoT related services.

- Management information such as state information from operational FC.

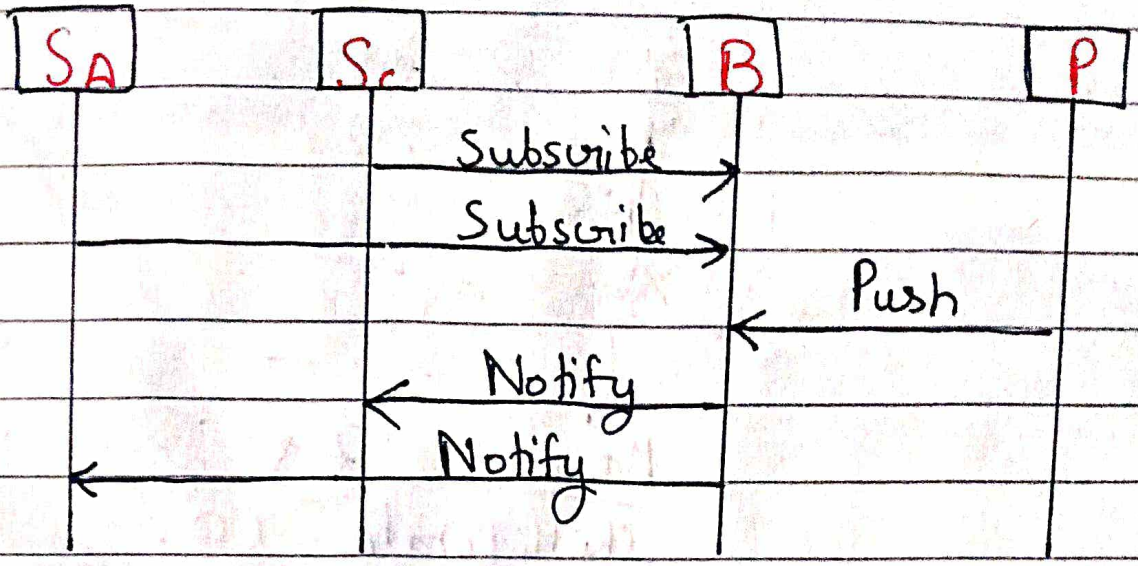
Information handling

→ The presentation of information handling in an IoT system assumes that FCs exchange and process information.

→ The exchange of information between FCs follows the interaction patterns below



4)



- Deployment and Operational View.

→ Description of the main real world components of the system such as devices, network routers, servers etc.

→ It aims at providing users of the IoT Reference model with a set of guidelines to drive them through the different design choices that they have to face while designing the actual implementation of their services.

→ It will discuss how to move from the service & description and the identification of the different functional elements to the selection among the many available technologies in the IoT to build up the overall networking behaviour for the deployment.

Representational State Transfer (REST)

→ It is a type of software architecture that was designed to ensure interopera-bility between different internet computer systems.

→ It works by putting in place very strict constraints for the development of web services.

→ Services that can request and edit text version of a web resource via a predefined set of operations that are uniform and stateless.

Architectural Constraints.

- Client-Server

→ Separation of concerns is the principle behind the client-server constraints.

→ By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.

- **Stateless**

→ This constraint states that the Server does not store any session data.
→ It means that all the information to understand a request is contained within the request.
→ Improves scalability.
→ Session state is therefore kept entirely on the client.

- **Cacheable**

→ It requires that every response should include whether a response can be cacheable or not.
→ For subsequent requests, the client can retrieve from its cache, to need to send request to the Server.
→ Reduces network latency, improves efficiency, scalability.

- **Uniform interface**

→ Uniform interface is the key differentiator between REST & Rf Non REST APIs.
→ There are 4 elements of Uniform Interface constraint.
  - Identification of Resources (typically by an URL)
  - ~~Manage~~ Manipulation of Resources through representations
  - Self-descriptive messages for each request.
  - HATEOS (Hypermedia As the Engine of app. State)

→ Promotes generality as all components interact in the same way.

- **Layered System**

→ In or It allows an architecture to be composed of hierarchical layers.
→ Each layer doesn't Know anything beyond the immediate layer.
→ Disadvantage is latency.

- **Code on Demand**

→ It allows client functionality to be extended by downloading and executing code in the form of applets or scripts.
→ Allowing features to be downloaded after deployment improves system extensibility.

Architectural properties

- Scalability allowing the support of large numbers of components and interactions among components.
- Simplicity of a uniform interface.
- Modafibility of components to meet changing needs.

- Visibility of communication between components by service agents.
- Portability of components by moving program code with the data.
- Reliability in the resistance to failure at the system level in the presence of failures within components, connectors or data.

Uniform Resource Identifiers

- A URI is a sequence of characters that identifies a logical or physical resource. URI are specified in the internet engineering
- It describes the mechanism used to access resources, the computers on which resources are housed and the names of the resources on each computer.

There are two types of URIs

- URL (Uniform Resource Locator)

→ It is the mechanism used by browsers to retrieve any published resource on the web.
→ It is the address of a given unique resource on the web.
→ It is handled by the webserver.

- **Uniform Resource Name (URN)**

→ URNs are globally unique persistent identifiers assigned within defined namespac so they will be available for a long period of time, even after the resource which they identify ceases to exist or becomes unavailab

Challenges in IoT

- **Security challenges in IoT**

1. Lack of Encryption.

- Although encryption is a great way to prevent hackers from accessing data, it is also one of the leading IoT security challenges.
- These devices lack the storage and processing capabilities that would be found on a traditional computer.
- The result is an increase in attacks where hackers can easily manipulate the algorithms that were designed for protection.

2. ~~Out dated legacy Security.~~

A

2. Insufficient testing and updating.

- With the increase in number of IoT devices, IoT manufacturers are more eager to produce and deliver their devices as fast as they can, without giving security too much of athought.
- Most of these devices and IoT products don't get enough testing and updates and are prone to hackers and other security issues.

3. Brute-forcing and the issue of default passwords

- Weak credentials and login details leave nearly all IoT devices vulnerable to password hacking and brute forcing. Mirai Malware
- Any company that used factory default credentials on their devices is placing both their business and its assets and the customer and their valuable information at risk of being susceptible to a brute force attack.

4. IoT malware and ransomware.

- Increases with increase in devices.
- Ransomware uses encryption to effectively lock out users from ~~serveral~~ several devices and platforms and steal user's valuable data info.
- For example - A hacker can hijack a computer camera and take pictures..

By using malware access point, the hackers can demand ransom to unlock the device and return the data.

5. IoT botnets aiming at cryptocurrency.

- IoT botnets workers can manipulate data privacy, which could be a massive risk for an open crypto-market. The exact value and creation of cryptocurrencies could face danger from mal-intentioned hackers.
- IoT The blockchain companies are trying to boost security. Blockchain technology itself is not particularly ~~v~~ vulnerable but the app development process is.

~~Des~~

- Design Challenges in IoT.

1. Battery life is a limitation.

Issues in packaging and integration of small size chip with low weight and lesser power consumption.

2. Increased cost and time to market

- Embedded systems are tightly constrained by cost.
- The need originates to derive better approaches when designing the IoT devices in order to handle the cost modelling or cost optimality with digital electronic components.
- Designers also need to solve the design time problem and bring embedded devices at the right time to the market.

3. Security of the system

- Systems have to be designed and implemented to be robust and reliable and have to be secure with cryptographic algorithms and security procedures.
- It involves different approaches to secure all the components of embedded systems from prototype to deployment.

~~Deveto~~

- Development challenges in IoT.

1. Connectivity

- It is the foremost concern while connecting devices applications and cloud platforms.

- Connected devices that provide useful front-end information is extremely valuable. But poor connectivity becomes a challenge where IoT sensors are required to monitor, process data and supply information.

2. Cross Platform Compatibility (~~Hardware &~~ Devices).

- IoT applications must be developed keeping in mind the technological changes of the future.
- Its development requires a balance of hardware and software functions.
- It is a challenge for IoT application developers to ensure that device and IoT platform delivers the best performance despite heavy OS, device updates and bug fixings.

3. Data Collection and Processing.

- In IoT development, data play an important role but what is more crucial here is the processing or usefulness of stored data.
- Along with security and privacy, development teams need to ensure that they plan well for the way data is collected, stored or

processed within an environment.

4. Lack of skill set

- All of the development challenges above can only be handled if there is a proper skilled resource working on the IoT application development
- A right talent will always get you past the major challenges and will be an important IoT application development asset.