UNIT-II

IoT Architecture Overview

IoT can be classified into a four or five-layered architecture which gives you a complete overview of how it works in real life. The various components of the architecture include the following:

Four-layered architecture: this includes media/device layer, network layer, service and application support layer, and application layer.

Five-layered architecture: this includes perception layer, network layer, middleware layer, application layer, and business layer.

Functions of Each Layer

Sensor/Perception layer: This layer comprises of wireless devices, sensors, and radio frequency identification (RFID) tags that are used for collecting and transmitting raw data such as the temperature, moisture, etc. which is passed on to the next layer.

Network layer: This layer is largely responsible for routing data to the next layer in the hierarchy with the help of network protocols. It uses wired and wireless technologies for data transmission.

Middleware layer: This layer comprises of databases that store the information passed on by the lower layers where it performs information processing and uses the results to make further decisions.

Service and application support layer: This layer involve business process modeling andexecution as well as IoT service monitoring and resolution.

Application layer: It consists of application user interface and deals with various applicationssuch as home automation, electronic health monitoring, etc.

Business layer: this layer determines the future or further actions required based on the dataprovided by the lower layers.

Building an IoT ArchitectureBUILDING BLOCKS of IoT

Four things form basic building blocks of the IoT system –sensors, processors, gateways, applications. Each of these nodes has to have its own characteristics in order to form a useful IoT system.

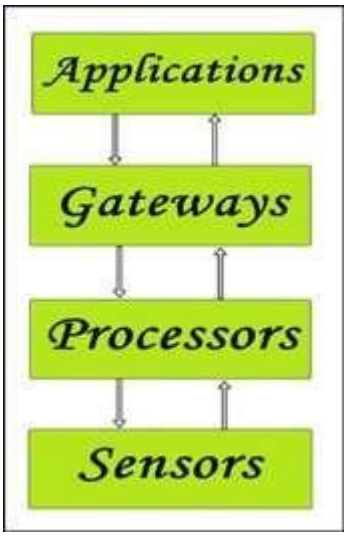


Fig 2.1 Simplified block diagram of the basic building blocks of the IoTSensors:

These form the front end of the IoT devices. These are the so-called "Things" of the system. Their main purpose is to collect data from its surroundings (sensors) or give out data to its surrounding (actuators). These have to be uniquely identifiable devices with a unique IP address so that they can be easily identifiable over a large network. These have to be active in nature which means that they should be able to collect real-time data. These can eitherwork on their own (autonomous in nature) or can be made to work by the user depending on their needs (user-controlled).

Examples of sensors are gas sensor, water quality sensor, moisture sensor, etc.**Processors:**

Processors are the brain of the IoT system. Their main function is to process the data capturedby the sensors and process them so as to extract the valuable data from the enormous amount

of raw data collected. In a word, we can say that it gives intelligence to the data. Processors mostly work on real-time basis and can be easily controlled by applications. These are also responsible for securing the data – that is performing encryption and decryption of data. Embedded hardware devices, microcontroller, etc are the ones that process the data because they have processors attached to it.

Gateways:

Gateways are responsible for routing the processed data and send it to proper locations for its (data) proper utilization. In other words, we can say that gateway helps in to and fro communication of the data. It provides network connectivity to the data. Network connectivity is essential for any IoT system to communicate.

LAN, WAN, PAN, etc are examples of network gateways.**Applications:**

Applications form another end of an IoT system. Applications are essential for proper utilization of all the data collected. These cloud-based applications which are responsible for rendering the effective meaning to the data collected. Applications are controlled by usersand are a delivery point of particular services. Examples of applications are home automation apps, security systems, industrial control hub, etc.

Main design principles of IoT

1. **Do your research**

When designing IoT-enabled products, designers might make the mistake of forgetting why customers value these products in the first place. That's why it's a good idea to think aboutthe value an IoT offering should deliver at the initial phase of your design. When getting intoIoT design, you're not building products anymore. You're building services and experiences that improve people's lives. That's why in-depth qualitative research is the key to figuring out how you can do that. Assume the perspective of your customers to understand what they need and how your IoT implementation can solve their pain points. Research your target audience deeply to see what their existing experiences are and what they wish was different about them.

2. Concentrate on value

Early adopters are eager to try out new technologies. But the rest of your customer basemight be reluctant to put a new solution to use. They may not feel confident with it and are likely to be cautious about using it. If you want your IoT solution to become widely adopted, you need to focus on the actual tangible value it's going to deliver to your target audience. What is the real end-user value of your solution? What might be the barriers to adopting new technology? How can your solution address them specifically? Note that the features the early tech adopters might find valuable might turn out to be completely uninteresting for the majority of users. That's why you need to carefully plan which features to include and inwhat order, always concentrating on the actual value they provide.

3. Don't forget about the bigger picture

One characteristic trait of IoT solutions is that they typically include multiple devices that come with different capabilities and consist of both digital and physical touchpoints. Your solution might also be delivered to users in cooperation with service providers. That's why it's not enough to design a single touchpoint well. Instead, you need to take the bigger picture into account and treat your IoT system holistically. Delineate the role of every device and service. Develop a conceptual model of how users will perceive and understand the system. All the parts of your system need to work seamlessly together. Only then you'll be able to create a meaningful experience for your end-users.

4. Remember about the security

Don't forget that IoT solutions aren't purely digital. They're located in the real-world context, and the consequences of their actions might be serious if something goes wrong. At the same time, building trust in IoT solutions should be one of your main design drivers. Make sure that every interaction with your product builds consumer trust rather than breakingit. In practice, it means that you should understand all the possible error situations that maybe related to the context of its use. Then try to design your product in a way to prevent them. If error situations occur, make sure that the user is informed appropriately and provided with help. Also, consider data security and privacy as a key aspect of your implementation. Users need to feel that their data is safe, and objects located in their workspaces or home can't be hacked. That's why quality assurance and testing the system in the real-world context are so important.

5. Build with the context in mind

And speaking of context, it pays to remember that IoT solutions are located at the intersectionof the physical and digital world. The commands you give through digital interfaces produce real-world effects. Unlike digital commands, these actions may not be easily undone. In a real-world context, many unexpected things may happen. That's why you need to make sure that the design of your solution enables users to feel safe and in control at all times. The context itself is a crucial consideration during IoT design. Depending on the physical context of your solution, you might have different goals in mind. For example, you might want to minimize user distraction or design devices that will be resistant to the changing weather conditions. The social context is an important factor, as well. Don't forget that the devices you design for workspaces or homes will be used by multiple users.

6. Make good use of prototypes

IoT solutions are often difficult to upgrade. Once the user places the connected objectsomewhere, it might be hard to replace it with a new version – especially if the user would have to pay for the upgrade. Even the software within the object might be hard to update because of security and privacy reasons. Make sure that your design practices help to avoid costly hardware iterations. Get your solution right from the start. From the design perspective, it means that prototyping and rapid iteration will become critical in the early stages of the project.

Standards consideration for IoT

Alliances have been formed by many domestic and multinational companies to agree on common standards and technology for the IoT. However, no universal body has been formed

yet. While organizations such as IEEE, Internet Engineering Task Force (IETF), ITU-T, OneM2M, 3GPP, etc., are active at international level, Telecommunication Standards Development Society, India (TSDSI), Global ICT Standardization Forum for India (GISFI), Bureau of Indian Standards (BIS), Korean Agency for Technology and Standards (KATS), and so on, are active at national level and European Telecommunications Standards Institute (ETSI) in the regional level for standardization.

IoT Architecture

Introduction

The Internet of Things (IoT) has seen an increasing interest in adaptive frameworks and architectural designs to promote the correlation between IoT devices and IoT systems. This is because IoT systems are designed to be categorized across diverse application domains and geographical locations. It, therefore, creates extensive dependencies acrossdomains, platforms and services. Considering this interdependency between IoT devices and IoT systems, an intelligent, connection-aware framework has become a necessity, this is where IoT architecture comes into play! Imagine a variety of smart IoT systems from sensors and actuators to internet getaways and Data Acquisition Systems all under the centralized control of one “brain”! The brain here can be referred to as the IoT architecture, whose effectiveness and applicability directly correlate with the quality of its building blocks. The way a system interacts and the different functions an IoT device performs are various approaches to IoT architecture. Since we can call the architecture the brain, it’s also possible to say that the key causes of poor integration in IoT systems are the shortage of intelligent, connection-aware architecture to support interaction in IoT systems.

An IoT architecture is the system of numerous elements that range from sensors, protocols, actuators, to cloud services, and layers. Besides, devices and sensors the Internet of Things (IoT) architecture layers are distinguished to track the consistency of a system through protocols and gateways. Different architectures have been proposed by researchers and we can all agree that there is no single consensus on architecture for IoT. The most basic architecture is a three-layer architecture.

State-of-the-art

The IoT can be considered both a dynamic and global networked infrastructure that manages self-configuring objects in a highly intelligent way. This, in turn, allows the interconnection of IoT devices that share their information to create new applications and services which can improve human lives. Originally, the concept of the IoT was first introduced by Kevin Ashton, who is the founder of MIT auto-identification centre

in 1999. Ashton has said, “The Internet of Things has the potential to change the world, just as the Internet did. Maybe even more so”. Later, the IoT was officially presented by the International Telecommunication Union (ITU) in 2005. The IoT has many definitions suggested by many organizations and researchers. However, the definition provided by ITU in 2012 is the most common. It stated: “a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies”. In addition, Guillemin and Friess in have suggested one of the simplest definitions that describe the IoT in a smooth manner. It stated: “The Internet of Things allows people and things to be connected Anytime, Anyplace, with anything and anyone, ideally using any path/network and any service”. Several definitions were suggested by many researchers describing the IoT system from different perspectives but the important thing that majority or researchers have agreed on is the IoT is created for a better world for all the human beings. The IoT is a promising technology that starts to grow significantly. There were already more objects/things connected to the Internet than people from 2008. Predictions are made that in the future; the number of Internet-connected devices will reach or even exceed 50 billion. In addition, the IoT becomes the most massive device market that enables companies to save billions of dollars. It has added $1.7 trillion in value to the global economy in 2019. This involves hardware, software, management services, installation costs, and economic value from realized IoT efficiencies. Nowadays, the IoT notion has evolved to include the perception of realizing a global infrastructure of interconnected networks of physical and virtual objects. The huge technological development has expanded the idea of the IoT to involve other technologies such as Cloud computing and Wireless Sensor Networks (WSNs). The IoT has become able to connect both humans and things anywhere, and anytime, ideally using any path/network. The IoT has become one of the interesting topics to many researchers. According to Google, the number of IoT journal and conference papers has almost doubled from 2004 to 2010. From 2010, the IoT articles are dramatically increased to reach about 985 articles in 2015.

Architecture Reference ModelIntroduction

A reference model is a division of functionality together with data flow between the pieces. A reference model is a standard decomposition of a known problem into parts that cooperatively solve the problem. Arising from experience, reference models are a characteristic of mature domains. Can you name the standard parts of a compiler or adatabase management system? Can you explain in broad terms how the parts work together toaccomplish their collective purpose? If so, it is because you have been taught a reference model of these applications.

A reference architecture is a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them. Whereas a reference model divides the functionality, a reference architecture is the mapping of that functionality onto a system decomposition. The mapping may be, but by no means necessarily is, one to one. A software element may implement part of a function or several functions. Reference models, architectural patterns, and reference architectures arenot architectures; they are useful concepts that capture elements of an architecture. Each is the outcome of early design decisions. The relationship among these design elements is shown in Figure 1

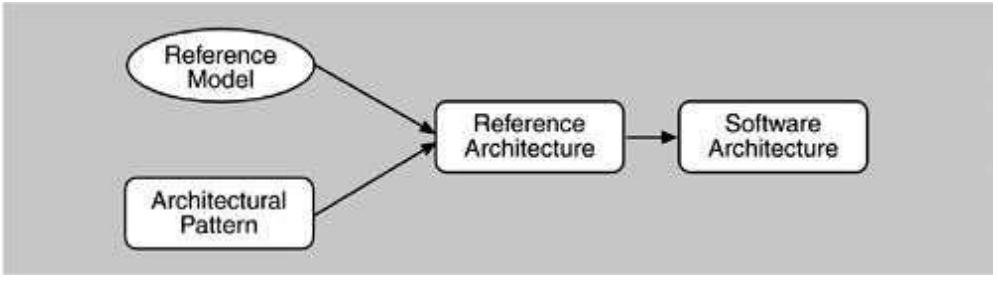


Fig 2.2 The relationships of reference models, architectural patterns, reference architectures,and software architectures.

IoT Reference Architecture

The reference architecture consists of a set of components. Layers can be realized by means of specific technologies, and we will discuss options for realizing each component. There are also some cross-cutting/vertical layers such as access/identity management.

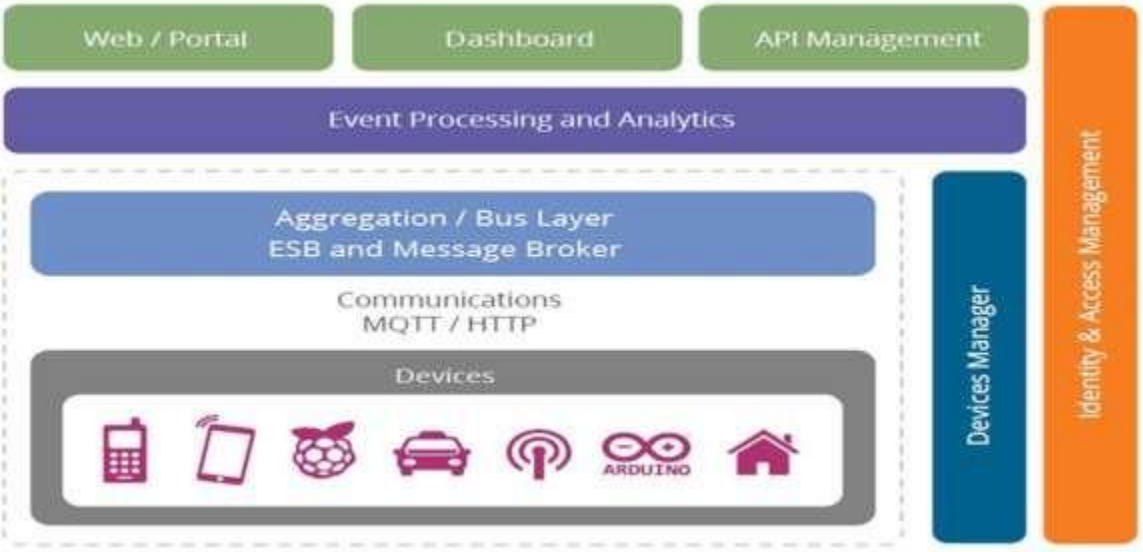


Fig 2.3 IoT Reference Architecture

The layers are

- Client/external communications - Web/Portal, Dashboard, APIs
- Event processing and analytics (including data storage)
- Aggregation/bus layer – ESB and message broker
- Relevant transports - MQTT/HTTP/XMPP/CoAP/AMQP, etc.
- Devices

The cross-cutting layers are

- Device manager
- Identity and access managements

THE DEVICE LAYER

The bottom layer of the architecture is the device layer. Devices can be of various types, but in order to be

considered as IoT devices, they must have some communications that either indirectly or directly attaches to the Internet. Examples of direct connections are

- Arduino with Arduino Ethernet connection
- Arduino Yun with a Wi-Fi connection
- Raspberry Pi connected via Ethernet or Wi-Fi
- Intel Galileo connected via Ethernet or

Wi-Fi Examples of indirectly connected devices include

- ZigBee devices connected via a ZigBee gateway
- Bluetooth or Bluetooth Low Energy devices connecting via a mobile phone
- Devices communicating via low power radios to a

Raspberry Pi There are many more such examples of each type.

Each device typically needs an identity. The identity may be one of the following:

- A unique identifier (UUID) burnt into the device (typically part of the System-on Chip, orprovided by a secondary chip) •

A UUID provided by the radio subsystem (e.g. Bluetooth identifier, Wi-Fi MAC address)

- An OAuth2 Refresh/Bearer Token (this may be in addition to one of the above)
- An identifier stored in nonvolatile memory such as EEPROM

For the reference architecture we recommend that every device has a UUID (preferably an unchangeable ID provided by the core hardware) as well as an OAuth2 Refresh and Bearer token stored in EEPROM. The specification is based on HTTP; however, (as we will discuss in the communications section) the reference architecture also supports these flows over MQTT.

COMMUNICATIONS LAYER

The communication layer supports the connectivity of the devices. There are multiple potential protocols for communication between the devices and the cloud.

The most well-known three potential protocols are

- HTTP/HTTPS (and RESTful approaches on those)
- MQTT 3.1/3.1.1(Message Queuing Telemetry Transport)
- Constrained application protocol (CoAP)

HTTP is well known, and there are many libraries that support it. Because it is a simple text- based protocol, many small devices such as 8-bit controllers can only partially support the protocol – for example enough code to POST or GET a resource. The larger 32-bit based devices can utilize full HTTP client libraries that properly implement the whole protocol. There are several protocols optimized for IoT use. The two best known are MQTT6 and CoAP7. MQTT was invented in 1999 to solve issues in embedded systems and SCADA. It has been through some iterations and the current version (3.1.1) is undergoing standardizationin the OASIS MQTT Technical Committee8. MQTT is a publish-subscribe messaging systembased on a broker model. The protocol

has a very small overhead (as little as 2 bytes per message), and was designed to support lossy and intermittently connected networks. MQTT was designed to flow over TCP. In addition, there is an associated specification designed for ZigBee-style networks called MQTT-SN (Sensor Networks). CoAP is a protocol from the IETF that is designed to provide a RESTful application protocol modeled on HTTP semantics, but with a much smaller footprint and a binary rather than a text- based approach. CoAP is a more traditional client-server approach rather than a brokered approach. CoAP is designed to be used over UDP. For the reference architecture we have opted to select MQTT as the preferred device communication protocol, with HTTP as an alternative option.

The reasons to select MQTT and not CoAP at this stage are

- Better adoption and wider library support for MQTT;
- Simplified bridging into existing event collection and event processing systems; and
- Simpler connectivity over firewalls and NAT networks

However, both protocols have specific strengths (and weaknesses) and so there will be some situations where CoAP may be preferable and could be swapped in. In order to support MQTT we need to have an MQTT broker in the architecture as well as device libraries. We will discuss this with regard to security and scalability later. One important aspect with IoT devices is not just for the device to send data to the cloud/ server, but also the reverse. This is one of the benefits of the MQTT specification: because it is a brokered model, clients connectan outbound connection to the broker, whether or not the device is acting as a publisher or subscriber. This usually avoids firewall problems because this approach works even behind firewalls or via NAT. In the case where the main communication is based on HTTP, the traditional approach for sending data to the device would be to use HTTP Polling. This isvery inefficient and costly, both in terms of network traffic as well as power requirements. The modern replacement for this is the WebSocket protocol9 that allows an HTTP connection to be upgraded into a full two-way connection. This then acts as a socket channel (similar to a pure TCP channel) between the server and client. Once that has been established, it is up to the system to choose an ongoing protocol to tunnel over the connection. For the reference architecture we once again recommend using MQTT as a protocol with WebSockets. In some cases, MQTT over Web Sockets will be the only protocol. This is because itis even more firewall-friendly than the base MQTT specification as well as supporting pure browser/JavaScript clients using the same protocol. Note that while there is some support for Web Sockets on small controllers, such as Arduino, the combination of network code, HTTP and Web Sockets would utilize most of the available code space on a typical Arduino 8-bit device. Therefore, it is recommended the use of Web Sockets on the larger 32-bit devices.

AGGREGATION/BUS LAYER

An important layer of the architecture is the layer that aggregates and brokers communications. This is an important layer for three reasons:

1. The ability to support an HTTP server and/or an MQTT broker to talk to the devices

2. The ability to aggregate and combine communications from different devices and to route communications to a specific device (possibly via a gateway)

3. The ability to bridge and transform between different protocols, e.g. to offer HTTP based APIs that are mediated into an MQTT message going to the device. The aggregation/bus layer provides these capabilities as well as adapting into legacy protocols. The bus layer may also provide some simple correlation and mapping from different correlation models (e.g. mapping a device ID into an owner‘s ID or vice-versa). Finally, the aggregation/bus layer needs to perform two key security roles. It must be able to act as an OAuth2 Resource

Server (validating Bearer Tokens and associated resource access scopes). It must also be able to act as a policy enforcement point (PEP) for policy-based access. In this model, the bus makes requests to the identity and access management layer to validate access requests. The identity and access management layer acts as a policy decision point (PDP) in this process. The bus layer then implements the results of these calls to the PDP to either allow or disallowresource access.

EVENT PROCESSING AND ANALYTICS LAYER

This layer takes the events from the bus and provides the ability to process and act upon these events. A core capability here is the requirement to store the data into a database. This may happen in three forms. The traditional model here would be to write a server-side application,
e.g. this could be a JAX-RS application backed by a database. However, there are many approaches where we can support more agile approaches. The first of these is to use a big data analytics platform. This is a cloudscalable platform that supports technologies such as Apache Hadoop to provide highly scalable map reduce analytics on the data coming from the devices. The second approach is to support complex event processing to initiate near real- time activities and actions based on data from the devices and from the rest of the system.

Our recommended approach in this space is to use the following approaches:

- Highly scalable, column-based data storage for storing events
- Map-reduce for long-running batch-oriented processing of data
- Complex event processing for fast in-memory processing and near real-time reaction and autonomic actions based on the data and activity of devices and other systems
- In addition, this layer may support traditional application processing platforms, such as Java Beans, JAX-RS logic, message-driven beans, or alternatives, such as node.js, PHP, Ruby or Python.

CLIENT/EXTERNAL COMMUNICATIONS LAYER

The reference architecture needs to provide a way for these devices to communicate outside of the device-oriented system. This includes three main approaches. Firstly, we need the ability to create web-based front-ends and portals that interact with devices and with the event-processing layer. Secondly, we need the ability to create dashboards that offer views into analytics and event processing. Finally, we need to be able to interact with systems outside this network using machine-to-machine communications (APIs). These APIs need to be managed and controlled and this happens in an API management system. The recommended approach to building the web front end is to utilize a modular front-endarchitecture, such as a portal, which allows simple fast composition of useful UIs. Of course, the architecture also supports existing Web server-side technology, such as Java Servlets/ JSP, PHP, Python, Ruby, etc. Our recommended approach is based on the Java framework and the most popular Java-based web server, Apache Tomcat. The dashboard is a re-usable system focused on creating graphs and other visualizations of data coming from the devices and the event processing layer.

The API management layer provides three main functions:

- The first is that it provides a developer-focused portal (as opposed to the user focused portal previously mentioned), where developers can find, explore, and subscribe to APIs from the system. There is also support for publishers to create, version, and manage the available and published APIs;
- The second is a gateway that manages access to the APIs, performing access control checks (for external requests) as well as throttling usage based on policies. It also performs routing and load- balancing;

- The final aspect is that the gateway publishes data into the analytics layer where it is stored as well as processed to provide insights into how the APIs are used.

DEVICE MANAGEMENT

Device management (DM) is handled by two components. A server-side system (the device manager) communicates with devices via various protocols and provides both individual and bulk control of devices. It also remotely manages software and applications deployed on the device. It can lock and/or wipe the device if necessary. The device manager works in conjunction with the device management agents. There are multiple different agents for different platforms and device types. The device manager also needs to maintain the list of device identities and map these into owners. It must also work with the identity and access management layer to manage access controls over devices (e.g. who else can manage the device apart from the owner, how much control does the owner have vs. the administrator, etc.) There are three levels of device: non-managed, semi-managed and fully managed (NM, SM, FM). Fully managed devices are those that run a full DM agent.

A full DM agent supports:

- Managing the software on the device
- Enabling/disabling features of the device (e.g. camera, hardware, etc.)
- Management of security controls and identifiers
- Monitoring the availability of the device • Maintaining a record of the device location ifavailable
- Locking or wiping the device remotely if the device is compromised, etc.

Non-managed devices can communicate with the rest of the network, but have no agent involved. These may include 8-bit devices where the constraints are too small to support the agent. The device manager may still maintain information on the availability and location of the device if this is available. Semi-managed devices are those that implement some parts of the DM (e.g. feature control, but not software management).

IDENTITY AND ACCESS MANAGEMENT

The final layer is the identity and access management layer. This layer needs to provide the following services:

- OAuth2 token issuing and validation
- Other identity services including SAML2 SSO and OpenID Connect support for identifyinginbound requests from the Web layer
- XACML PDP
- Directory of users (e.g. LDAP)
- Policy management for access control (policy control point)

The identity layer may of course have other requirements specific to the other identity and access management for a given instantiation of the reference architecture. In this section we have outlined the major components of the reference architecture as well as specific decisions we have taken around technologies. These decisions are motivated by the specific requirements of IoT architectures as well as best practices for building agile, evolvable, scalable Internet architectures.

IoT Reference Model

In an IoT system, data is generated by multiple kinds of devices, processed in different ways, transmitted to different locations, and acted upon by applications. The proposed IoT reference model is comprised of seven levels. Each level is defined with terminology that can be standardized to create a globally accepted frame of reference. The IoT Reference Model does not restrict the scope or locality of its components. For example, from a physical perspective, every element could reside in a single rack of equipment or it could be distributed across the world. The IoT Reference Model also allows the processing occurring at each level to range from trivial to complex, depending on the situation. The model describes how tasks at each level should be handled to maintain simplicity, allow high scalability, and ensure supportability. Finally, the model defines the functions required for an IoT system to be complete. Figure illustrates the IoT Reference model and its levels. It is important to note that in the IoT, data flows in both directions. In a control pattern, control information flows from the top of the model (level 7) to the bottom (level 1). In a monitoring pattern, the flowof information is the reverse. In most systems, the flow will be bidirectional.

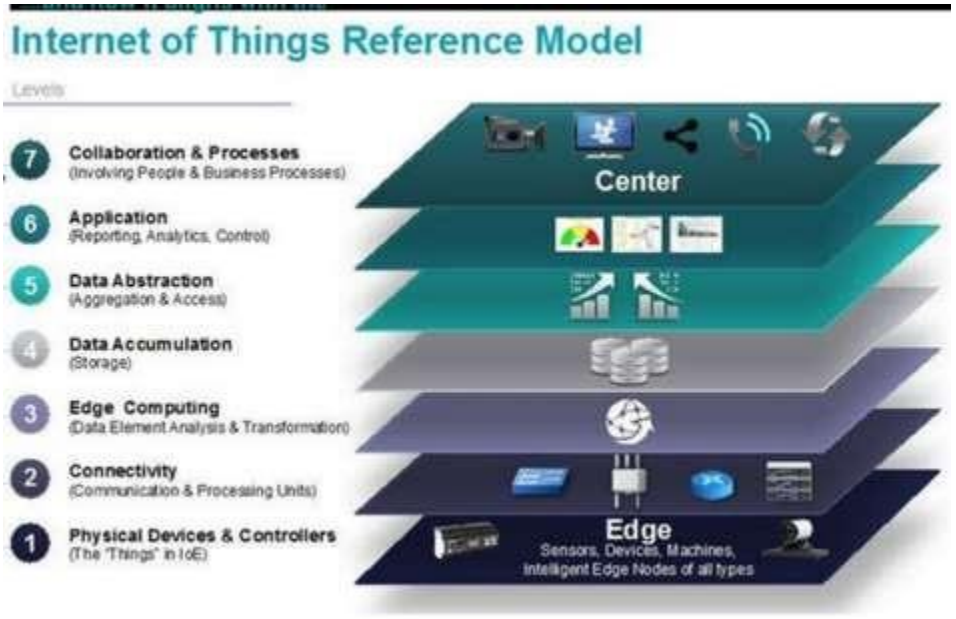


Fig 2.4 IoT Reference Model

Level 1: Physical Devices and Controllers

The IoT Reference Model starts with Level 1: physical devices and controllers that might control multiple devices. These are the "things" in the IoT, and they include a wide range of endpoint devices that send and receive information. Today, the list of devices is already extensive. It will become almost unlimited as more equipment is added to the IoT over time. Devices are diverse, and there are no rules about size, location, form factor, or origin. Some devices will be the size of a silicon chip. Some will be as large as vehicles. The IoT must support the entire range. Dozens or hundreds of equipment manufacturers will produce IoT devices. To simplify compatibility and support manufacturability, the IoT Reference Model generally describes the level of processing needed from Level 1 devices.

Level 2: Connectivity

Communications and connectivity are concentrated in one level—Level 2.

The most important function of Level 2 is reliable, timely information transmission. This includes transmissions: ● Between devices (Level 1) and the network ● Across networks (east-west) ● Between the network (Level 2) and low-level information processing occurring at Level 3

Traditional data communication networks have multiple functions, as evidenced by the International Organization for Standardization (ISO) 7-layer reference model. However, a complete IoT system contains many levels in addition to the communications network. One objective of the IoT Reference Model is for communications and processing to be executed by existing networks. The IoT Reference Model does not require or indicate creation of a different network—it relies on existing networks. However, some legacy devices aren't IP- enabled, which will require introducing communication gateways. Other devices will require proprietary controllers to serve the communication function. However, over time, standardization will increase. As Level 1 devices proliferate, the ways in which they interact with Level 2 connectivity equipment may change. Regardless of the details, Level 1 devices communicate through the IoT system by interacting with Level 2 connectivity equipment

Level 3: Edge (Fog) Computing

The functions of Level 3 are driven by the need to convert network data flows into information that is suitable for storage and higher-level processing at Level 4 (data accumulation). This means that Level 3 activities focus on high-volume data analysis and transformation. For example, a Level 1 sensor device might generate data samples multiple times per second, 24 hours a day, 365 days a year. A basic tenet of the IoT Reference Model is that the most intelligent system initiates information processing as early and as close to the edge of the network as possible. This is sometimes referred to as fog computing. Level 3 is where this occurs. Given that data is usually submitted to the connectivity level (Level 2) networking equipment by devices in small units, Level 3 processing is performed on a packet-by-packet basis. This processing is limited, because there is only awareness of data units—not "sessions" or "transactions." Level 3 processing can encompass many examples, such as:

- Evaluation: Evaluating data for criteria as to whether it should be processed at a higherlevel
- Formatting: Reformatting data for consistent higher-level processing
- Expanding/decoding: Handling cryptic data with additional context (such as the origin)
- Distillation/reduction: Reducing and/or summarizing data to minimize the impact of data and traffic on the network and higher-level processing systems
- Assessment: Determining whether data represents a threshold or alert; this could include redirecting data to additional destinations

Level 4: Data Accumulation

Networking systems are built to reliably move data. The data is "in motion." Prior to Level 4, data is moving through the network at the rate and organization determined by the devices generating the data. The model is event driven. As defined earlier, Level 1 devices do not include computing capabilities themselves. However, some computational activities could occur at Level 2, such as protocol translation or application of network security policy. Additional compute tasks can be performed at Level 3, such as packet inspection. Driving computational tasks as close to the edge of the IoT as possible, with heterogeneous systems distributed across multiple management domains represents an example of fog computing.

Fog computing and fog services will be a distinguishing characteristic of the IoT. Most applications cannot, or do not need to, process data at network wire speed. Applications typically assume that data is "at rest"—or unchanging—in memory or on disk. At Level 4, Data Accumulation, data in motion is converted to data at rest.

Level 4 determines:

- If data is of interest to higher levels: If so, Level 4 processing is the first level that is configured to serve the specific needs of a higher level.
- If data must be persisted: Should data be kept on disk in a non-volatile state or accumulated in memory for short-term use?
- The type of storage needed: Does persistency require a file system, big data system, or relational database?
- If data is organized properly: Is the data appropriately organized for the required storage system?
- If data must be recombined or recomputed: Data might be combined, recomputed, or aggregated with previously stored information, some of which may have come from non-IoT sources.

As Level 4 captures data and puts it at rest, it is now usable by applications on a non-real- time basis. Applications access the data when necessary. In short, Level 4 converts event- based data to query-based processing. This is a crucial step in bridging the differencesbetween the real-time networking world and the non-real-time application world.

Level 5: Data Abstraction

IoT systems will need to scale to a corporate—or even global—level and will require multiple storage systems to accommodate IoT device data and data from traditional enterprise ERP, HRMS, CRM, and other systems. The data abstraction functions of Level 5 are focused on rendering data and its storage in ways that enable developing simpler, performance- enhanced applications.

With multiple devices generating data, there are many reasons why this data may not land in the same data storage:

- There might be too much data to put in one place.
- Moving data into a database might consume too much processing power, so that retrieving it must be separated from the data generation process. This is done today with onlinetransaction processing (OLTP) databases and data warehouses.
- Devices might be geographically separated, and processing is optimized locally.
- Levels 3 and 4 might separate "continuous streams of raw data" from "data that represents an event." Data storage for streaming data may be a big data system, such as Hadoop. Storagefor event data may be a relational database management system (RDBMS) with faster query times.
- Different kinds of data processing might be required.

For example, in-store processing will focus on different things than across-all-stores summary processing. For these reasons, the data abstraction level must process many different things. These include:

- Reconciling multiple data formats from different sources
- Assuring consistent semantics of data across sources
- Confirming that data is complete to the higher-level application

- Consolidating data into one place (with ETL, ELT, or data replication) or providing accessto multiple data stores through data virtualization
- Protecting data with appropriate authentication and authorization
- Normalizing or denormalizing and indexing data to provide fast application access Level

Application Level 6

It is the application level, where information interpretation occurs. Software at this level interacts with Level 5 and data at rest, so it does not have to operate at network speeds. The IoT Reference Model does not strictly define an application. Applications vary based on vertical markets, the nature of device data, and business needs. For example, some applications will focus on monitoring device data. Some will focus on controlling devices. Some will combine device and non-device data. Monitoring and control applications represent many different application models, programming patterns, and software stacks, leading to discussions of operating systems, mobility, application servers, hypervisors, multi- threading, multi-tenancy, etc. These topics are beyond the scope of the IoT Reference Model discussion. Suffice it to say that application complexity will vary widely.

Examples include:

- Mission-critical business applications, such as generalized ERP or specialized industrysolutions
- Mobile applications that handle simple interactions
- Business intelligence reports, where the application is the BI server
- Analytic applications that interpret data for business decisions
- System management/control center applications that control the IoT system itself and don'tact on the data produced by it

If Levels 1-5 are architected properly, the amount of work required by Level 6 will be reduced. If Level 6 is designed properly, users will be able to do their jobs better.

Level 7: Collaboration and Processes

One of the main distinctions between the Internet of Things (IoT) and IoT is that IoT includes people and processes. This difference becomes particularly clear at Level 7: Collaboration and Processes. The IoT system, and the information it creates, is of little value unless it yieldsaction, which often requires people and processes. Applications execute business logic to empower people. People use applications and associated data for their specific needs. Often, multiple people use the same application for a range of different purposes. So, the objective isnot the application—it is to empower people to do their work better. Applications (Level 6) give business people the right data, at the right time, so they can do the right thing. But frequently, the action needed requires more than one person. People must be able to communicate and collaborate, sometimes using the traditional Internet, to make the IoT useful. Communication and collaboration often require multiple steps. And it usually transcends multiple applications. This is why Level 7, represents a higher level than a single application.

IoT Reference Architecture

- Reference Architecture is a starting point for generating concrete architectures and actual systems. A concrete

architecture addresses the concerns of multiple stakeholders of the actual system, and it is typically presented as a series of views thataddress different stakeholder concerns.

- A Reference Architecture, on the other hand, serves as a guide for one or more concrete system architects. However, the concept of views for the presentation of an architecture is also useful for the IoT Reference Architecture.
- Views are useful for reducing the complexity of the Reference Architecture blueprintsby addressing groups of concerns one group at a time.
- However, since the IoT Reference Architecture does not contain details about the environment where the actual system is deployed, some views cannot be presented in detail or at all; for example, the view that shows the concrete Physical Entities and Devices for a specific scenario.
- The stakeholders for a concrete IoT system are the people who use the system(Human Users); the people who design, build, and test the Resources , Services, Active Digital Artifacts, and Applications; the people who deploy Devices and attach them to Physical Entities; the people who integrate IoT capabilities of functions with an existing ICT system (e.g. of an enterprise); the people who operate, maintain, and troubleshoot the Physical and Virtual Infrastructure; and the people who buy and own an IoT system or parts thereof (e.g. city authorities).
- In order to address the concerns of mainly the concrete IoT architect, and secondly theconcerns of most of the above stakeholders, we have chosen to present the Reference Architecture as a set of architectural views .
- Functional View: Description of what the system does, and its main functions.
- Information View: Description of the data and information that the system handles.
- Deployment and Operational View: Description of the main real world components ofthe system such as devices, network routers, servers, etc.

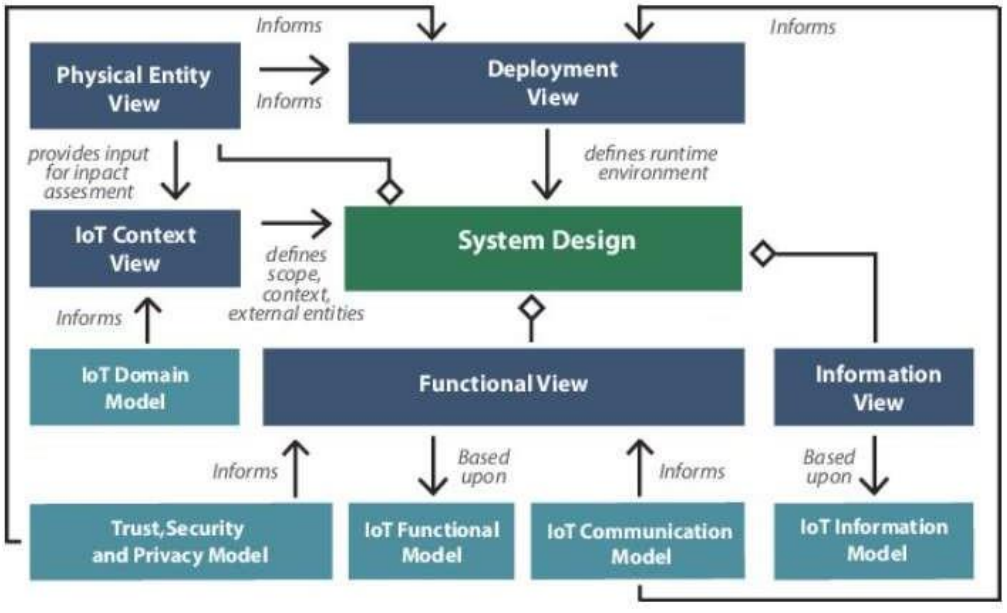


Fig: 2.5 Various views in IoT

● The **Development View** defines how to implement the system.

The Development View addresses the concerns of developers and testers.

○ All software projects involve some amount of new code being written.

○ This view provides a stable environment for more detailed design work.

● The Deployment View defines how to transition the system to live operation.

Focuses on aspects of the system important after the system has been built and is ready to be put into live operation.

● Defines: ○ The physical environment it will run in. ■ Hardware and hosting environment (processing nodes, network interconnections, disk storage). ○ Technical environment requirements for each processing node. ○ Mapping of elements to the runtime environment that will execute them.

It is needed when the system has…

○ Complex runtime dependencies. ■ Third party libraries, network services. ○ Complex runtime environments. ■ Elements distributed across many machines. ○ Dependencies on unfamiliar HW/SW. ■ Deployed on cloud hardware. ● When the system will be deployed in… ○ Wildly varying software environments. ■ Commercial software run on a PC. ○ Wildly varying physical environments. ■ Specialist or unfamiliar hardware.

● **The Operational View** defines how to keep the system alive in the field

Identifies a system-wide strategy for addressing operational concerns. ○ Helps to ensure system is a reliable and effective part of its environment. ○ For packaged software, helps illustrate the types of issues that could occur once installed. ○ Documents how the system can be architected to reduce or address these concerns. ● Often least well-defined view, as many of the details are not fully-defined until construction is underway

Installation and upgrade

○ Team performs the install. ○ Users install and configure themselves. ○ Resources allocated to a cloud environment. ● Is this a pure installation or an upgrade? ○ Upgrades can be more complex. ○ Must respect existing data and settings, state of running elements. ○ Can you keep the system running during update? ● Ensure the system can be installed or updated successfully.

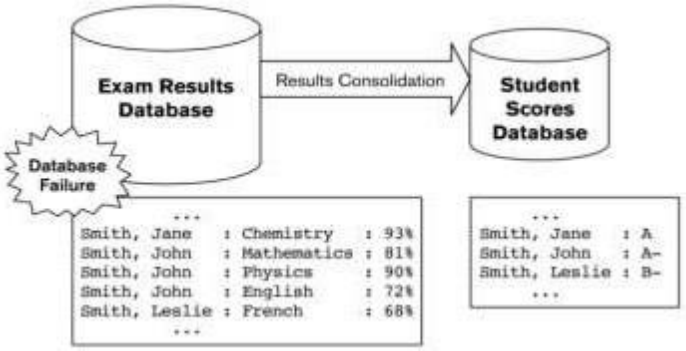
Documenting Installation and Upgrade ● Help the reader understand: ○ What needs to be installed or upgraded to move the system into production. ○ What dependencies exist between groups of items to be installed or upgraded (determines event order). ○ What constraints exist on the installation process. ○ What needs to be done to abandon and undo the installation/upgrade if there is a problem. ● Do not need a complete guide. ○ Instead, constraints the architecture imposes on installation and upgrade.

Operational Monitoring and Control ● Systems require routine monitoring. ● Control operations can be used to keep the system running correctly. ○ Startup, shutdown, transaction resubmission. ● How much is required depends on how many unexpected operational conditions are likely to occur. ● Balance against cost and time. ● Consider deployment environment to identify solutions.

Alerting ● A system should send notifications when something bad happens. ○ Technical: Unable to connect to database. ○ Functional: Bad data on an automated input. ○ Significant non-error conditions (startup, shutdown) ● Active function of a system. ○ Sent to appropriate humans for action. ● Define which events require alerts, what information should be included, and where it should be sent. ● Avoid sending too many alerts

Backup and Restore ● Data must be protected and insured. ○ Backup processes should be designed, built, and tested regularly. ● It must be possible to restore data from a backup in a transactionally consistent state. ○ All

updates committed to the restored database or not recovered at all. ○ Consider data lost as part of restoring (at least any transactions active during failure). ● Failure in one element could corrupt system. ○ Recover or recreate lost data. ○ Revert system to older state.



Academic records in databases. ○ Exam results database. ○ Scores database transforms data into a overall score. ● Corruption requires restoration of exam database. ○ Over three months old. ○ Results from those months will need to be reentered. ○ However, student scores already reflect that data. Must prevent reentered data from changing scores.

Documenting System Administration ● Monitoring and control facilities ○ How to monitor and adjust the system. ○ Custom utilities, existing management environments. ○ Basic message log to full-blown infrastructure. ○ Define what features you will offer, how to use them, and any limitations. ● Required routine procedures ○ What needs to be performed regularly? ○ Backup and health check procedures. ○ Define purpose of each procedure, when performed, who performs it, and the steps involved.

Likely error conditions ○ Error conditions may require administrative intervention (disk full, network failure). ○ What is unique about architecture? ○ Explain error conditions, when they occur, how to recognize them, and HOW to correct them. ● Performance monitoring facilities ○ Watch the system for performance problems. ○ Extracted and analyzed routinely. ○ Explain measures taken, how they can be extracted and analyzed.

Functional view

- The functional view for the IoT Reference Architecture is presented in Figure.5.1 ,and is adapted from IoT-A .
- It consists of the Functional Groups (FGs) presented earlier in the IoT FunctionalModel, each of which includes a set of Functional Components (FCs).
- It is important to note that not all the FCs are used in a concrete IoT architecture, andtherefore the actual system as explained earlier

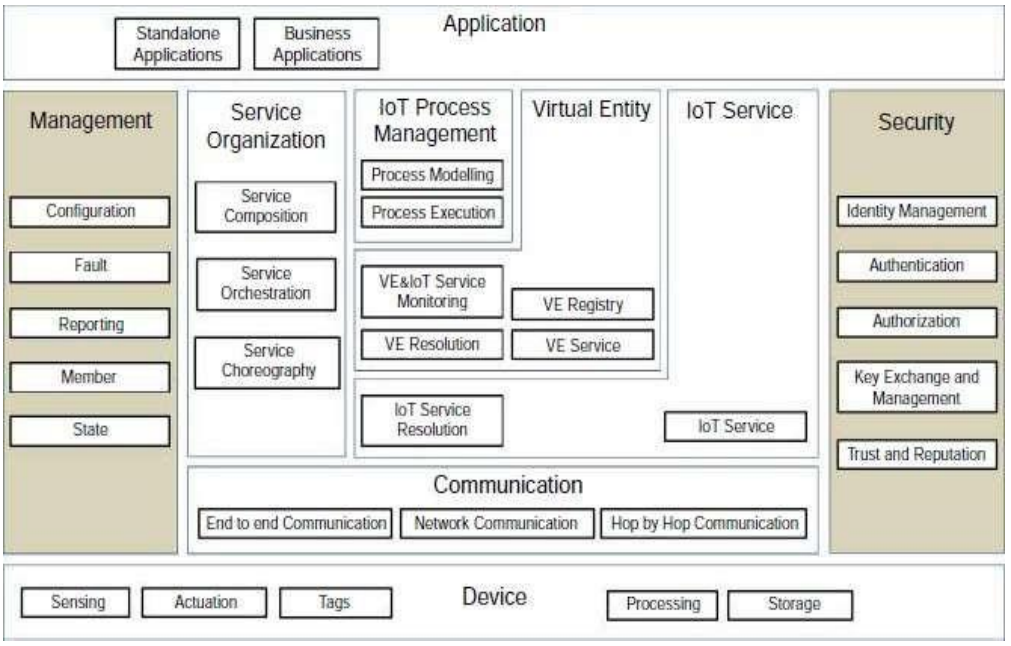


Fig.2.6 IoT Functional View

Device and Application functional group

- The Device and Application FGs are already covered in the IoT Functional Model. For convenience the Device FG contains the Sensing, Actuation, Tag, Processing, Storage FCs, or simply components.
- These components represent the resources of the device attached to the Physical Entities of interest. The Application FG contains either standalone applications (e.g. for iOS, Android, Windows phone), or Business Applications that connect the IoT system to an Enterprise system.

Communication functional group

The Communication FG contains the End-to-End Communication, Network Communication, and Hop-by-Hop communication components:

- The Hop-by-Hop Communication is applicable in the case that devices are equipped with mesh radio networking technologies such as IEEE 802.15.4 for which messages have to traverse the mesh from node-to-node (hop-by-hop) until they reach a gateway node which forwards the message (if needed) further to the Internet.
- The hop-by-hop FC is responsible for transmission and reception of physical and MAC layer frames to/from other devices. This FC has two main interfaces: (a) one "southbound" to/from the actual radio on the device, and (b) one "northbound" to/from the Network FC in the Communication FG.
- The Network FC is responsible for message routing & forwarding and the necessary translations of various identifiers and addresses.
- The translations can be (a) between network layer identifiers to MAC and/or physical network identifiers, (b) between high-level human readable host/node identifiers to network layer addresses (e.g. Fully Qualified

Domain Names (FQDN) to IP addresses, a function implemented by a Domain Name System (DNS) server), and (c) translation between node/service identifiers and network locators in case the higher layers above the networking layer use node or service identifiers that are decoupled from the node addresses in the network (e.g.Host Identity Protocol (HIP; Moskovitz & Nikander 2006) identifiers and IP addresses).

- Potential fragmentation and reassembly of messages due to limitations of the underlying layers is also handled by the Network FC.
- Finally, the Network FC is responsible for handling messages that cross different networking or MAC/PHY layer technologies, a function that is typically implementedon a network gateway type of device.
- The End-to-End Communication FC is responsible for end-to-end transport of application layer messages through diverse network and MAC/PHY layers.
- In turn, this means that it may be responsible for end-to-end retransmissions of missing frames depending on the configuration of the FC. For example, if the End-to- End Communication FC is mapped in an actual system to a component implementingthe Transmission Control Protocol (TCP) protocol, reliable transfer of frames dictates the retransmission of missing frames.
- Finally, this FC is responsible for hosting any necessary proxy/cache and any protocol translation between networks with different transport/application layer technologies. An example of such functionality is the HTTP-CoAP proxy, which performs transport-layer protocol translation. The End-to-End FC interfaces the Network FCon the “southbound” direction.

IoT Service functional group

The IoT Service FG consists of two FCs: The IoT Service FC and the IoT ServiceResolution FC:

- The IoT Service FC is a collection of service implementations, which interface the related and associated Resources. For a Sensor type of a Resource, the IoT Service FC includes Services that receive requests from a User and returns the Sensor Resource value in synchronous or asynchronous (e.g. subscription/notification) fashion.
- The services corresponding to Actuator Resources receive User requests for actuation, control the Actuator Resource, and may return the status of the Actuator after the action.
- A Tag IoT Service can behave both as a Sensor (for reading the identifier of the Tag), or as an Actuator (for writing a new identifier or information on the Tag, if possible).
- The IoT Service Resolution FC contains the necessary functions to realize a directory of IoT Services that allows dynamic management of IoT Service descriptions and discovery/lookup/resolution of IoT Services by other Active Digital Artifacts.
- The Service descriptions of IoT Services contain a number of attributes as seen earlier in the IoT Functional Model section. Dynamic management includes methods such as creation/update/ deletion (CUD) of Service description, and can be invoked by both the
- IoT Services themselves, or functions from the Management FG (e.g.bulk creation of IoT Service descriptions upon system start-up).

- The discovery/lookup and resolution functions allow other Services or Active Digital Artifacts to locate IoT Services by providing different types of information to the IoT Service Resolution FC.
- By providing the Service identifier (attribute of the Service description) a lookup method invocation to the IoT Service Resolution returns the Service description, while the resolution method invocation returns the contact information (attribute of the service description) of a service for direct Service invocation (e.g. URL).
- The discovery method, on the other hand, assumes that the Service identifier is unknown, and the discovery request contains a set of desirable Service description attributes that matching Service descriptions should contain.

Virtual Entity functional group

- The **Virtual Entity FG** contains functions that support the interactions between Users and Physical Things through Virtual Entity services.
- An example of such an interaction is the query to an IoT system of the form, "What isthe temperature in the conference room Titan?" The Virtual Entity is the conference room "Titan," and the conference room attribute of interest is "temperature."
- Assuming that the room is actually instrumented with a temperature sensor, if theUser had the knowledge of which temperature sensor is installed in the room (e.g. TempSensor #23), then the User could re-formulate and re-target this query to, "What is the value of TempSensor #23?" dispatched to the relevant IoT Service representing the temperature resource on the TempSensor #23.
- The Virtual Entity interaction paradigm requires functionality such as discovery of IoT Services based on Virtual Entity descriptions, managing the Virtual Entity-IoT Service associations, and processing VirtualEntity-based queries. The following FCs are defined for realizing these functionalities:
- The **Virtual Entity Service FC** enables the interaction between Users and Virtual Entities by means of reading and writing the Virtual Entity attributes (simple or complex), which can be read or written, of course.
- Some attributes (e.g. the GPS coordinates of a room) are static and non-writable by nature, and some other attributes are non-writable by access control rules.
- In general attributes that are associated with IoT Services, which in turn represent Sensor Resources, can only be read. There can be, of course, special Virtual Entities associated with the same Sensor Resource through another IoT Service that allow write operations.
- An example of such a special case is when the Virtual Entity represents the Sensor device itself (for management purposes).
- In general, attributes that are associated with IoT Services, which in turn represent Actuator Resources, can be read and written. A read operation returns the actuator status, while a write operation results in a command sent to the actuator.
- The **Virtual Entity Registry FC** maintains the Virtual Entities of interest for the specific IoT system and their associations.

- The component offers services such as creating/reading/updating/deleting Virtual Entity descriptions and associations. Certain associations can be static; for example, the entity “Room #123” is contained in the entity “Floor #7” by construction, while other associations are dynamic, e.g. entity “Dog” and entity “Living Room” due to at least Entity mobility. Update and Deletion operations take the Virtual Entity identifieras a parameter.
- The Virtual Entity Resolution FC maintains the associations between Virtual Entities and IoT Services, and offers services such as creating/reading/updating/deleting associations as well as lookup and discovery of associations.
- The Virtual Entity Resolution FC also provides notification to Users about the status of the dynamic associations between a Virtual Entity and an IoT Service, and finally allows the discovery of IoT Services provided the certain Virtual Entity attributes.
- The Virtual Entity and IoT Service Monitoring FC includes: (a) functionality to assert static Virtual EntityIoT Service associations, (b) functionality to discover new associations based on existing associations or
- Virtual Entity attributes such as location or proximity, and (c) continuous monitoring of the dynamic associations between Virtual Entities and IoT Services and updates of their status in case existing associations are not valid any more.

IoT process management functional group

- The IoT Process Management FG aims at supporting the integration of business processes with IoT-related services. It consists of two FCs:

✓ The Process Modeling FC provides that right tools for modeling a business process that utilizes IoT-related services.

✓ The Process Execution FC contains the execution environment of the process models created by the Process Modelling FC and executes the created processes by utilizing the Service Organization FG in order to resolve high-level application requirements tospecific IoT services.

✓ **Service Organization functional group**

✓ The Service Organization FG acts as a coordinator between different Services offered by the system. It consists of the following FCs:

- The Service Composition FC manages the descriptions and execution environment of complex services consisting of simpler dependent services. An example of a complex composed service is a service offering the average of the values coming from a number of simple Sensor Services. The complex composed service descriptions can be wellspecified or dynamic/flexible depending on whether the constituent services are well-defined and known at the execution time or discovered on-demand. The objective of a dynamic composed service can be the maximization of the quality of information achieved by the composition of simpler Services, as is the case with the example “average” service described earlier.

- The Service Orchestration FC resolves the requests coming from IoT Process Execution FC or User into the concrete IoT services that fulfill the requirements.

- The Service Choreography FC is a broker for facilitating communication among Services using the Publish/Subscribe pattern. Users and Services interested in specific IoT- related services subscribe to the Choreography FC, providing the desirable service attributes even if the desired services do not exist. The Choreography FC notifies the Users when services fulfilling the subscription criteria are found.

✓ **Security functional group**

✓ The Security FG contains the necessary functions for ensuring the security and privacy of an IoT system. It consists of the following FCs:

- The Identity Management FC manages the different identities of the involved Services or Users in an IoT system in order to achieve anonymity by the use of multiple pseudonyms.

- The Authentication FC verifies the identity of a User and creates an assertion upon successful verification. It also verifies the validity of a given assertion.

- The Authorization FC manages and enforces access control policies. It provides services to manage policies (CUD), as well as taking decisions and enforcing them regarding access rights of restricted resources. The term "resource" here is used as a representation of any item in an IoT system that needs a restricted access. Such an item can be a database entry(Passive Digital Artifact), a Service interface, a Virtual Entity attribute (simple or complex), aResource/Service/Virtual Entity description, etc.

✓ The Key Exchange & Management is used for setting up the necessary security keys between two communicating entities in an IoT system.

✓ The Trust & Reputation FC manages reputation scores of different interacting entities in an IoT system and calculates the service trust levels.

✓ **Management functional group**

✓ The Management FG contains system-wide management functions that may use individual FC management interfaces. It is not responsible for the management of each component, rather for the management of the system as

✓ a whole. It consists of the following FCs:

- The Configuration FC maintains the configuration of the FCs and the Devices in an IoT system (a subset of the ones included in the Functional View). The component collects the current configuration of all the FCs and devices, stores it in a historical database, and compares current and historical configurations. The component can also set the system-wide configuration (e.g. upon initialization), which in turn translates to configuration changes to individual FCs and devices.

✓ The Fault FC detects, logs, isolates, and corrects system-wide faults if possible. This means that individual component fault reporting triggers fault diagnosis and fault recovery procedures in the Fault FC.The Member FC manages membership information about the relevant entities in an IoT system. Example relevant entities are the FGs, FCs Services, Resources, Devices, Users, and Applications. Membership

✓ information is typically stored in a database along with other useful information such as capabilities, ownership,

and access rules & rights, which are used by the Identity Management and Authorization FCs.

- The State FC is similar to the Configuration FC, and collects and logs state information from the current FCs, which can be used for fault diagnosis, performance analysis and prediction, as well as billing

✓ purposes. This component can also set the state of the other FCs based on system-wise state information.

- The Reporting FC is responsible for producing compressed reports about the system state based on input from FCs.

Information view

- The information view consists of (a) the description of the information handled in the IoT System, and (b) the way this information is handled in the system; in other words, the information lifecycle and flow (how information is created, processed, and deleted), and the information handling components.

- **Information description**

The pieces of information handled by an IoT system complying to an ARM such asthe IoT-A (Carrez et al. 2013) are the following:

- Virtual Entity context information, i.e. the attributes (simple or complex) as represented by parts of the IoT Information model (attributes that have values and metadata such as the temperature of a room). This is one

of the most important pieces of information that should be captured by an IoT system,and represents the properties of the associated Physical Entities or Things.

- IoT Service output itself is another important part of information generated by an IoT system. For example, this is the information generated by interrogating a Sensor or a Tag Service.

- Virtual Entity descriptions in general, which contain not only the attributes coming from IoT Devices (e.g. ownership information).

- Associations between Virtual Entities and related IoT Services.

- Virtual Entity Associations with other Virtual Entities (e.g. Room #123 is on Floor #7).
- IoT Service Descriptions, which contain associated Resources, interface descriptions,etc.

- Resource Descriptions, which contain the type of resource (e.g. sensor), identity, associated Services, and Devices.

- Device Descriptions such as device capabilities (e.g. sensors, radios).

- Descriptions of Composed Services, which contain the model of how a complex service is composed of simpler services.

- IoT Business Process Model, which describes the steps of a business process utilizing other IoT-related services (IoT, Virtual Entity,Composed Services).

- Security information such as keys, identity pools, policies, trust models, reputation scores, etc.
- Management information such as state information from operational FCs used for fault/performance purposes, configuration snapshots, reports, membership information, etc.

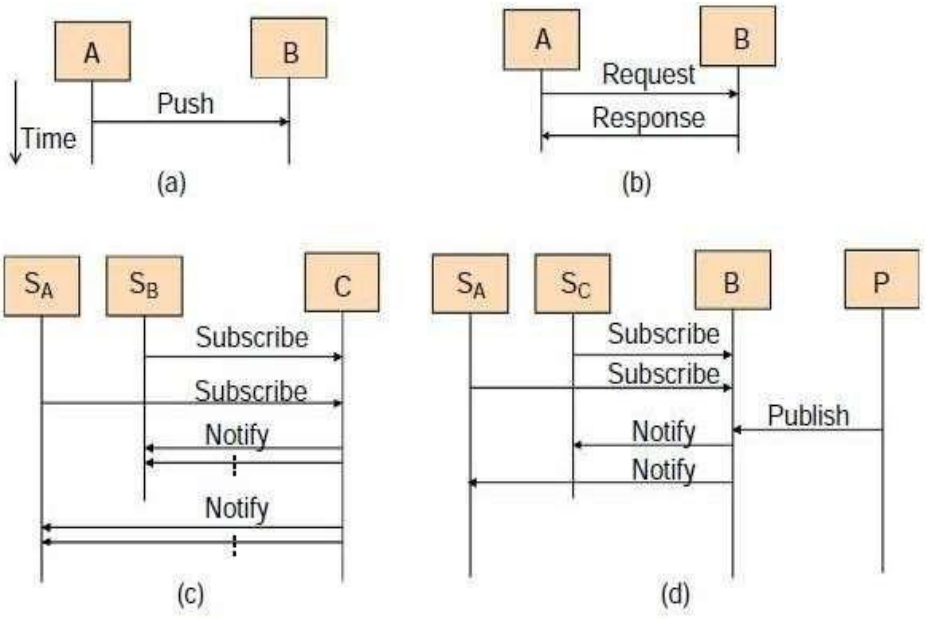


Fig. 2.7 Information exchange patterns.

- **Push:** An FC A pushes the information to another FC B provided that the contact information of the component B is already configured in component A, and component B listens for such information pushes.
- **Request/Response:** An FC A sends a request to another FC B and receives a response from B after A serves the request. Typically the interaction is synchronous inthe sense that A must wait for a response from B before proceeding to other tasks, but in practice this limitation can be realized with parts of component A waiting, and other parts performing other tasks. Component B may need to handle concurrent requests and responses from multiple components, which imposes certain requirements on the capabilities for the device or the network that hosts the FC.
- **Subscribe/Notify:** Multiple subscriber components (SA, SB) can subscribe for information to a component C, and C will notify the relevant subscribers when the requested information is ready. This is typically an asynchronous information request after which each subscriber can perform other tasks. Nevertheless, a subscriber needs to have some listening components for receiving the asynchronous response. The target component C also needs to maintain state information about which subscribers requested which information and their contact information.
- The Subscribe/Notify pattern is applicable when typically one component is the hostof the information needed by multiple other components. Then the subscribers need only establish a Subscribe/Notify relationship with one component. If multiple components can be information producers or information hosts, the Publish/Subscribe pattern is a more scalable solution from the point of view of the subscribers.
- **Publish/Subscribe:** In the Publish/Subscribe (also known as a Pub/Sub pattern), there is a third component called the broker B, which mediates subscription and publications between subscribers (information

consumers) and publishers (or information producers). Subscribers such as SA and SB subscribe to the broker about the information they are interested in by describing the different properties of the information. Publishers publish information and metadata to the broker, and the broker pushes the published information to (notification) the subscribers whose interests match the published information.

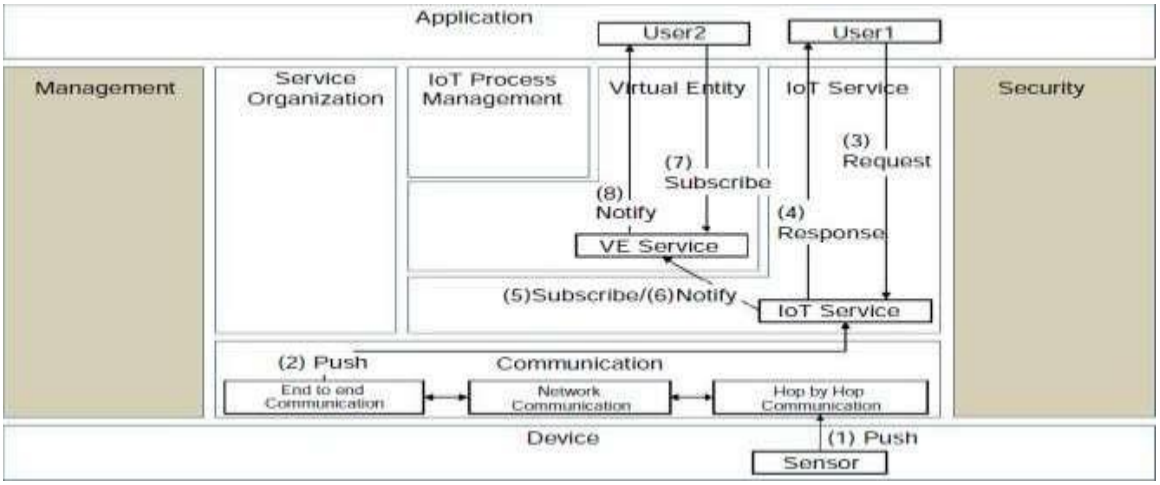


Fig. 2.8 Device, IoT Service, and Virtual Entity Service Interactions.

- In Figure it is assumed that the generated sensed data is pushed by a sensor device (under Steps 1 and 2) that is part of a multi-hop mesh network such as IEEE 802.15.4 through the Hop-by-Hop, Network, and End-to-End communication FCs towards the Sensor Resource hosted in the network.

- Please note that the Sensor Resource is not shown in the figure, only the associated IoT Service. A cached version of the sensor reading on the Device is maintained onthe IoT Service. When User1 (Step 3) requests the sensor reading value from the specific Sensor Device (assuming User1 provides the Sensor resource identifier), the IoT Service provides the cached copy of the sensor reading back to the User1 annotated with the appropriate metadata information about the sensor measurement, for example, timestamp of the last known reading of the sensor, units, and location of the Sensor Device.
- Also assume that that the Virtual Entity Service associated with the Physical Entity (e.g. a room in a building) where the specific Sensor Device has been deployed already contains the IoT Service as a provider of the "hasTemperature" attribute of its description. The Virtual Entity Service subscribes to the IoT Service for updates of the sensor readings pushed by the Sensor Device (Step 5). Every time the Sensor Device pushes sensor readings to the IoT Service, the IoT Service notifies (Step 6) the Virtual Entity Service, whichupdates the value of the attribute "hasTemperature" with the sensor reading of the Sensor Device. At a later stage, a User2 subscribing (Step 7) to changes on the VirtualEntity attribute "hasTemperature" is notified every time the attribute changes value (Step 8).

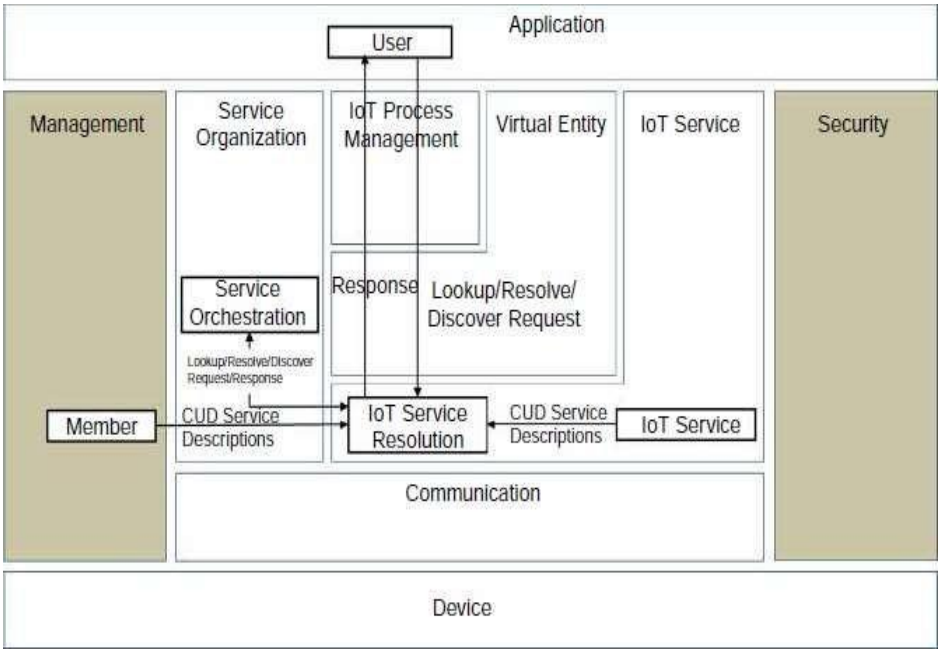


Fig. 2.9 IoT Service Resolution

- Figure depicts the information flow when utilizing the IoT Service Resolution FC. The IoT Service Resolution implements two main interfaces, one for the CUD of Service Description objects in the IoT Service Resolution database/store, and one for lookup/resolution/discovery of IoT Services.
- As a reminder, the lookup and resolution operations provide the Service Description and the Service locator, respectively, given the Service identifier and the discovery operation returns a (set of) Service Description(s) given a list of desirable attributes that matching Service Descriptions should contain.
- The CUD operations can be performed by the IoT Service logic itself or by a management component (e.g. Member FC in Figure). The lookup/resolution and discovery operation can be performed by a User as a standalone query or the Service Orchestration as a part of a Composed Service or an IoT Process.
- If a discovery operation returns multiple matching Service Descriptions, it is upon the User or the Service Orchestration component to select the most appropriate IoTService for the specific task.
- Although the interactions in Figure follow the Request/Response patterns, the lookup/resolution/discovery operations can follow the Subscribe/Notify pattern in the sense that a User or the Service Orchestration FC subscribe to changes of existing IoT Services for lookup/resolution and for the discovery of new Service Descriptions in the case of a discovery operation.

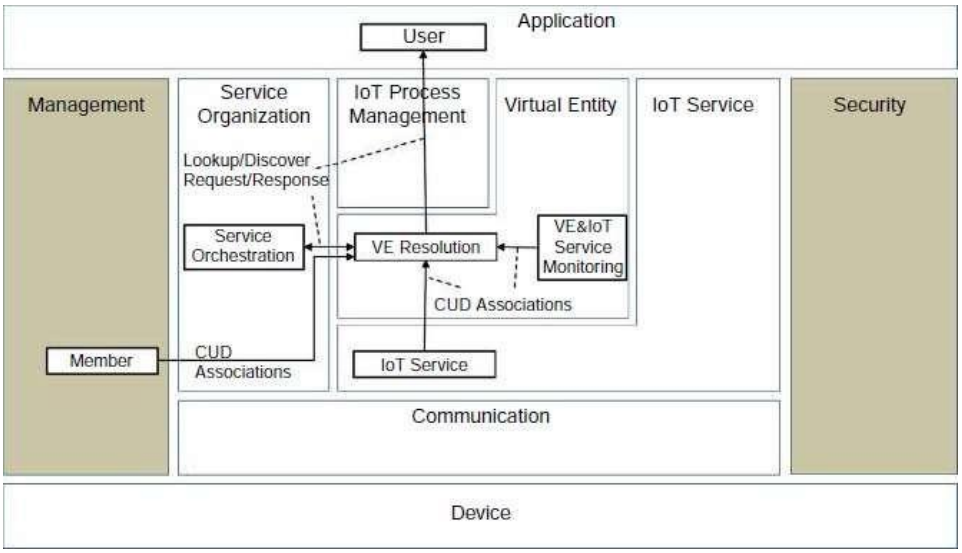


Fig. 2.10 Virtual Entity Service Resolution

- Figure describes the information flow when the Virtual Entity Service Resolution FC is utilized. The Virtual Entity Resolution FC allows the CUD of Virtual Entity Descriptions, and the lookup and discovery of Virtual Entity Descriptions.
- A lookup operation by a User or the Service Orchestration FC returns the Virtual Entity Description given the Virtual Entity identity, while the discovery operation returns the Virtual Entity Description(s) given a set of Virtual Entity attributes (simpleor complex) that matching Virtual Entities should contain.
- Note that the Virtual Entity Registry is also involved in the information flow because it is the storage component of Virtual Entity Descriptions, but it is omitted from the figure to avoid cluttering. The Virtual Entity Resolution FC mediates the requests/responses/ subscriptions/notifications between Users and the Virtual Entity Registry, which has a simple create/read/update/delete (CRUD) interface given the Virtual Entity identity.
- The FCs that could perform CUD operations on the Virtual Entity Resolution FC are the IoT Services themselves due to internal configuration, the Member Management FC that maintains the associations as part of the system setup, and the Virtual Entity and IoT Service Monitoring component whose purpose is to discover dynamic associations between Virtual Entities and IoT Services.
- It is important to note that the Subscribe/Notify interaction patterns can also be applicable to the lookup/ discovery operations, the same as the Request/Response patterns provided the involved FCs implement Subscribe/Notify interfaces.

Deployment and operational view

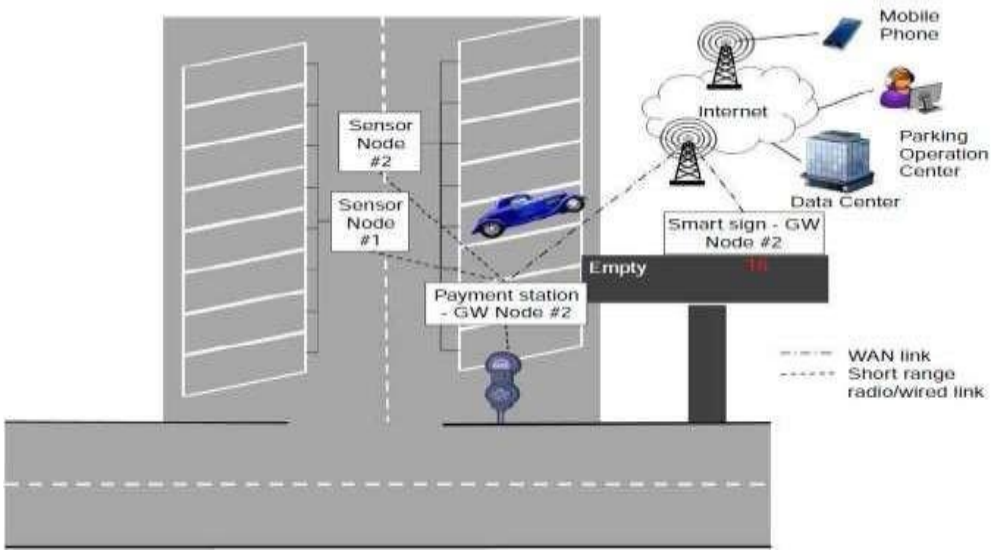


Fig. 2.11 Parking Lot Deployment and Operational View, Devices.

- The Deployment and Operational View depends on the specific actual use case and requirements, and therefore we present here one way of realizing the Parking Lot example seen earlier.
- Figure depicts the Devices view as Physical Entities deployed in the parking lot, as well as the occupancy sign. There are two sensor nodes (#1 and #2), each of which are connected to eight metal/car presence sensors.
- The two sensor nodes are connected to the payment station through wireless or wired communication. The payment station acts both as a user interface for the driver to pay and get a payment receipt as well as a communication gateway that connects the two sensor nodes and the payment interface physical devices (displays, credit card slots, coin/note input/output, etc.) with the Internet through Wide Area Network(WAN) technology.
- The occupancy sign also acts as a communication gateway for the actuator node (display of free parking spots), and we assume that because of the deployment, a direct connection to the payment station is not feasible (e.g. wired connectivity is too prohibitive to be deployed or sensitive to vandalism).
- The physical gateway devices connect through a WAN technology to the Internet and towards a data center where the parking lot management system software is hosted as one of the virtual machines on a Platform as a Service (PaaS;) configuration.
- The two main applications connected to this management system are human user mobile phone applications and parking operation center applications. We assume that the parking operation center manages several other parking lots using similar physicaland virtual infrastructure.

- Figure shows two views superimposed, the deployment and functional views, for the parking lot example. Please note that several FGs and FCs are omitted here for simplicity purposes, and certain non-IoT-specific

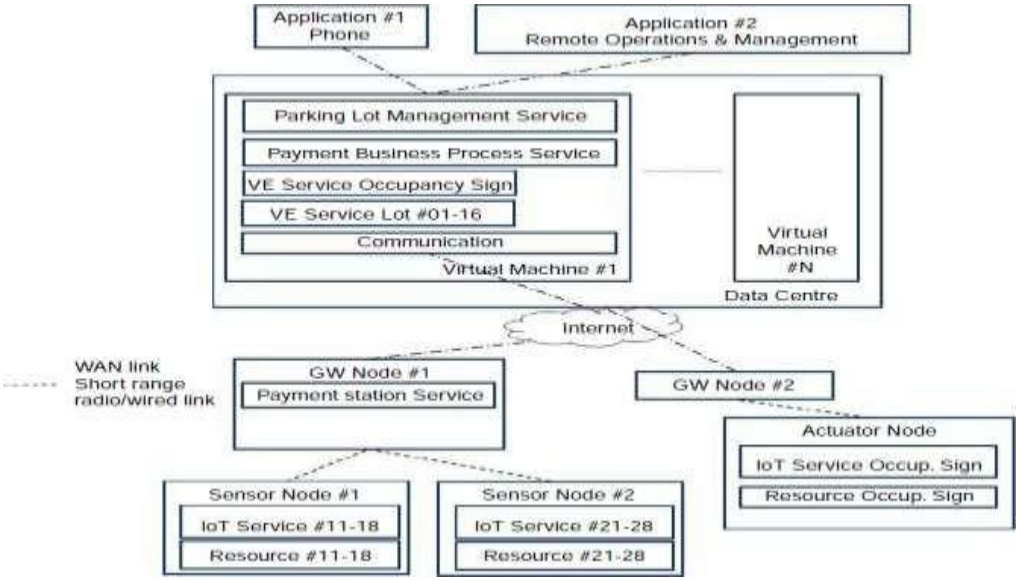
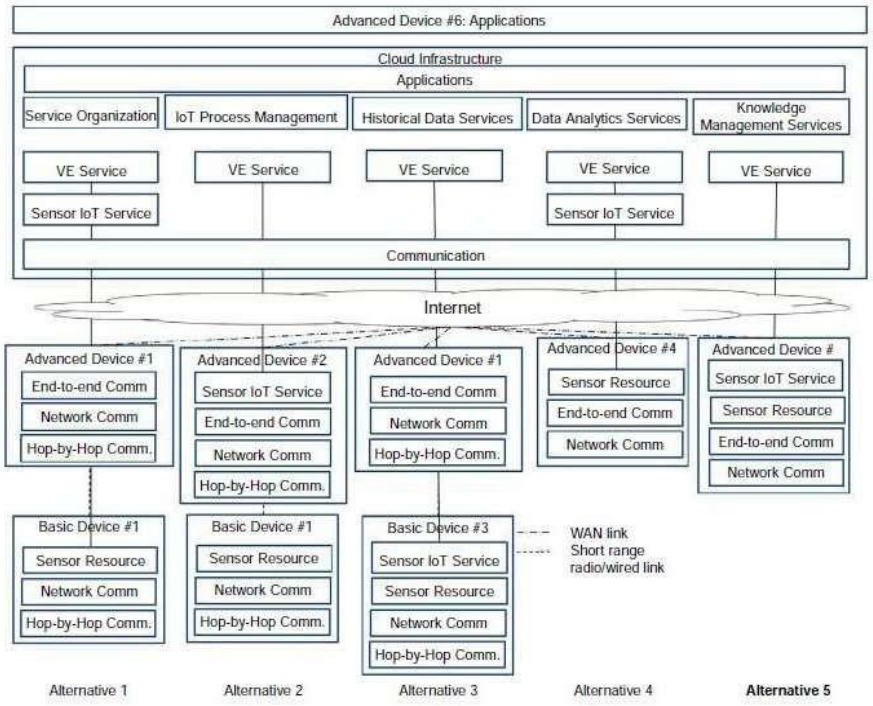


Fig 2.11 Parking Lot Deployment & Operational View, Resources, Services, VirtualEntities, Users

- Services appear in the figure 5.7 because an IoT system is typically part of a larger system. Starting from the Sensor Devices, as seen earlier, Sensor Node #1 hosts Resource #11#18, representing the sensors for the parking spots #01#08, while earlier Sensor Node #2 hosts Resource #21#28, representing the sensors for the parking spots#09#16.

- Assume that the sensor nodes are powerful enough to host the IoT Services #11#18 and #21#28 representing the respective resources. The two sensor nodes are connected to the gateway device that also hosts the payment service with the accompanying sensors and actuators, as seen earlier. The other gateway device hosts the occupancy sign actuator resource and corresponding service. The management system for the specific parking lot, as well as others, is deployed on a virtual machine on a data center. The virtual machine hosts communication capabilities, Virtual Entity services for the parking spots #01#16, the Virtual Entity services for the occupancy sign, a payment business process that involves the payment station and input from the occupancy sensor services, and the parking lot management service that provides exposure and access control to the parking lot occupancy data for the parking operation center and the

- consumer phone applications.As a reminder, the Virtual Entity service of the parking lot uses the IoT Services hosted on two sensor nodes and performs the mapping between the sensor node identifiers (#11#18 and #21#28) to parking spot identifiers (spot #01#16). The services offered on these parking spots are to read the current state of the parking spot to see whether it is “free” or “occupied.” The Virtual Entity corresponding to the occupancy sign contains one writable attribute: the number of free parking spots. A User writing this Virtual



- **Fig 2.12 Mapping IoT Domain Model concepts to Deployment View**

- Entity attribute results in an actuator command to the real actuator resource to change its display to the new value.

- Figure shows an example of mapping an IoT Domain Model and Functional View to Devices with different capabilities (different alternatives) connecting to a cloud infrastructure. Alternative 1 shows devices that can host only a simple Sensor Device and a short-range wired or wireless connectivity technology (Basic Device #1).

  • Such kind of device needs an Advanced Device of type #1 that allows the basic device to perform protocol adaptation (at least from the short-range wired or wireless connectivity technology to a WAN technology) so that the Sensor IoT service in thecloud and the Sensor Resource on the Basic Device #1 can exchange information.

- The Virtual Entity representing the Physical Entity where the Basic Device #1 is deployed is also hosted in the cloud.

- In alternative 2, Advanced Devices (type #2) can host the Sensor IoT Service communicating to the Sensor Resource on a Basic Device #1.

- The cloud infrastructure in this case only hosts the Virtual Entity Service corresponding to the Sensor IoT Service. The difference between alternative 1 and 2 is that the Sensor IoT Service hosted on an Advanced Device #2 should be capable of responding to requests from Users (cloud services, Applications) with the

appropriate secure mediation of course.

- In alternative 3, the Basic Device #3 is capable of providing the Sensor Resource and the Sensor IoT Service but still needs an Advanced Device #1 to transport IoT servicerequests/responses/subscriptions/
- notifications/publications to the Users in the cloud. According to experience, this kindof deployment scenario imposes a high burden on a Basic Device, which potentially makes the Basic Device the weakest link in the information flow
- If malicious Users launch a Denial of Service (DoS) attack on the node, the probability of the node going down is very high.
- Alternatives 4 and 5 show Advanced Devices offering a WAN interface.In alternative 4, only the Sensor Resource is hosted on the Device, while in alternative 5, even the IoT Service is hosted on the Device. The Virtual Entity Service is hosted in the cloud.

Real-World Design Constraints

Devices and Networks:

- The devices that form networks in the M2M Area Network domain must be selected, or designed, with certain functionality suitable to IoT applications.
- The devices must have an energy source (e.g. batteries), computational capability (e.g. an MCU), appropriate communications interface (e.g. a Radio Frequency Integrated Circuit (RFIC) and front end RF circuitry), memory (program and data), and sensing (and/or actuation) capability.
- These must be integrated in such a way that the functional requirements of the desired application can be satisfied with additional nonfunctional requirements.

Functional Requirements:

1. Specific sensing and actuating capabilities
2. Sensing principle and data requirements: Sometimes continuous sampling of sensing data is required. For some applications, sampling after specific intervals is required.
3. The parameters like higher network throughput, data loss, energy use, etc are decided based on sensing principle.

Sensing and communications field:

- The sensing field is to be considered for sensing in local area or distributed sensing. The distance between sensing points is also important factor to be considered.
- The physical environment has an implication on the communications technologies selected and the reliability of the system in operation thereafter.
- Devices must be placed in close enough proximity to communicate. Where the distance is too great, routing devices may be necessary.

Programming and embedded intelligence:

- Devices in the IoT are heterogeneous such as various computational architectures, including MCUs (8-, 16-,32-bit, ARM, 8051, RISC, Intel, etc.), signal conditioning (e.g. ADC), and memory (ROM, S/F/D) RAM, etc.), communications media, peripheral components (sensors, actuators, buttons, screens, LEDs), etc.
- In every case, an application programmer must consider the hardware selected or designed, and its capabilities.

- Application-level logic decides the sampling rate of the sensor, the local processing performed on sensor readings, the transmission schedule (or reporting rate), and the management of the communications protocol stack, among other things.
- The programmers have to reconfigure and reprogram devices in case of change in devices in IoT application.

Power:

- Power is essential for any embedded or IoT device.
- Depending on the application, power may be provided by the mains, batteries, or hybrid power sources.
- Power requirements of the application are modeled prior to deployment. This allows the designer to estimate the cost of maintenance over time.

Gateway:

- Gateway devices or proxies are selected according to need of data transitions.

Nonfunctional requirements:

The non-functional requirements are technical and non-technical.

1. Regulations:

- For applications that require placing nodes in public places, prior permissions are important.
- Radio Frequency (RF) regulations limit the power with which transmitters can broadcast.

2. Ease of use, installation, maintenance, accessibility:

- This relates to positioning, placement, site surveying, programming, and physical accessibility of devices for maintenance purposes.

3. Physical constraints:

- Integration of additional electronics into existing system
- Suitable packaging
- Kind and size of antenna
- Type of power supply

Financial cost:

Financial cost considerations are as follows:

- Component Selection: Typically, the use of these devices in the M2M Area Network domain is to reduce the overall cost burden. However, there are research and development costs likely to be incurred for each individual application in the IoT that requires device development or integration. Developing devices in small quantities is expensive.
- Integrated Device Design: Once the energy, sensors, actuators, computation, memory, power, connectivity, physical, and other functional and nonfunctional requirements are considered, it is likely that an integrated device must be produced.

Data representation and visualization:

Each IoT application has an optimal visual representation of the data and the system. Data that is generated from heterogeneous systems has heterogeneous visualization requirements. There are currently no satisfactory standard data representation and storage methods that satisfy all of the potential IoT applications.

- **Data Representation and Visualization**

- IoT Data Visualization is the technique where the raw data is presented into a more insightful one that is derived from different data streams. It analysis the data and looks into the certain patterns & behaviours that

improves with better business decision making. It helps to create a viable business strategy.

- The Data Visualization Helps to Unlock Multiple Insightful Values
- Helps to make real-time decisions with the combination of multiple data sources into a single insightful dashboard with multi-layered visual data.
- Combines the new IoT data transmitted from data sensors with the existing data to analyse and bring light to new business opportunities.
- Supports to monitor IoT devices and infrastructure for better performance on IoT dataflow.
- Helps to analyse multiple data correlations in real-time.
- Data Visualization Tools for IoT application:
- **Grafana Tool:**
- Grafana supports various data sources seamlessly like Elasticsearch, MySQL,PostgreSQL, Graphite, Prometheus and so on.
- Provides time series analytics to monitor, analyze data over a period of time.
- Upbeat of this Grafana tool is it provides on-premises cloud storage or any othercloud of your choice, which gives complete control of the infrastructure.
- Alert notification can be set up whenever an unfavourable event occurs which getsprompt notification using any communication platform.
- It has several built-in support features like Graphite, CloudWatch, Elastic Search,InfluxDB.
- Kibana Tool is an open source data visualization tool for analyzing large volumes of log data. To work with Kibana tool, it needs two more technological stack which is Elasticsearch and Logstash. It is popularly known as ELK stack, globally used log management platform.
- **Kibana Tool:**
- Working of kibana
- Initially, the logstash is responsible to collect all the data from the various remote sources
- Next, these data logs are then pushed and sent to the Elasticsearch
- Elasticsearch acts as the database to the kibana tool with all the log information
- Finally, Kibana tool presents these log data in the form of pie charts, bar or linegraphs to the user.
- Highlights of Kibana:
- Canvas visualization gives colorful visual data comprising of different patterns, texts known as workpad. Kibana also represents data in the form of bar chart, pie chart, heat map, line graph and so on.

- Contains Interactive dashboards and easily it can be converted into reports for future references
- Create visualization with the help of several dev tools where you can work with indexes to add, delete and update the data.
- Timelion, a timeline visualization tool helps to get the historical data and compare them with current data for getting deeper analysis.
- Supports third-party plugins and to get near to real experience view, it effectively uses coordinate and region maps
- Power BI Tool for Real-Time Data Visualization
- Microsoft's product PowerBI is a popular Business Intelligence Tool. Like its predecessors, Tableau and other BI tools, it provides a detailed analysis reports for large Enterprises. Power BI comes with a suite of products with Power BI desktop, mobile Power BI apps and Power BI services for SaaS.
- Power BI Desktop – Helps to create reports
- Power BI Services – Helps to Publish those reports
- Power BI mobile app – Help to Views the reports and dashboards
- How does Power BI work?
- First, the data is collected from the external data sources. With 'Get Data' option it allows you to get information from various sources including Facebook, GoogleAnalytics, Azure Cloud, Salesforce etc. Also, it provides ODBC connection to get ODBC data as well.
- Using Power BI, you can create visualization in 2 ways, one is by adding from the right-side panel to the report canvas which is in a table type visualization format or by simple drag and drop of value axis under visualization. Once the report is developed,it can be published to web portal with the help of Power BI service. We can access thereport, export it in pdf, excel or any preferred format.
- **Highlights of Power BI:**
- Though PowerBI offers paid services, it is comparatively cheaper than other BI tools. It offers free services upto 1GB storage
- Helps to analyse both streaming and static data
- Provides rich data visualization
- Short learning curve
- Provides IoT integrations
- **Industrial Automation**
- Industrial automation is all about intelligent process control. The IoT doesn't depend on your hardware because you can choose independent control systems, sensors, and network components. The IoT's power goes beyond

the limited features and functionalities your device manufacturer or software provider offers. Those who use the IoT in industrial automation processes can connect multiple sites and locations so that they operate in harmony. Industrial automation is well-known for diverting technology from the commercial sphere and adapting it to new ends. The industry's widespread IoT adoption builds upon this tested concept in numerous ways:

- Wireless Improvement of Existing Monitoring and Control Systems
- Wireless connectivity makes it simpler to implement complex control systems in awkward, remote or hazardous environments. For instance, using wired networks to link remotely controlled cranes, robot arms and other manufacturing devices can be problematic due to their unique ranges of motion and exposure to harsh fabrication environments. The IoT's compatibility with wireless technology lets enterprises replace standard linkages with fully enclosed mesh radios that perform the same functions. Even better, these alternatives may be more useful for automation processes that require fine-tuning or ongoing adjustments. For instance, you don't have to replace miles of Ethernet cable to achieve higher transmission speeds with Wi-Fi.
- Building Factories That Build and Run Themselves
- Growth has decided on the pros and cons. Although few experiences beat the thrill of taking your organizational training wheels off and cruising along, doing business at a higher volume introduces unique risks, such as the potential for greater waste should you take a wrong turn. A company that wanted to conserve resources might use an industrial sensor system to tell it when to shut down auxiliary production lines. An enterprise that relies on automated stock machines to transport replacement parts to workstations could employ a connected framework to initiate new deliveries without waiting for approval from a line manager. The IoT also makes it possible to create digital twins. These replicas of existing systems serve as testbeds for new projects andexperiments.
- Managing Communications Whenever the Need Arises
- The IoT enhances traditional automation schemes by making everything on-demand. When you make a change from a control dashboard, you get to see its effects ripple outward right away. What you might not expect is that the system also performs the innumerable tedious tasks that facilitate good digital communication, such as
- • Rerouting traffic to keep data moving no matter how much information happens to be passing through,
- • Accounting for the effects of network topologies to sustain optimized service quality,
- • Accommodating vendor-neutral communication protocols and schemes to support a wider variety of hardware and software,
- • Self-detecting equipment failures and automatically switching to functional network elements, and
- • Duplicating and storing data as necessary to prevent catastrophic losses.
- Although this kind of work may get overlooked because it goes on in the background, it's an essential part of ensuring that automation frameworks behave deterministically. When your industrial communication systems behave consistently, soundmanagement practices prove easier to execute.
- Decentralizing Debugging and Maintenance

- There’s no shortage of industrial automation maintenance philosophies to choose from, so debugging can get confusing. IoT mesh networks help stakeholders handle maintenance more logically. You can debug, tweak and maintain controllers and sensors from local network nodes to cut down on overhead and make the best use of limited bandwidth. Decentralized maintenance is the glue that helps automation systems stick together and run seamlessly even as they expand. By using the IoT to program functions at the node level, you can optimize resource usage and slash costs for a more productive enterprise.

- Investing in the IoT in Industrial Automation Settings

- Internet of Things technologies offer a spectrum of other potential benefits that we haven’t even covered. There are voice-recognition systems that let factory owners authenticate themselves and implement complex behaviours without any manual programming. Embedded and linked networks contribute to improved lifecycle oversight, demand-specific customization and better cost control, but choosing the best-equipped IoT layout and technical components can be a tough task. Optimality isn’t universal. It’s defined by the circumstances, so you need to move forward with an eye on building something that’s sufficiently flexible yet robust enough to survive the unexpected.

- **Interaction and Remote control**

- IoT devices produce many types of information, including telemetry, metadata, state, and commands and responses. Telemetry data from devices can be used in short operational timeframes or for longer-term analytics and model building. (For more on this diversity, read the overview of Internet of Things.)
    - Many devices support local monitoring in the form of a buzzer or an alarm panel on- premises.

  This type of monitoring is valuable, but has limited scope for in-depth or long-term analysis. This article instead discusses remote monitoring, which involves gathering and analysing monitoring information from a remote location using cloudresources. Operational and device performance data is often in the form of a time series, where each piece of information includes a time stamp. This data can be further enriched with dimensional labels (sometimes referred to as tags), such as labels that identify hardware revision, operating time zone, installation location, firmware version, and so on.

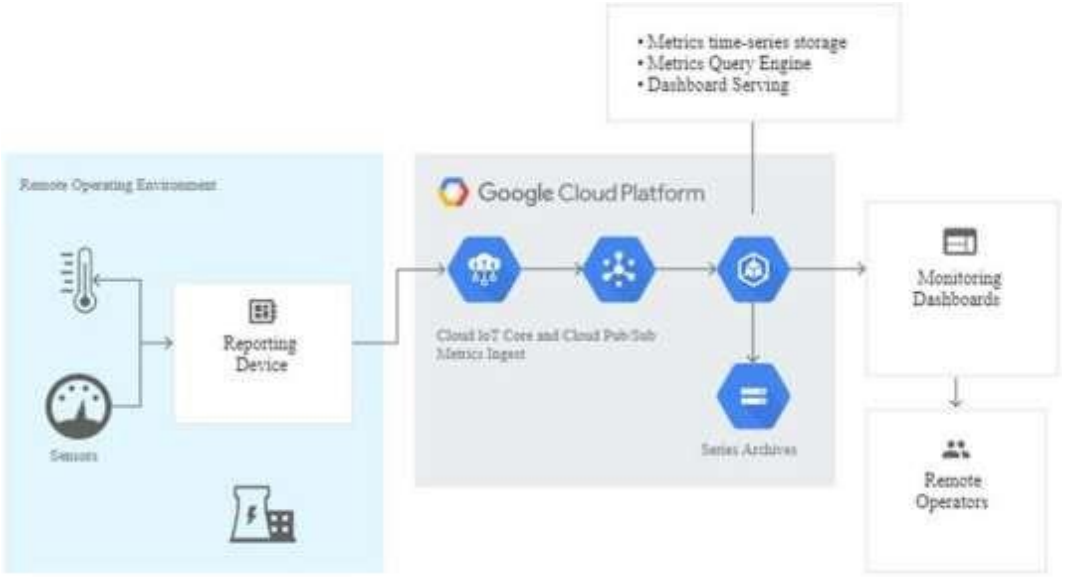
- Time-series telemetry can be collected and used for monitoring. Monitoring in this context refers to using a suite of tools and processes that help detect, debug, and resolve problems that occur in systems while those systems are operating. Monitoring can also give you insight into the systems and help improve them.

- The state of monitoring IT systems, including servers and services, has continuously improved. Monitoring tools and practices in the cloud-native world of microservices and Kubernetes are excellent at monitoring based on time-series metric data. These tools aren't designed specifically for monitoring IoT devices or physical processes, but the constituent parts—labelled series of metrics, visualization, and alerts—all can apply to IoT monitoring.

- **Remoteness**

- Unlike servers in a cluster, monitored devices might be far from the systems that are organizing the metric data and providing visualizations. There is debate in the monitoring community about push-based versus pull-based collection methods for monitoring telemetry. For IoT devices, push-based monitoring can be moreconvenient. But you must consider the trade-offs in the entire stack (including things like the power of the query language, and the efficiency and cost of the time-series storage) when you choose which metrics framework to use.

- In either approach, a remote device might become disconnected from the monitoring system. No effective monitoring can occur if data isn't flowing. Stale and missing metrics can hamper the value of a metric series where you might be calculating rates or other types of values derived over time. When you're monitoring remote devices, it's also important to recognize that variation in timestamps is possible and to ensure the best clock synchronization possible. The following diagram shows a schematic ofremote devices, with centralized monitoring compared to cluster-based monitoring.



- **Fig 2.13 Remote devices with centralized monitoring**

- **Monitoring design patterns**

- When it is determined which systems you're monitoring, you need to think about why you're monitoring. The system you're working with is providing a useful function, and the goal of monitoring is to help ensure that a function or service is performing as intended.

- When monitoring software services, you look for measurements around the performance of that service, such as web request response times. When the service is a physical process such as space heating, electrical generation, or water filtration, you might use devices to instrument that physical process and take measurements of things like engine hours or cycle times. Whether you're using a device as a means solely to instrument a physical process, or whether the device itself is performing a service, you want to have a number of measurements about the device itself.

- Measurements made at the point of instrumentation result in a metric being sent and recorded in the centralized monitoring system. Metrics might be low level (direct and unprocessed) or high level (abstract). Higher-level metrics might be computed from lower-level metrics. One should start by thinking about the high-level metrics you need in order to ensure delivery of service. One can then determine which lower-levelmetrics you need to collect in order to support your monitoring goals. Not all metrics are useful, and it's important not to fall into

the trap of measuring things just because you can, or because they look impressive (so called "vanity metrics").

- **Good metrics have the following characteristics:**

- They're actionable. They inform those who operate or revise the service when they need to change its behaviour.

- They're comparative. They compare the performance of something over time, or between groups of devices whose members are in different location or have different firmware or hardware versions.

- They're understandable and relevant in an operational context. This means that in addition to raw values like totals, they can provide information like ratios and rates.

- They provide information at the right resolution. You can choose how often you sample, how often you report, and how you average, bin, and graph your metrics. These values all need to be chosen in the domain context of the service you're tryingto deliver. For example, providing 1-second reporting on an IoT device's SD card capacity generates a lot of unnecessary detail and volume. And looking only at CPU load averaged per hourwill absorb and hide short, service-crushing spikes in activity. There might be periods of troubleshooting where you dial up the fidelity of metrics for better diagnostics. But the baseline resolution should be appropriate for what you need in order to meet your monitoring needs.

- They illuminate the difference between symptoms, causes, and correlations across what you're measuring. Some measurements are leading indicators of a problem, and you might want to build alerting on those. Other measurements are lagging indicators and help you understand what has happened; these measurements are oftenused for exploratory analysis.