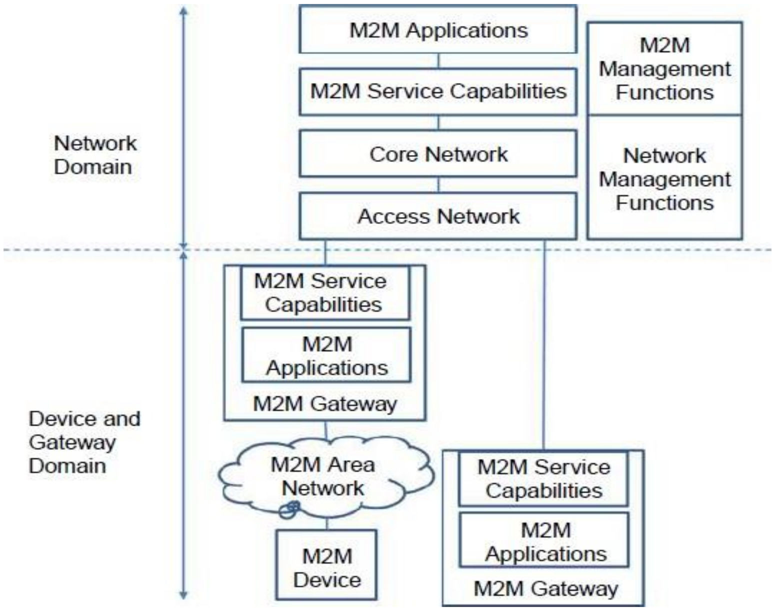
UNIT II: IoT ARCHITECTURE
M2M high- IETF architecture for IoT - OGC architecture - IoT reference model - Domain model - information model - functional model - communication model - IoT reference architecture.

ETSI M2M high-level architecture

- This high-level architecture is a combination of both a functional and topological view showing some functional groups (FG) clearly associated with pieces of physical infrastructure (e.g. M2M Devices, Gateways).
- There are two main domains, a network domain and a device and gateway domain.
- The boundary between these conceptually separated domains is the topological border between the physical devices and gateways and the physical communication infrastructure (Access network).



The Device and Gateway Domain contains the following functional/topological entities:

- **M2M Device:**
  - This is the device of interest for an M2M scenario, for example, a device with a temperature sensor.
  - An M2M Device contains M2M Applications and M2M Service Capabilities.
  - An M2M device connects to the Network Domain either directly or through an M2M Gateway:

    • **Direct connection**: The M2M Device is capable of performing registration, authentication, authorization, management, and provisioning to the Network Domain. Direct connection also means that the M2M device contains the appropriate physical layer to be able to communicate with the Access Network.

    • **Through one or more M2M Gateway**: M2M device does not have the appropriate physical layer, compatible with the Access Network technology, and therefore it needs a network domain proxy. Moreover, a number of M2M devices may form their own local M2M Area Network that typically employs a different networking technology from the Access Network. The M2M Gateway acts as a proxy for the Network Domain and performs the procedures of authentication, authorization, management, and provisioning. An M2M Device could connect through multiple M2M Gateways.

- **M2M Area Network:**
  - This is a local area network (LAN) or a Personal Area Network (PAN) and provides connectivity between M2M Devices and M2M Gateways.
  - Typical networking technologies are IEEE 802.15.1 (Bluetooth), IEEE 802.15.4 (ZigBee, IETF 6LoWPAN/ROLL/CoRE), MBUS, KNX (wired or wireless) PLC, etc.
- **M2M Gateway:**
  - The device that provides connectivity for M2M Devices in an M2M Area Network towards the Network Domain.
  - The M2M Gateway contains M2M Applications and M2M Service Capabilities.
  - The M2M Gateway may also provide services to other legacy devices that are not visible to the Network Domain.

The Network Domain contains the following functional/topological entities:

- **Access Network:**
  - This is the network that allows the devices in the Device and Gateway Domain to communicate with the Core Network.
  - Example Access Network Technologies are fixed (xDSL, HFC) and wireless (Satellite, GERAN, UTRAN, E-UTRAN W-LAN, WiMAX).
- **Core Network:**
  - Examples of Core Networks are 3GPP Core Network and ETSI TISPAN Core Network. It provides the following functions:
    - IP connectivity.
    - Service and Network control.
    - Interconnection with other networks.
    - Roaming.
- **M2M Service Capabilities:**
  - These are functions exposed to different M2M Applications through a set of open interfaces.
  - These functions use underlying Core Network functions, and their objective is to abstract the network functions for the sake of simpler applications.
- **M2M Applications:**
  - These are the specific M2M applications (e.g. smart metering) that utilize the M2M Service Capabilities through the open interfaces.
- **Network Management Functions:**

  - These are all the necessary functions to manage the Access and Core Network (e.g. Provisioning, Fault Management, etc.).

- **M2M Management Functions:**
  - These are the necessary functions required to manage the M2M Service Capabilities on the Network Domain.
  - There are two M2M Management functions:

    • **M2M Service Bootstrap Function (MSBF):** The MSBF facilitates the bootstrapping of permanent M2M service layer security credentials in the M2M Device or Gateway and the M2M Service Capabilities in the Network Domain.

    • **M2M Authentication Server (MAS):** This is the safe execution environment where permanent security credentials such as the M2M Root Key are stored.

- The most relevant entities in the ETSI M2M architecture are the M2M Nodes and M2M Applications.
- An M2M Node can be a Device M2M, Gateway M2M, or Network M2M Node.
- An M2M Application is the main application logic that uses the Service Capabilities to achieve the M2M system requirements.

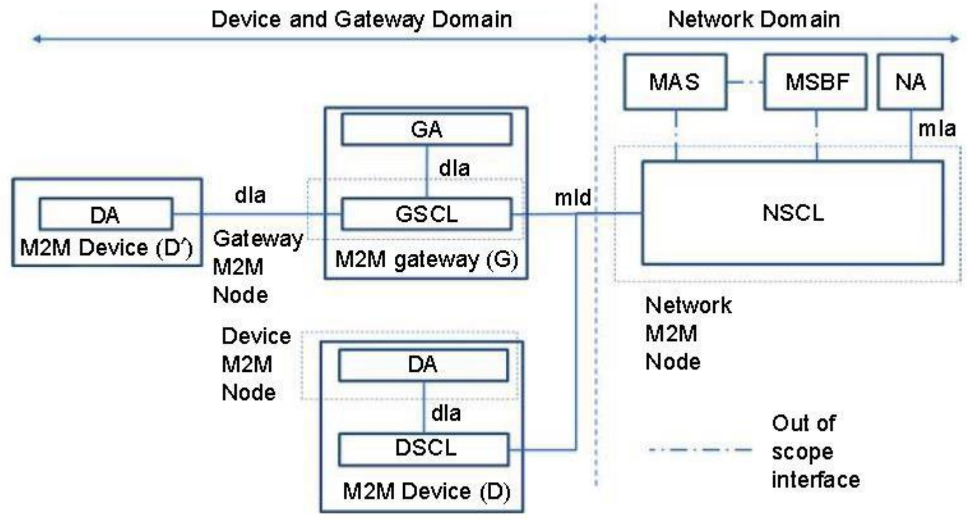


FIGURE 6.2

M2M Service Capabilities, M2M Nodes and Open Interfaces.

- The application logic can be deployed on a Device (Device Application, DA), Gateway (Gateway Application, GA) or Network (Network Application, NA).
- The SCL (Service Capability Layer) is a collection of functions that are exposed through the open interfaces or reference points mIa, dIa, and mId (ETSI M2M TC 2013b).
- Because the main topological entities that SCL can deploy are the Device, Gateway, and Network Domain, there are three types of SCL: DSCL (Device Service Capabilities

Layer), GSCL (Gateway Service Capabilities Layer), and NSCL (Network Service Capabilities Layer).

- SCL functions utilize underlying networking capabilities through technology-specific interfaces.

IETF architecture for IoT

- Internet Engineering Task Force architecture

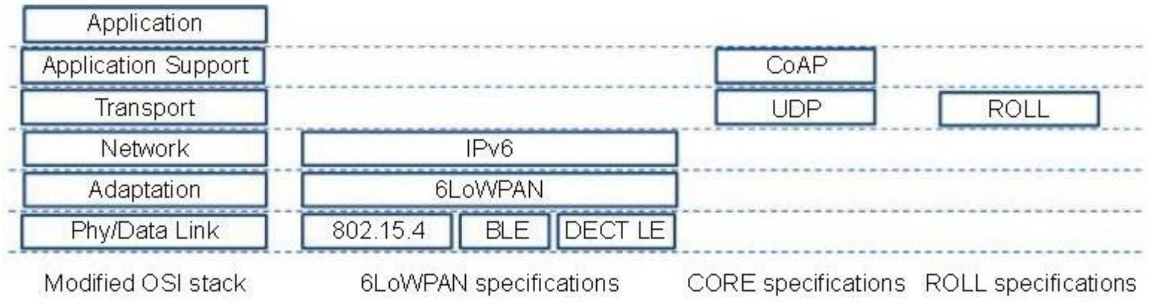


FIGURE 6.7

IETF Working Groups and Specification Scope.

- 6LoWPAN (IPv6 over Low-power WPAN), CoRE (Constrained RESTful Environments), and ROLL (Routing Over Low power and Lossy networks).
- Each set of specifications makes an attempt to address a different part of the communication stack of a constrained device.
- One layer called Application Support which includes the Presentation and Session Layers combined. one intermediate layer is introduced: the Adaptation Layer
- It positioned between the Physical/Data Link and the Network Layer and whose main function is to adapt the Network Layer packets to Phy/Link layer packets among others.
- An example of an adaptation layer is the 6LoWPAN layer designed to adapt IPv6 packets to IEEE 8021.5.4/Bluetooth Low Energy (BLE)/DECT Low Energy packets.
- An example of an Application Support Layer is IETF Constrained Application Protocol (CoAP), which provides reliability and RESTful operation support to applications; however, it does not describe the specific names of resources a node should host.
- The IETF CoAP draft specification describes the Transport and Application Support Layers, which essentially defines the transport packet formats, reliability support on top of UDP, and a RESTful application protocol with GET/PUT/POST/DELETE methods similar to HTTP with CoAP clients operating on CoAP server resources.
- A CoAP server is just a logical protocol entity, and the name “server” does not necessarily

imply that its functionality is deployed on a very powerful machine; a CoAP server can be hosted on a constrained device.

- The CoRE Link Format specification describes a discovery method for the CoAP resources of a CoAP server.
- For example, a CoAP client sending a request with the GET method to a specific well defined server resource (./well-known/core) should receive a response with a list of CoAP resources and some of their capabilities (e.g. resource type, interface type).
- The CoRE interface specification describes interface types and corresponding expected behavior of the RESTful methods.
- The IETF stack for IoT does not currently include any specifications that are similar to the profile specifications of other IoT technologies such as ZigBee
- Profile specification means a document that describes a list of profile names and their mappings to specific protocol stack behavior, specific information model, and specific serialization of this information model over the relevant communication medium.
- An example of a profile specification excerpt would mandate that an exemplary "Temperature" profile:
  (a) should support a resource called /temp,
  (b) the resource /temp must respond to a GET method request from a client, and
  (c) the response to a GET method request shall be a temperature value in degrees Celsius formatted as a text string with the format ",temperature value encoded in a decimal number ._C"
- A Resource Directory is a CoAP server resource (/rd) that maintains a list of resources, their corresponding server contact information (e.g. IP addresses or fully qualified domain name, or FQDN), their type, interface, and other information similar to the information that the CoRE Link Format document specifies.

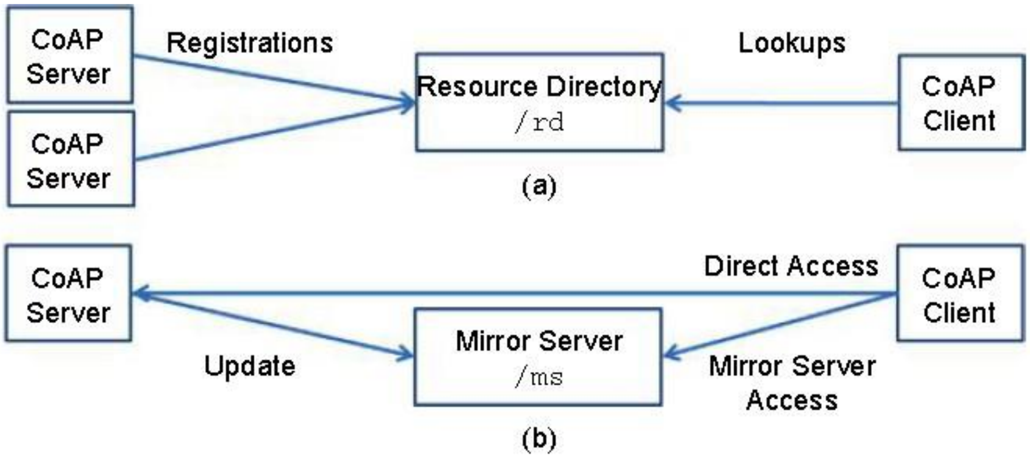


FIGURE 6.8

IETF CoRE Functional Components: (a) Resource Directory, (b) Mirror Server.

- An RD plays the role of a rendezvous mechanism for CoAP Server resource descriptions, in other words, for devices to publish the descriptions of the available resources and for CoAP clients to locate resources that satisfy certain criteria such as specific resource types. (e.g. temperature sensor resource type).
- Resource Directory is a rendezvous mechanism for CoAP Server resource descriptions, a Mirror Server is a rendezvous mechanism for CoAP Server resource presentations.
- A Mirror Server is a CoAP Server resource (/ms) that maintains a list of resources and their cached representations (Figure 6.8b).
- A CoAP Server registers its resources to the Mirror Server, and upon registration a new mirror server resource is created on the Mirror Server with a container (mirror representation) for the original server representation.
- The original CoAP Server updates the mirror representation either periodically or when the representation changes.
- A CoAP Client that retrieves the mirror representation receives the latest updated representation from the original CoAP Server. The Mirror Server is useful when the CoAP Server is not always available for direct access.
- An example of such a CoAP Server is one that resides on a real device whose communication capabilities are turned off in order to preserve energy, e.g. battery-powered radio devices whose radio and/or processor goes to sleep mode.
- Typically, a Mirror Server is hosted on a device or machine that is always available.
- The IETF CoRE workgroup has included the fundamentals of a mapping process between HTTP and CoAP in the IETF CoAP specification as well as a set of guidelines for the interworking between HTTP and CoAP.
- The main is the different transport protocols used by the HTTP and CoAP: HTTP uses TCP while CoAP uses UDP.
- The guidelines focus more on the HTTP-to-CoAP proxy and recommend addressing schemes (e.g. how to map a CoAP resource address to an HTTP address), mapping between HTTP and CoAP response codes, mapping between different media types carried in the HTTP/CoAP payloads, etc.
- HTTP Client sends an HTTP request to a CoAP server (Figure 6.9a) through a Gateway Device hosting an HTTP-CoAP Cross Proxy.
- The Gateway Device connects to the Internet via an Ethernet cable using a LAN, and on the CoAP side the CoAP server resides on a Sensor/Actuator (SAN) based on the IEEE 802.15.4 PHY/MAC.
- The HTTP request needs to include two addresses, one for reaching the Cross Proxy and one for reaching the specific CoAP Server in the SAN.
- The request is in plain text format and contains the method (GET). It traverses the IPv4 stack of the client, reaches the gateway, traverses the IPv4 stack of the gateway and reaches the Cross proxy.

- The request is translated to a CoAP request (binary format) with a destination CoAP resource coap://s.coap.example.com/foo, and it is dispatched in the CoAP stack of the gateway, which sends it over the SAN to the end device.
- A response is sent from the end device and follows the reverse path in the SAN in order to reach the gateway.
- The Cross proxy translates the CoAP response code to the corresponding HTTP code, transforms the included media, creates the HTTP response, and dispatches it to the HTTP client.

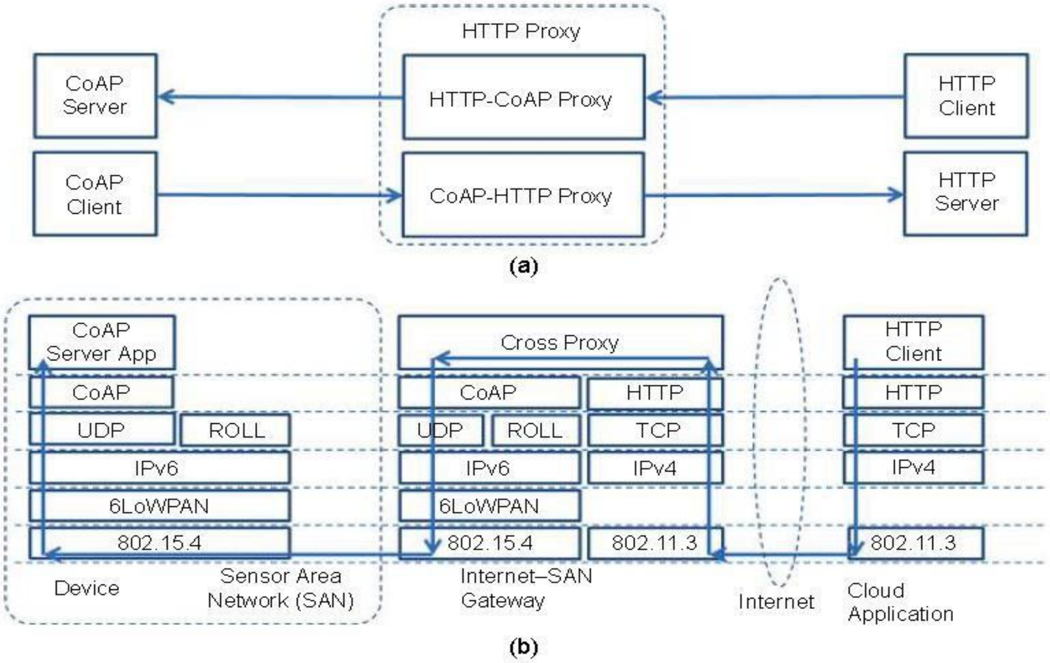


FIGURE 6.9

IETF CoRE HTTP Proxy: (a) possible configurations, (b) example layer interaction upon a request from a HTTP Client to a CoAP Server via a HTTP Proxy.

Open Geospatial Consortium architecture

- The Open Geospatial Consortium (OGC 2013) is an international industry consortium of a few hundred companies, government agencies, and universities that develops publicly available standards that provide geographical information support to the Web, and wireless and location-based services.
- OGC includes, among other working groups, the Sensor Web Enablement (SWE) (OGC SWE 2013) domain working group, which develops standards for sensor system models (e.g. Sensor Model Language, or SensorML), sensor information models (Observations &

Measurements, or O&M), and sensor services that follow the Service-Oriented Architecture (SOA) paradigm.

- The functionality that is targeted by OGC SWE includes:
  - Discovery of sensor systems and observations that meet an application's criteria.
  - Discovery of a sensor's capabilities and quality of measurements.
  - Retrieval of real-time or time-series observations in standard encodings.
  - Tasking of sensors to acquire observations.
  - Subscription to, and publishing of, alerts to be issued by sensors or sensor services based upon certain criteria.
- OGC SWE includes the following standards:
  - **SensorML and Transducer Model Language (TML),** which include a model and an XML schema for describing sensor and actuator systems and processes; for example, a system that contains a temperature sensor measuring temperature in Celsius, which also involves a process for converting this measurement to a measurement with Fahrenheit units.
  - **Observations and Measurements (O&M)**, which is a model and an XML schema for describing the observations and measurements for a sensor (Observations and Measurements, O&M).
  - **SWE Common Data model** for describing low-level data models (e.g. serialization in XML) in the messages exchanged between OGC SWE functional entities.
  - **Sensor Observation Service (SOS),** which is a service for requesting, filtering, and retrieving observations and sensor system information. This is the intermediary between a client and an observation repository or near real-time sensor channel.
  - **Sensor Planning Service (SPS),** which is a service for applications requesting a user-defined sensor observations and measurements acquisition. This is the intermediary between the application and a sensor collection system.
  - **PUCK**, which defines a protocol for retrieving sensor metadata for serial port (RS232) or Ethernet-enabled sensor devices.

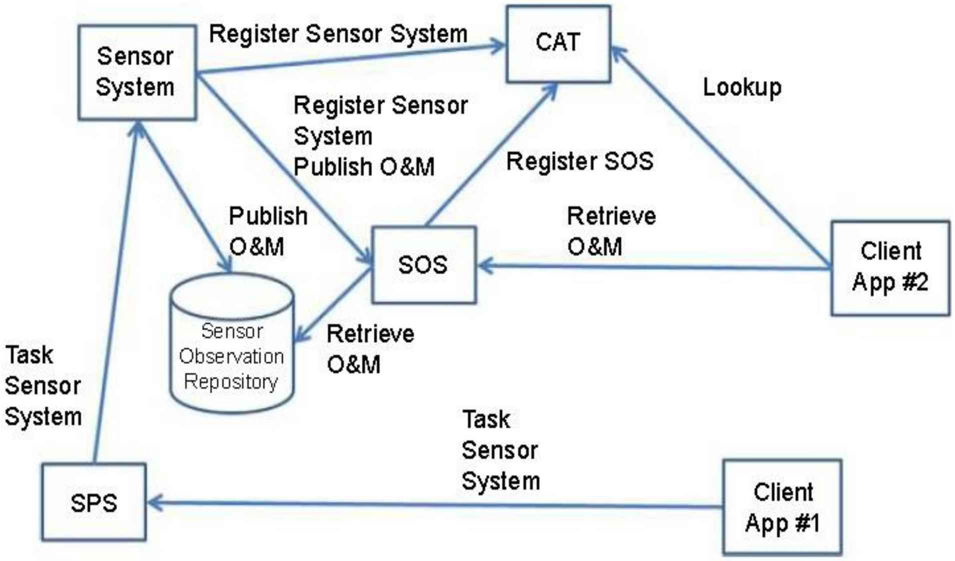


FIGURE 6.10

OGC functional architecture and interactions.

- OGC follows the SOA paradigm, there is a registry (CAT) that maintains the descriptions of the existing OGC services, including the Sensor Observation and Sensor Planning Services.
- Upon installation the sensor system using the PUCK protocol retrieves the SensorML description of sensors and processes, and registers them with the Catalog so as to enable the discovery of the sensors and processes by client applications.
- The Sensor System also registers to the SOS and the SOS registers to the Catalog.
- A client application #1 requests from the Sensor Planning Service that the Sensor System be tasked to sample its sensors every 10 seconds and publish the measurements using O&M and the SWE Common Data model to the SOS.
- Another client application #2 looks up the Catalog, aiming at locating an SOS for retrieving the measurements from the Sensor System.
- The application receives the contact information of the SOS and requests from the sensor observations from the specific sensor system from the SOS.
- As a response, the measurements from the sensor system using O&M and the SWE Common Data model are dispatched to the client application #2.
- The main objective of the OGC standards is to enable data, information, and service interoperability.

IOT Reference Model

- The IoT Reference Model aims at establishing a common grounding and a common language for IoT architectures and IoT systems.
- A reference model describes the domain using a number of sub-models (Figure 7.1).
- The domain model of an architecture model captures the main concepts or entities in the domain, the domain model adds descriptions about the relationship between the concepts.
- These concepts and relationships serve the basis for the development of an information model because a working system needs to capture and process information about its main entities and their interactions.
- A working system that captures and operates on the domain and information model contains concepts and entities of its own, and these need to be described in a separate model, the functional model.
- An M2M and IoT system contain communicating entities, and therefore the corresponding communication model needs to capture the communication interactions of these entities.

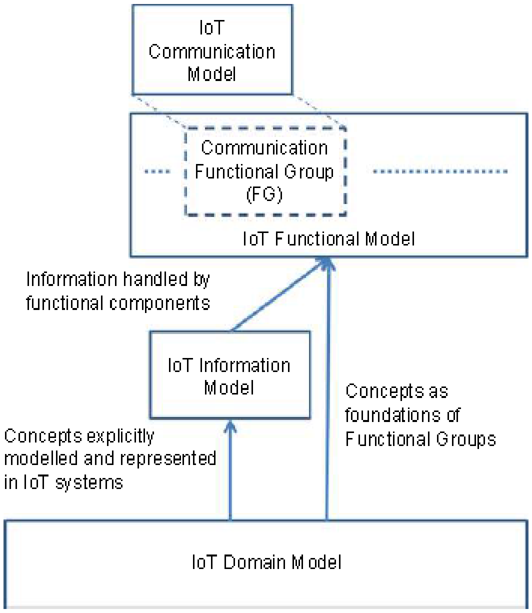

IoT Communication Model
Communication Functional Group (FG)
IoT Functional Model
Information handled by functional components
IoT Information Model
Concepts as foundations of Functional Groups
Concepts explicitly modelled and represented in IoT systems
IoT Domain Model

FIGURE 7.1

IoT Reference Model.

- The foundation of the IoT Reference Model is the IoT Domain Model, which introduces the main concepts of the Internet of Things like Devices, IoT Services and *Virtual Entities* (VE), and it also introduces relations between these concepts.
- Based on the IoT Domain Model, the IoT Information Model has been developed. It defines the structure (e.g. relations, attributes) of IoT related information in an IoT system on a conceptual level without discussing how it would be represented.
- The information pertaining to those concepts of the IoT Domain Model is modelled, which is explicitly gathered, stored and processed in an IoT system, e.g. information about Devices, IoT Services and Virtual Entities.

- The IoT Functional Model identifies groups of functionalities, of which most are grounded in key concepts of the IoT Domain Model.
- A number of these *Functionality Groups* (FG) build on each other, following the relations identified in the IoT Domain Model.
- The Functionality Groups provide the functionalities for interacting with the instances of these concepts or managing the information related to the concepts, e.g. information about Virtual Entities or descriptions of IoT Services.
- The functionalities of the FGs that manage information use the IoT Information Model as the basis for structuring their information.
- A key functionality in any distributed computer system is the communication between the different components.
- The IoT Communication Model introduces concepts for handling the complexity of communication in heterogeneous IoT environments. Communication also constitutes one FG in the IoT Functional Model.

IoT domain model

- Domain model as a description of concepts belonging to a particular area of interest.
- The domain model also defines basic attributes of these concepts, such as name and identifier.
- The domain model defines relationships between concepts, for instance "*Services* expose *Resources*".
- Domain models also help to facilitate the exchange of data between domains.
- The main purpose of a domain model is to generate a common understanding of the target domain in question.
- The domain model is an important part of any reference model since it includes a definition of the main abstract concepts (abstractions), their responsibilities, and their relationships.
- The domain model captures the basic attributes of the main concepts and the relationship between these concepts.

Model notation and semantics

- Class diagrams in order to present the relationships between the main concepts of the IoT domain model.
- The Class diagrams consist of boxes that represent the different classes of the model connected with each other through typically continuous lines or arrows, which represent relationships between the respective classes.
- Each class is a descriptor of a set of objects that have similar structure, behavior, and relationships.
- A class contains a name and a set of attributes and operations.
- Notation-wise this is represented as a box with two compartments, one containing the class name and the other containing the attributes.
- The Generalization/Specialization relationship is represented by an arrow with a solid line and a hollow triangle head.
- Depending on the starting point of the arrow, the relationship can be viewed as a generalization or specialization.

- For example, in Figure 7.4, Class A is a general case of Class B or Class B is special case or specialization of Class A.
- Generalization is also called an "is-a" relationship. For example, in Figure 7.4 Class B "is-a" Class A. A specialized class/subclass/child class inherits the attributes and the operations from the general/super/parent class, respectively, and also contains its own attributes and operations.
- The Aggregation relationship is represented by a line with a hollow diamond in one end and represents a whole-part relationship or a containment relationship and is often called a "has-a" relationship.

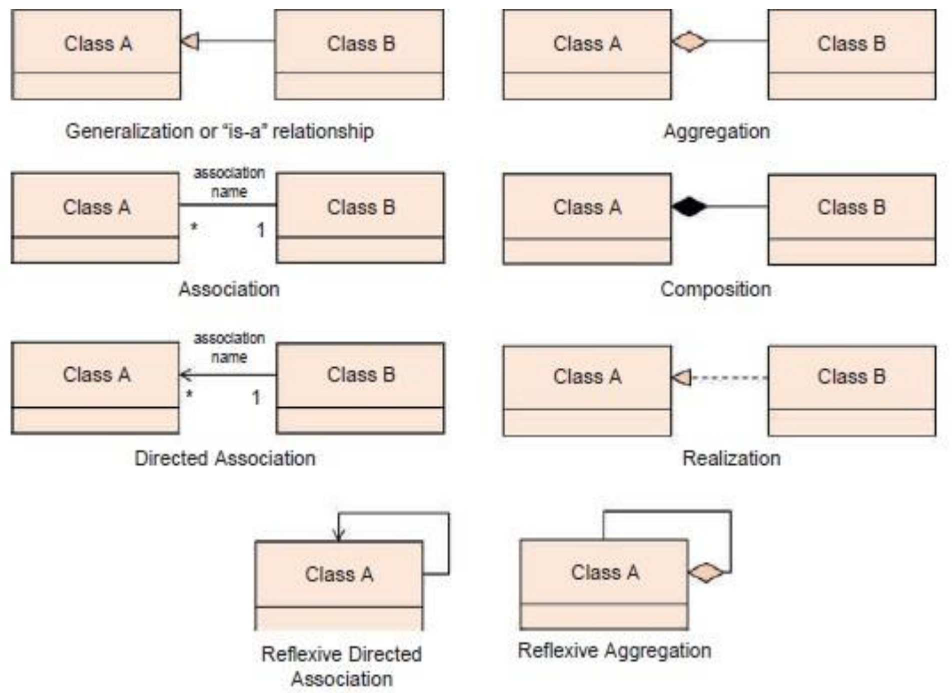


FIGURE 7.4

UML Class diagram main modeling concepts.

- The class that touches the hollow diamond is the whole class while the other class is the part class.
- For example, in Figure 7.4, class B represents a part of the whole Class A, or in other words, an object of Class A "contains" or "has-a" object of Class B.
- When the line with the hollow diamond starts and ends in the same class, then this relationship of one class to itself is called Reflexive Aggregation, and it denotes that objects of a class (e.g. Class A in Figure 7.4) contain objects of the same class.
- The Composition relationship is represented by a line with a solid black diamond in one end, and also represents a whole-part relationship or a containment relationship.
- The class that touches the solid black diamond is the whole class while the other class is the part class. For example, in Figure 7.4, Class B is part of Class A. Composition and Aggregation are very similar, with the difference being the coincident lifetime to the objects of classes related with composition.

- In other words, if an object of Class B is part of an object of Class A (composition), when the object of Class A disappears, the object of Class B also disappears.
- A plain line without arrowheads or diamonds represents the Association relationship.
- Directed Association that is represented with a line with a normal arrowhead.
- An Association (Directed or not) contains an explicit association name. The Directed Association implies navigability from a Class B to a Class A in Figure 7.4.
- Navigability means that objects of Class B have the necessary attributes to know that they relate to objects of Class A while the reverse is not true: objects of Class A can exist without having references to objects of Class B.
- When the arrow starts and ends at the same class, then the class is associated to itself with a Reflexive Directed Association, which means that an object of this class is associated with objects of the same class with the specific named association.
- An arrow with a hollow triangle head and a dashed line represents the Realization relationship. This relationship represents a association between the class that specifies the functionality and the class that realizes the functionality.
- For example, Class A in Figure 7.4 specifies the functionality while Class B realizes it. Contain multiplicity information such as numbers (e.g "1"), ranges (e.g. "0_1", open ranges "1. . ._"), etc. in one or the other end of the relationship line/arrow.
- These multiplicities denote the potential number of class objects that are related to the other class object.
- For example, in Figure 7.4, a plain association called "association name," relates one (1) object of Class B with zero (0) or more objects from Class A. An asterisk "_" denotes zero (0) or more

Main concepts

- The IoT is a support infrastructure for enabling objects and places in the physical world to have a corresponding representation in the digital world.

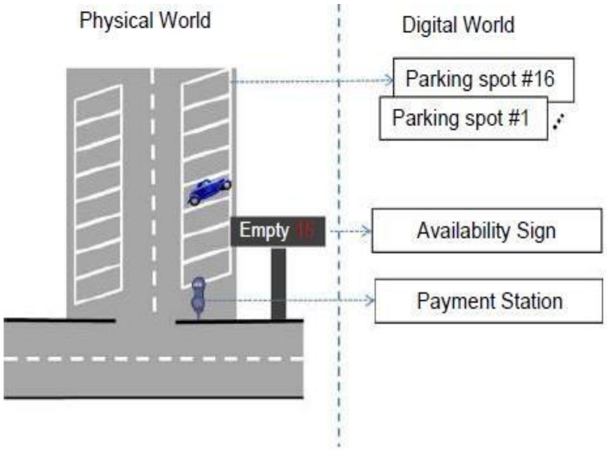


FIGURE 7.5

Physical vs. Virtual World.

- Monitoring a parking lot with 16 parking spots.
- The parking lot includes a payment station for drivers to pay for the parking spot after they park their cars.

- The parking lot also includes an electronic road sign on the side of the street that shows in real-time the number of empty spots.
- Frequent customers also download a smart phone application that informs them about the availability of a parking spot before they even drive on the street where the parking lot is located.
- The relevant physical objects as well as their properties need to be captured and translated to digital objects such as variables, counters, or database objects so that software can operate on these objects and achieve the desired effect.
- In the digital world, a parking spot is a variable with a binary value (“available” or “occupied”).
- The parking lot payment station also needs to be represented in the digital world in order to check if a recently parked car owner actually paid the parking fee.
- Internet serves a rather virtual world of content and services (although these services are hosted on real physical machines)
- IoT is all about interaction through the Internet with physical Things.
- As interaction with the physical world is the key for the IoT; it needs to be captured in the domain model.
- A User and a Physical Entity are two concepts that belong to the domain model.
- A User can be a Human User, and the interaction can be physical (e.g. parking the car in the parking lot). The physical interaction is the result of the intention of the human to achieve a certain goal (e.g. park the car).
- The objects, places, and things represented as Physical Entities are the same as Assets.
- A Physical Entity is represented in the digital world as a Virtual Entity.
- A Virtual Entity can be a database entry, a geographical model (mainly for places), an image or avatar, or any other Digital Artifact.
- Each Virtual Entity also has a unique identifier for making it addressable among other Digital Artifacts.
- A Virtual Entity representation contains several attributes that correspond to the Physical Entity current state (e.g. the parking spot availability).
- The Virtual Entity representation and the Physical Entity actual state should be synchronized.
- For example, a remotely controlled light (Physical Entity) represented by a memory location (Virtual Entity) in an application could be switched on/off by the User by changing the Virtual Entity representation, or in other words writing a value in the corresponding memory location.
- A Digital Artifact is an artifact of the digital world, and can be passive (e.g. a database entry) or active (e.g. application software).
- The model captures human-to-machine, application (active digital artifact)-to-machine, and M2M interaction when a digital artifact, and thus a User, interacts with a Device that is a Physical Entity.
- Physical Entities or their surrounding environment needs to be instrumented with certain kinds of Devices, or certain Devices need to be embedded/attached to the environment.
- IoT Domain Model, three kinds of Device types are the most important:

1. **Sensors:**
   - These are simple or complex Devices that typically involve a transducer that converts physical properties such as temperature into electrical signals.
   - These Devices include the necessary conversion of analog electrical signals into digital signals, e.g. A video camera can be example of a complex sensor that could detect and recognize people.
2. **Actuators:**
   - These are also simple or complex Devices that involve a transducer that converts electrical signals to a change in a physical property (e.g. turn on a switch or move a motor).
   - These Devices also include potential communication capabilities, storage of intermediate commands, processing, and conversion of digital signals to analog electrical signals.
3. **Tags:**
   - Tags in general identify the Physical Entity that they are attached to.
   - In reality, tags can be Devices or Physical Entities but not both, as the domain model shows.
   - An example of a Tag as a Device is a Radio Frequency Identification (RFID) tag, while a tag as a Physical Entity is a paper-printed immutable barcode or Quick Response (QR) code.
   - Either electronic Devices or a paper-printed entity tag contains a unique identification that can be read by optical means (bar codes or QR codes) or radio signals (RFID tags).
   - The reader Device operating on a tag is typically a sensor, and sometimes a sensor and an actuator combined in the case of writable RFID tags.

- Any type of IoT Device needs to
  (a) have energy reserves (e.g. a battery)
  (b) be connected to the power grid
  (c) perform energy scavenging (e.g. converting solar radiation to energy).
- The Device communication, processing and storage, and energy reserve capabilities determine several design decisions such as if the resources should be on-Device or not.
- Resources are software components that provide data for, or are endpoints for, controlling Physical Entities.
- Resources can be of two types, on-Device resources and Network Resources.
- An on-Device Resource is typically hosted on the Device itself and provides information, or is the control point for the Physical Entities that the Device itself is attached to.
- An example is a temperature sensor on a temperature node deployed in a room that hosts a software component that responds to queries about the temperature of the room.
- The Network Resources are software components hosted somewhere in the network or cloud.

- A Virtual Entity is associated with potentially several Resources that provide information or control of the Physical Entity represented by this Virtual Entity.
- Resources can be of several types: sensor resources that provide sensor data, actuator resources that provide actuation capabilities or actuator state (e.g. "on"/"off"), processing resources that get sensor data as input and provide processed data as output, storage resources that store data related to Physical Entities, and tag resources that provide identification data of Physical Entities.
- IoT Services can be classified into three main classes according to their level of abstraction:

  1. Resource-Level Services typically expose the functionality of a Device by exposing the on-Device Resources. In addition, these services typically handle quality aspects such as security, availability, and performance issues. An example of such a Network Resource is a historical database of measurements of a specific resource on a specific Device.

  2. Virtual Entity-Level Services provide information or interaction capabilities about Virtual Entities, and as a result the Service interfaces typically include an identity of the Virtual Entity.

  3. Integrated Services are the compositions of Resource-Level and Virtual Entity-Level services, or any combination of both service classes.

Further considerations

- Identification of Physical Entities is important in an IoT system in order for any User to interact with the physical world though the digital world.
- Two ways to describe :

  (a) primary identification that uses natural features of a Physical Entity, and

  (b) secondary identification when using tags or labels attached to the Physical Entity.
- Both types of identification are modeled in the IoT Domain Model.
- Extracting natural features can be performed by a camera Device (Sensor) and relevant Resources that produce a set of features for specific Physical Entities.
- In physical spaces, a GPS Device or another type of location Device (e.g. an indoor location Device) can also be used to record the GPS coordinates of the space occupied by the Physical Entity.
- With respect to secondary identification, tags or labels attached to Physical Entities are modeled in the IoT Domain model, and there are relevant RFID or barcode technologies to realize such identification mechanisms.
- Apart from identification, location and time information are important for the annotation of the information collected for specific Physical Entities and represented in Virtual Entities.
- Information without one or the other (i.e. location or time) is practically useless apart from the case of Body Area Networks (BAN, networks of sensors attached to a human

body for live capture of vital signals, e.g. heart rate); that location is basically fixed and associated with the identification of the Human User.

- Nevertheless, in such cases, sometimes the location of the whole BAN or Human User is important for correlation purposes (e.g. upon moving outdoors, the Human User heart rate increases in order to compensate for the lower temperature than indoors).
- Therefore, the location, and often the timestamp of location, for the Virtual Entity can be modeled as an attribute of the Virtual Entity that could be obtained by location sensing resources (e.g. GPS or indoor location systems).

Communication model

- The communication model for an IoT Reference Model consists of the identification of the endpoints of interactions, traffic patterns (e.g. unicast vs. multicast), and general properties of the underlying technologies used for enabling such interactions.
- It is used to identification of the endpoints of the communication paths.
- The potential communicating endpoints or entities are the Users, Resources, and Devices from the IoT Domain Model.
- Users include Human Users and Active Digital Artifacts (Services, internal system components, external applications).
- Devices with a Human_Machine Interface mediate the interactions between a Human User and the physical world (e.g. keyboards, mice, pens, touch screens, buttons, microphones, cameras, eye tracking, and brain wave interfaces, etc.), and therefore the Human User is not a communication model endpoint.
- The User (Active Digital Artifact, Service)-to-Service interactions include the User-to-Service and Service-to-Service interactions as well as the Service_Resource_Device interactions.
- The User-to-Service and Service-to-Service communication is typically based on Internet protocols and one or both Services are hosted in Service-to-Service interactions on constrained/low-end Devices such as embedded systems.
- The communication model for these interactions includes several types of gateways (e.g. network, application layer gateways) to bridge between two or more disparate communication technologies.
- The Devices may be so constrained that they cannot host the Services, while the Resources could be hosted or not depending on the Device capabilities.
- This inability of the Device to host Resources or Services results in moving the corresponding Resources and/or Services out of the Device and into more powerful Devices or machines in the cloud.
- Then the Resource-to-Device or the Service-to-Resource communication needs to involve multiple types of communication stacks.

Functional model

- ➢ The IoT Functional Model aims at describing mainly the Functional Groups (FG) and their interaction with the ARM.
- ➢ Functional View of a Reference Architecture describes the functional components of an FG, interfaces, and interactions between the components.
- ➢ The Functional View is typically derived from the Functional Model in conjunction with high level requirements.

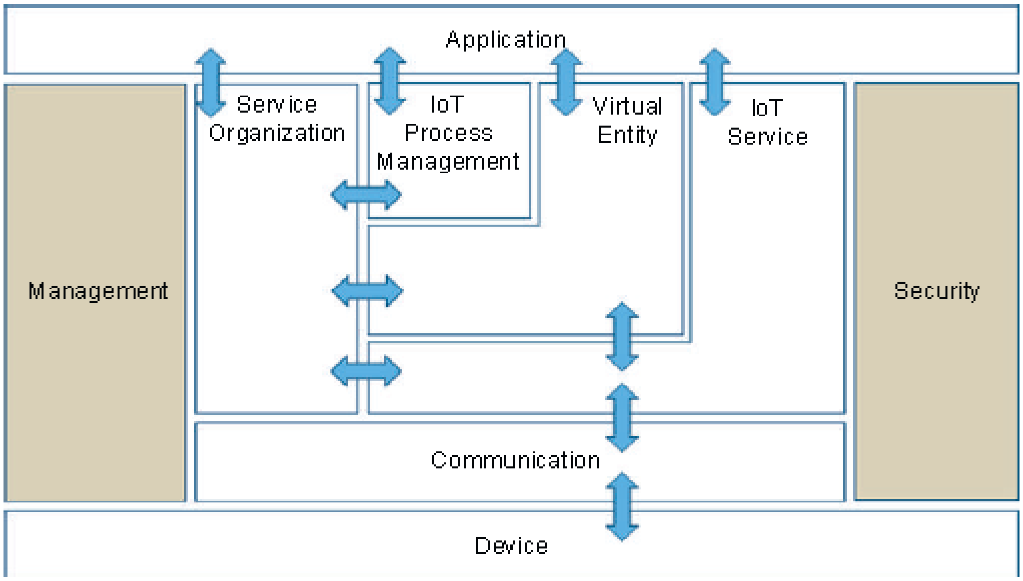


FIGURE 7.11

IoT-A Functional Model.

- ➢ The Application, Virtual Entity, IoT Service, and Device FGs are generated by starting from the User, Virtual Entity, Resource, Service, and Device classes from the IoT Domain Model.
- ➢ The need to compose simple IoT services in order to create more complex ones, as well as the need to integrate IoT services (simple or complex) with existing Information and Communications Technology (ICT) infrastructure, is the main driver behind the introduction of the Service Organization and IoT Process Management FGs respectively.
- ➢ All the above-mentioned FGs need to be supported by management and security functionality captured by the corresponding FGs.

Device functional group

➢ The Device FG contains all the possible functionality hosted by the physical Devices that are used for instrumenting the Physical Entities.

➢ This Device functionality includes sensing, actuation, processing, storage, and identification components, the sophistication of which depends on the Device capabilities.

Communication functional group

- The Communication FG abstracts all the possible communication mechanisms used by the relevant Devices in an actual system in order to transfer information to the digital world components or other Devices.
- Examples of such functions include wired bus or wireless mesh technologies through which sensor Devices are connected to Internet Gateway Devices.
- Communication technologies used between Applications and other functions such as functions from the IoT Service FG are out of scope because they are the typical Internet technologies.

IoT Service functional group

- The IoT Service FG corresponds mainly to the Service class from the IoT Domain Model, and contains single IoT Services exposed by Resources hosted on Devices or in the Network (e.g. processing or storage Resources).
- Support functions such as directory services, which allow discovery of Services and resolution to Resources, are also part of this FG.

Virtual Entity functional group

- The Virtual Entity FG corresponds to the Virtual Entity class in the IoT Domain Model, and contains the necessary functionality to manage associations between Virtual Entities with themselves as well as associations between Virtual Entities and related IoT Services, i.e. the Association objects for the IoT Information Model.
- Associations between Virtual Entities can be static or dynamic depending on the mobility of the Physical Entities related to the corresponding Virtual Entities.
- An example of a static association between Virtual Entities is the hierarchical inclusion relationship of a building, floor, room/corridor/open space, i.e. a building contains multiple floors that contain rooms, corridors, and open spaces.
- An example of a dynamic association between Virtual Entities is a car moving from one block of a city to another (the car is one Virtual Entity while the city block is another).
- A major difference between IoT Services and Virtual Entity Services is the semantics of the requests and responses to/from these services.
- The parking lot example, the Parking Sensor Service provides as a response only a number “0” or “1” given the identifier of a Loop Sensor (e.g. #11).
- The Virtual Entity Parking Spot #01 responds to a request about its occupancy status as “free.” The IoT Service provides data or information associated to specific Devices or Resources, including limited semantic information (e.g. Parking sensor #11, value5“0”, units 5 none); the Virtual IoT Service provides information with richer semantics (“Parking spot #01 is free”), and is closer to being human-readable and understandable.

IoT Service Organization functional group

- The purpose of the IoT Service Organization FG is to host all functional components that support the composition and orchestration of IoT and Virtual Entity services.
- This FG acts as a service hub between several other functional groups such as the IoT Process Management FG when, for example, service requests from Applications or the IoT Process Management are directed to the Resources implementing the necessary Services.

- Therefore, the Service Organization FG supports the association of Virtual Entities with the related IoT Services, and contains functions for discovery, composition, and choreography of services.
- Simple IoT or Virtual Entity Services can be composed to create more complex services, e.g. a control loop with one Sensor Service and one Actuator service with the objective to control the temperature in a building.
- Choreography is the brokerage of Services so that Services can subscribe to other services in a system.

IoT Process Management functional group

- The IoT Process Management FG is a collection of functionalities that allows smooth integration of IoT-related services (IoT Services, Virtual Entity Services, Composed Services) with the Enterprise (Business) Processes.

Management functional group

- The Management FG includes the necessary functions for enabling fault and performance monitoring of the system, configuration for enabling the system to be flexible to changing User demands, and accounting for enabling subsequent billing for the usage of the system.
- Support functions such as management of ownership, administrative domain, rules and rights of functional components, and information stores are also included in the Management FG.

Security functional group

- The Security FG contains the functional components that ensure the secure operation of the system as well as the management of privacy.
- The Security FG contains components for Authentication of Users (Applications, Humans), Authorization of access to Services by Users, secure communication (ensuring integrity and confidentiality of messages) between entities of the system such as Devices, Services, Applications, assurance of privacy of sensitive information relating to Human Users.
- These include privacy mechanisms such as anonymization of collected data, anonymization of resource and Service accesses (Services cannot deduce which Human User accessed the data), and un-linkability (an outside observer cannot deduce the Human User of a service by observing multiple service requests by the same User).

Application functional group

- The Application FG is just a placeholder that represents all the needed logic for creating an IoT application.
- The applications typically contain custom logic tailored to a specific domain such as a Smart Grid.
- An application can also be a part of a bigger ICT system that employs IoT services such as a supply chain system that uses RFID readers to track the movement of goods within a factory in order to update the Enterprise Resource Planning (ERP) system.

Modular IoT functions

- It is important to note that not all the FGs are needed for a complete actual IoT system.

- The Functional Model, as well as the Functional View of the Reference Architecture, contains a complete map of the potential functionalities for a system realization.
- The functionalities that will eventually be used in an actual system are dependent on the actual system requirements.
- FGs are organized in such a way that more complex functionalities can be built based on simpler ones, thus making the model modular.

Information model

- Information is defined as the enrichment of data (raw values without relevant or usable context) with the right context, so that queries about who, what, where, and when can be answered.
- IoT information model captures the details of a Virtual Entity centric model.

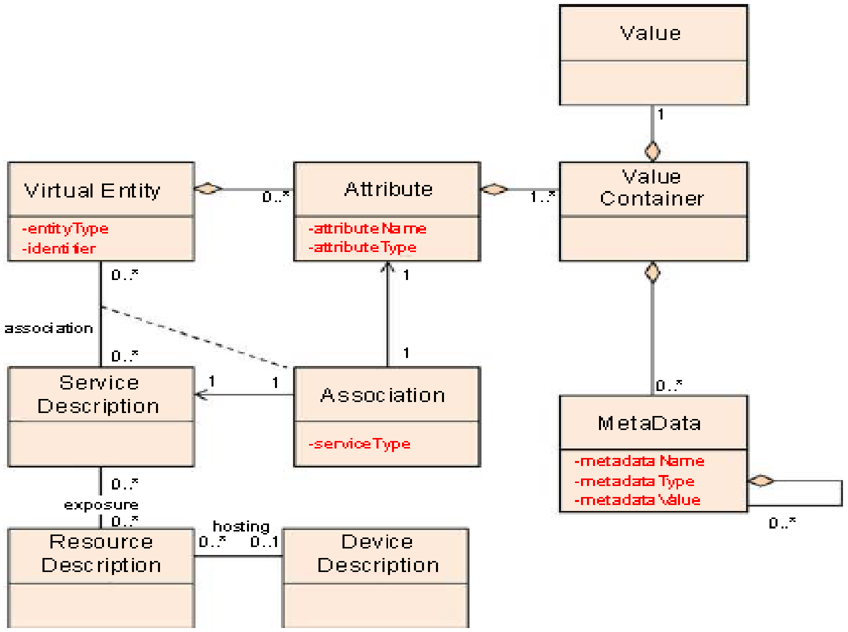


FIGURE 7.8

High-level IoT Information Model.

- Association class in Figure 7.8 contains information about the specific association between a Virtual Entity and a related Service.
- On a high-level, the IoT Information Model maintains the necessary information about Virtual Entities and their properties or attributes.
- These properties/attributes can be static or dynamic and enter into the system in various forms, e.g. by manual data entry or reading a sensor attached to the Virtual Entity.
- Virtual Entity attributes can also be digital synchronized copies of the state of an actuator.
- In the presentation of the high-level IoT information model, we omit the attributes that are not updated by an IoT Device (sensor, tag) or the attributes that do not affect any IoT

Device (actuator, tag), with the exception of essential attributes such as names and identifiers.

- Examples of omitted attributes that could exist in a real implementation are room names and floor numbers, in general, context information that is not directly related to IoT Devices, but that is nevertheless important for an actual system.
- The IoT Information Model describes Virtual Entities and their attributes that have one or more values annotated with meta-information or metadata.
- The attribute values are updated as a result of the associated services to a Virtual Entity.
- The associated services are related to Resources and Devices as seen from the IoT Domain Model.
- A Virtual Entity object contains simple attributes/properties:
  (a) entityType to denote the type of entity, such as a human, car, or room (the entity type can be a reference to concepts of a domain ontology, e.g. a car ontology);
  (b) a unique identifier; and
  (c) zero or more complex attributes of the class Attributes.
- The class Attributes should not be confused with the simple attributes of each class.
- This class Attributes is used as a grouping mechanism for complex attributes of the Virtual Entity.
- Objects of the class Attributes, in turn, contain the simple attributes with the self descriptive names attributeName and attributeType.
- The attribute type is the semantic type of the value (e.g. that the value is a temperature value), and can refer to an ontology such as the NASA quantities and units SWEET ontology (NASA JPL 2011).
- The Attribute class also contains a complex attribute ValueContainer that is a container of the multiple values that an attribute can take.
- The container includes complex attributes of the class Value and the class MetaData.
- The container contains exactly one value and meta-information (modeled as the class MetaData), such as a timestamp, describing this single value.
- Objects of the MetaData class can contain MetaData objects as complex attributes, as well as the simple attributes with the self-descriptive namesmetadataName, metadataType, and metadataValue.
- a Virtual Entity is associated with Resources that expose Services about the specific Virtual Entity.
- This association between a Virtual Entity and its Services is captured in the Information Model with the explicit class called Association.
- Objects of this class capture the relationship between objects of the complex Attribute class (associated with a Virtual Entity) and objects of the Service Description class.
- The class Association describes the relationship between a Virtual Entity and Service Description through the Attribute class, there is a dashed line between Association class and the line between the Virtual Entity and Service Description classes.

- ➢ The attribute serviceType can take two values:

  (a) "INFORMATION," if the associated service is a sensor service (i.e. allows reading of the sensor), or

  (b) "ACTUATION," if the associated serviceis an actuation service (i.e. allows an action executed on an actuator).
- ➢ In both cases, the eventual value of the attribute will be a result of either reading a sensor or controlling an actuator.

Example

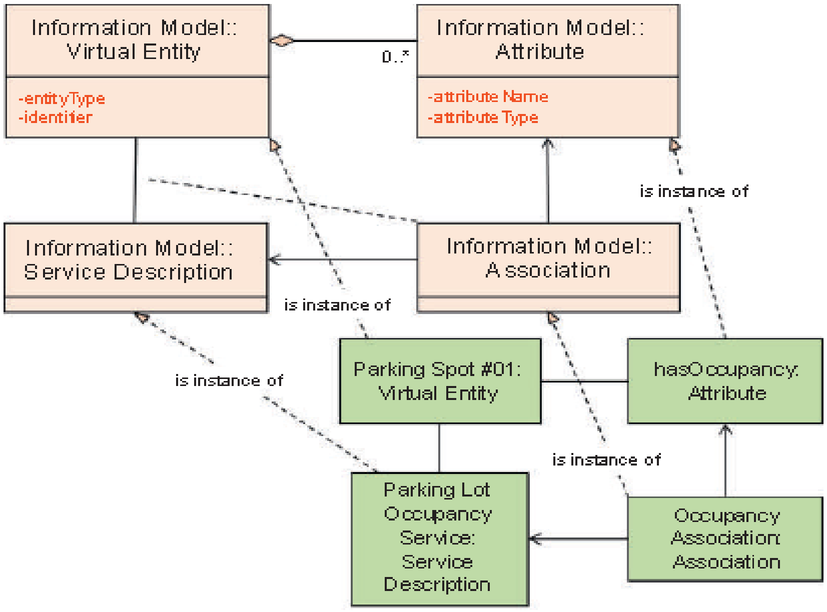


FIGURE 7.9

IoT Information Model example.

- ➢ Not show all the possible Virtual Entities, but only one corresponding to one parking spot.
- ➢ This Virtual Entity is described with one Attribute (among others) called hasOccupancy.
- ➢ This Attribute is associated with the Parking Lot Occupancy Service Description through the Occupancy Association.
- ➢ The Occupancy Association is the explicit expression of the association (line) between the Parking Spot #1 Virtual Entity and the Parking Lot Occupancy Service.
- ➢ The dashed arrows with hollow arrowheads represent the relationship "is instance of" for the information model, as opposed to the Realization relationship for the IoT Domain Model.

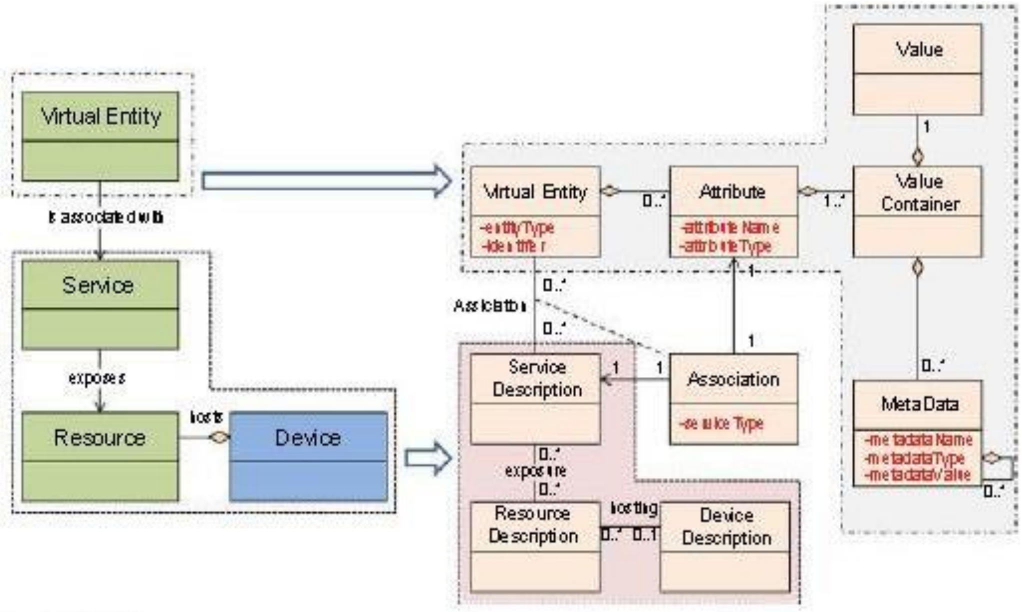


FIGURE 7.10

Relationship between core concepts of IoT Domain Model and IoT Information Model.

- Figure 7.10 presents the relationship between the core concepts of the IoT Domain Model and the IoT Information Model.
- The Information Model captures the Virtual Entity in the Domain Model being the "Thing" in the Internet of Things as several associated classes (Virtual Entity, Attribute, Value, MetaData, Value Container) that basically capture the description of a Virtual Entity and its context.
- The Device, Resource, and Service in the IoT Domain Model are also captured in the IoT Information Model because they are used as representations of the instruments and the digital interfaces for interaction with the Physical Entity associated with the Virtual Entity.
- The Information Model is a very high-level model, and omits certain details that could potentially be required in a concrete architecture and an actual system.
- These details could be derived by specific requirements from specific use cases describing the target actual system.
- Several other attributes or properties that could exist in a Virtual Entity description:

  1. Location and its temporal information are important because Physical Entities represented by Virtual Entities exist in space and time. These properties are extremely important when the interested Physical Entities are mobile (e.g. a moving car).A mobile Physical Entity affects the associations between Attributes and related Services, e.g. a person moving close to a camera (sensor) is associated with the Device, Resource, and Services offered by the camera for as long as she stays within the field of view of the camera. In such cases, the temporal availability of the associations between Attributes and Services need to be captured, as availability denotes also temporal observability of the Virtual Entity.

2. Even non-moving Virtual Entities contain properties that are dynamic with time, and therefore their temporal variations need to be modeled and captured by an information model.

3. Information such as ownership is also important in commercial settings because it may determine access control rules or liability issues. It is important to note that the Attribute class is general enough to capture all the interested properties of a Physical Entity, and thus provides an extensible model whose details can only be specified by the specific actual system in mind.

- The Services in the IoT Domain Model are mapped to the Service Description in the IoT Information Model.
- The Service Description contains the following :

1. Service type, which denotes the type of service, such as Big Web Service or RESTful Web Service. The interfaces of a service are described based on the description language for each service type, for example, Web Application Description Language (WADL) for RESTful Web Services, Web Services Description Language (WSDL) for Big Web Services, Universal Service Description Language (USDL). The interface description includes, among other information, the invocation contact information, e.g. a Uniform Resource Locator (URL).

2. Service area and Service schedule are properties of Services used for specifying the geographical area of interest for a Service and the potential temporal availability of a Service, respectively. For sensing services, the area of interest is equivalent to the observation area, whereas for actuation services the area of interest is the area of operation or impact.

3. Associated resources that the Service exposes.

4. Metadata or semantic information used mainly for service composition. This is information such as the indicator of which resource property is exposed as input or output, whether the execution of the service needs any conditions satisfied before invocation.

- The IoT Information Model also contains Resource descriptions because Resources are associated with Services and Devices in the IoT Domain model.
- **A Resource description contains the following information**:

1. Resource name and identifier for facilitating resource discovery.

2. Resource type, which specifies if the resource is

(a) a sensor resource, which provides sensor readings;

(b) an actuator resource, which provides actuation capabilities (to affect the physical world) and actuator state;

(c) a processor resource, which provides processing of sensor data and output of processed data;

(d) a storage resource, which provides storage of data about a Physical Entity;

(e) a tag resource, which provides identification data for Physical Entities.

3. Free text attributes or tags used for capturing typical manual input such as "fire alarm, ceiling."

4. Indicator of whether the resource is an on-Device resour ce or network resource.

5. Location information about the Device that hosts this resource in case of an on-Device resource.

6. Associated Service information.

7. Associated Device description information.

- A Device is a Physical Entity that could have a sensor, actuator, or tag instantiation.

IoT reference architecture

- Architecture Reference Model (ARM) consists of two main parts:
  1. a Reference model
  2. a Reference Architecture.
- The foundation of an IoT Reference Architecture description is an IoT reference model.
- A System Architecture is a communication tool for different stakeholders of the system.
- Developers, component and system managers, partners, suppliers, and customers have different views of a single system based on their requirements and their specific interactions with the system.
- The high-level abstraction is called Reference Architecture as it serves as a reference for generating concrete architectures and actual systems, as shown in the Figure 7.2.

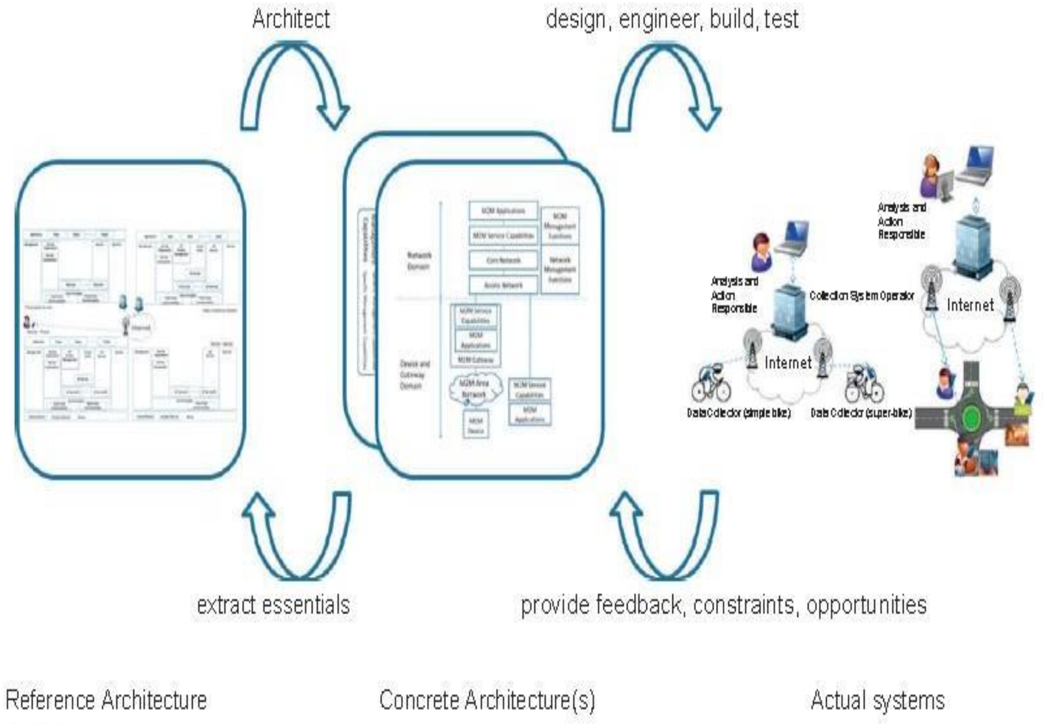

Architect
design, engineer, build, test
Analysis and Action Responsible
Collection System Operator
Internet
Analysis and Action Responsible
Internet
Data Collector (simple bike)
Data Collector (super-bike)
extract essentials
provide feedback, constraints, opportunities
Reference Architecture
Concrete Architecture(s)
Actual systems

FIGURE 7.2

From reference to concrete architectures and actual systems.

- ➢ Concrete architectures are instantiations of rather abstract and high-level Reference Architectures.
- ➢ A Reference Architecture captures the essential parts of an architecture, such as design principles, guidelines, and required parts (such as entities), to monitor and interact with the physical world for the case of an IoT Reference Architecture.
- ➢ A concrete architecture can be further elaborated and mapped into real world components by designing, building, engineering, and testing the different components of the actual system.
- ➢ The general essentials out of multiple concrete architectures can then are aggregated, and contribute to the evolution of the Reference Architecture.

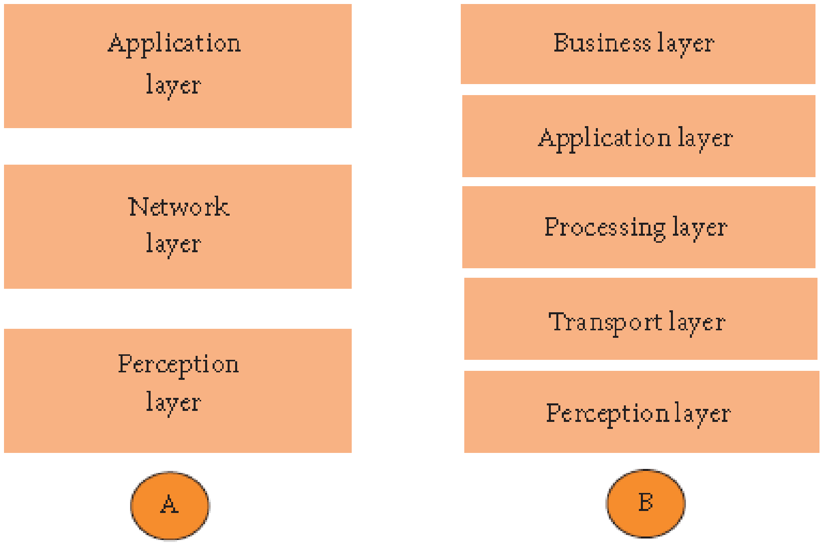


FIGURE 1: Architecture of IoT (A: three layers) (B: five layers).

- It has two types of Architecture:

  Three Layer Architectures
  Five-Layer Architectures

Three Layer Architectures

- It has three layers, namely, the perception, network, and application layers.

  (i) The *perception layer* is the physical layer, which has sensors for sensing and gathering information about the environment. It senses some physical parameters or identifies other smart objects in the environment.
  (ii) The *network layer* is responsible for connecting to other smart things, network devices, and servers. Its features are also used for transmitting and processing sensor data.
  (iii) The *application layer* is responsible for delivering application specific services to the user. It defines various applications in which the Internet of Things can be deployed, for example, smart homes, smart cities, and smart health.

- The three-layer architecture defines the main idea of the Internet of Things, but it is not sufficient for research on IoT because research often focuses on finer aspects of the Internet of Things.

Five Layer Architectures

- The five layers are perception, transport, processing, application, and business layers (see

➢ The role of the perception and application layers is the same as the architecture with three layers. We outline the function of the remaining three layers.

(i) The *transport layer* transfers the sensor data from the perception layer to the processing layer and vice versa through networks such as wireless, 3G, LAN, Bluetooth, RFID, and NFC.

(ii) The *processing layer* is also known as the middleware layer. It stores, analyzes, and processes huge amounts of data that comes from the transport layer. It can manage and provide a diverse set of services to the lower layers. It employs many technologies such as databases, cloud computing, and big data processing modules.

(iii) The *business layer* manages the whole IoT system, including applications, business and profit models, and users' privacy.

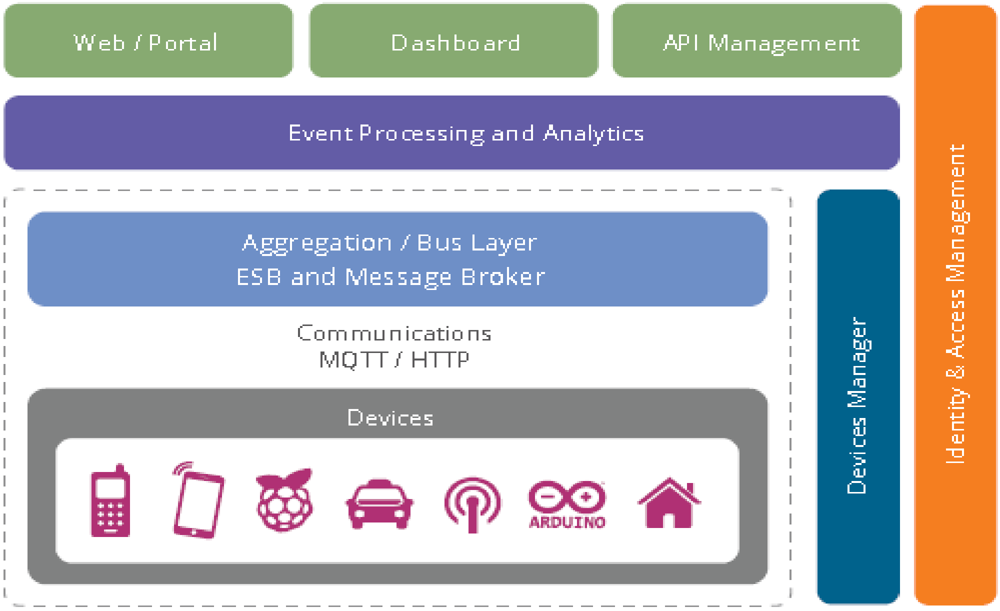


Figure 2. Reference architecture for IoT

➢ 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 management

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 device 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
- Each device typically needs an identity.
- The identity may be one of the following:
  - A unique identifier (UUID) burnt into the device
  - A UUID provided by the radio subsystem (e.g. Bluetooth identifier, Wi-Fi MAC address)
  - An OAuth2 Refresh/Bearer Token
  - An identifier stored in nonvolatile memory such as EEPROM

The 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
  - Constrained application protocol (CoAP)
- HTTP supports many libraries. Because it is a simple textbased protocol, many small devices such as 8-bit controllers can only partially support the .
- The larger 32-bit based devices can utilize full HTTP client libraries that properly implement the whole protocol.
- MQTT solve issues in embedded systems and SCADA.
- MQTT is a publish-subscribe messaging system based on a broker model. The protocol has a very small overhead.
- It is 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. CoAP is a more traditional client-server approach
- CoAP is designed to be used over UDP.

The Aggregation/Bus Layer

- This layer 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 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).
- 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.

The 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.
- It has 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

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).

➢ The API management layer provides three main functions:
• The first is that it provides a developer-focused portal 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.
➢ There are three levels of device: non-managed, semi-managed and fully managed (NM, SM, FM).
➢ 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’s location if available

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 identifying inbound requests from the Web layer

- XACML PDP

- Directory of users (e.g. LDAP)

- Policy management for access control (policy control point)