Chapter 7
IoT Services Platform: Functions and Requirements

IoT is expected to connect billions of sensors, devices, and applications over the Internet. One of the most critical prerequisites for successful, scalable, and effective IoT solutions is a Services Platform that provides abstraction across the multitude of diverse devices and data sources in addition to allowing for the management and control of a range of systems and processes. The operation of this platform requires a comprehensive and diverse set of requisites to gather relevant data, analyze it, and create actionable insights.

The Services Platform must surpass vertical solutions by integrating all essential technologies and required components into a common, open, and multi-application environment. The functions of the IoT Services Platform include the ability to deploy, configure, troubleshoot, secure, manage, and monitor IoT devices. They also include the ability to manage applications in terms of software/firmware installation, patching, starting/stopping, debugging, and monitoring. The Services Platform also provides capabilities that simplify application development through a core set of common application services that include data management, temporary caching, permanent storage, data normalization, policy-based access control and exposure. In addition to these, the Services Platform may offer some advanced application services, which include support for business rules, complex event processing, data analytics, and closed loop control. Figure 7.1 shows examples of key IoT Services Platform Functions. A more detailed and structured list will be provided in Sects. 7.2–7.12.

As can be seen from the list above, many of the capabilities of the IoT Services Platform represent what can be loosely categorized as "management functions." These, however, are different from traditional network management. Traditional network-level management functions were originally defined, in the early 1980s, by the Open Systems Interconnection (OSI) Systems Management Overview (SMO) standard as FCAPS: Fault, Configuration, Accounting, Performance, and Security. A decade later, the Telecommunications Management Network (TMN) of ITU-T, advanced the FCAPS as part of the TMN recommendation on Management

A. Rayes, S. Salam, *Internet of Things from Hype to Reality*,
https://doi.org/10.1007/978-3-030-90158-5_7

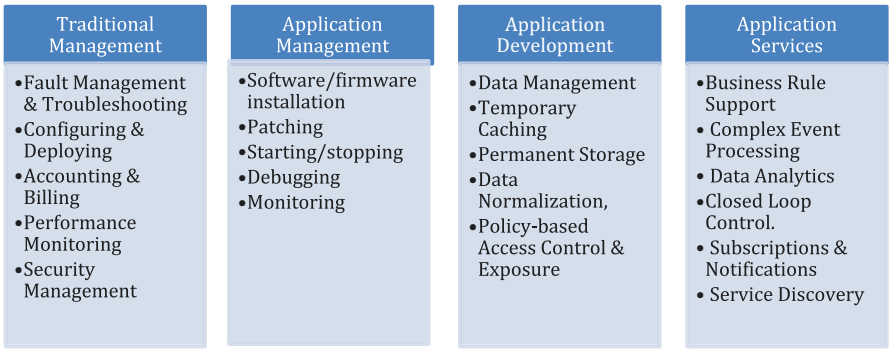

Fig. 7.1 Examples of key IoT Services Platform functions

Functions. The term FCAPS is often used in network management books as a useful way to break down the multipart network management functions.

While FCAPS still apply, the overall management functions of IoT solutions are more multifaceted than traditional networks. This is due to the following factors:

- IoT solutions include new devices (e.g., sensors, white-label gateways, and white-label switches). Some of these devices are inexpensive and generally lack the type or level of instrumentation required for traditional management functions.
- IoT solutions utilize relatively recent technologies (e.g., tracking exact location of IoT device using GPS triangulation) that were not considered by traditional management solutions.
- IoT solutions support more than two dozen access protocols (as was mentioned in Chaps. 4 and 5). The network management for each protocol may vary.
- IoT solutions support multiple verticals, each of which has different sets of management, quality of service, and grade of service requirements.
- IoT solutions utilize a new Fog layer with new and challenging network, compute and storage management requirements.
- Finally, many enterprises and service providers are expected to outsource and, in many cases, multisource key parts of the network and/or management functions. This requires additional, mostly new, capabilities such as secure integration that spans connecting workflows between multiple services providers.

This chapter describes the essential functions of the IoT Services Platform, as shown in Fig. 7.2. It focuses on identifying key capabilities with minimum emphasis on the relationship between the functions or their access protocol interfaces. Such relationship and protocols were addressed in the IoT Protocol Stack Chaps. 4 and 5.

Before introducing the main functions of the IoT Services Platform, we will first revisit the key components of IoT solutions that consist of IoT Device elements, IoT Network elements, IoT Services Platform, and IoT Applications as shown in Fig. 7.3.

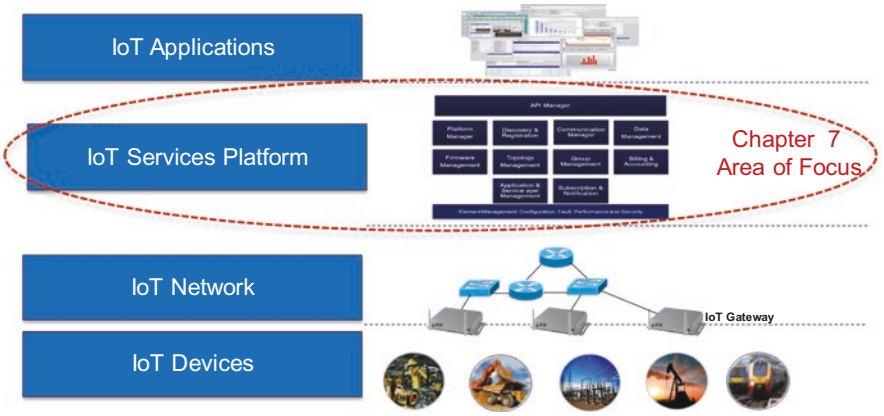

Fig. 7.2 Areas of focus for this chapter

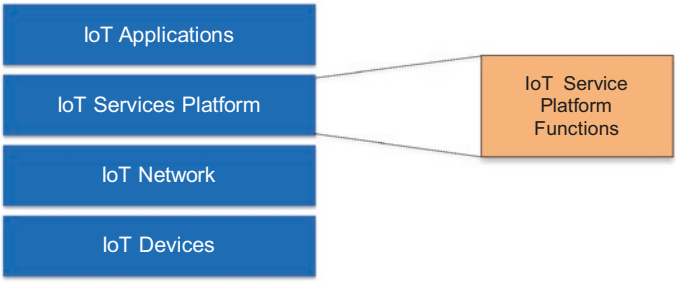

Fig. 7.3 Key components of IoT solution

- **IoT Device Entities**: IoT devices include sensing devices, actuators, and gateways. The main functions of the gateways are (1) collecting and aggregating information from the devices, (2) on-site filtering and simple correlation of collected information, (3) transferring correlated data to the network layer, and (4) taking action on the devices (e.g., shutting power off) based on commands from higher layers.
- **IoT Network Entities**: IoT network entities provide services from the underlying network to the Services platform. They include super-gateways, access routers, switches, and possibly element management servers with specific network management functions.
- **IoT Services Platform Entity**: The IoT Services Platform sometimes referred to as "IoT Platform" or "The IoT Application Services Platform," of any IoT solution. It is responsible for monitoring and controlling IoT elements in the IoT Device and Network Layers. It also allows the creation of direct integration between physical devices (e.g., sensors, actuators, gateways) and computer-based application systems to improve efficiency, accuracy, and economic benefit.

- IoT Services Platform entity receives information from IoT Device and Network Entities, and provides services to the Application Entities. More importantly, it provides network-level and often service-level management functions as will be discussed in this chapter.
- **IoT Application Entities**: Application entities receive information from the Services Platform and provide services and business level functions. These functions are typically vertical dependent. Examples of Application Entities include an IoT-based Automated Parking application, an IoT-based Hurricane Alert System application, etc.

7.1 IoT Services Platform Functions

Without a doubt, the IoT Services Platform constitutes the linchpin of successful IoT solutions. It is responsible for many of the most challenging and complex tasks of the solution. The IoT Services Platforms include numerous fundamental functions to ensure proper and secure deployment and comprehensive supervision and control. In this chapter, we will identify key IoT Services Platform functions by grouping related requirements together and by utilizing recent IoT standards such as those devised by oneM2M[1] and European Telecommunications Standards Institute (ETS) standards bodies. More information on the IoT standards was provided in Chap. 5 (Sect. 5.4.2).

The overall functions of the IoT Services Platform can be categorized into the following 11 key areas:

1. Platform Manager
2. Discovery and Registration Manager
3. Communication Manager
4. Data Management and Repository
5. Firmware Manager
6. Topology Manager
7. Group Manager
8. Billing and Accounting Manager
9. Cloud Service Integration Function/Manager
10. API Manager
11. Element Manager: Configuration Management, Fault Management, Performance Management and Security Management

Figure 7.4 shows the IoT Services Platform functions. It does not constrain the multiplicity of the entities nor the relationships among them.

[1] OneM2M is the global standards initiative for Machine-to-Machine Communications and the Internet of Things.

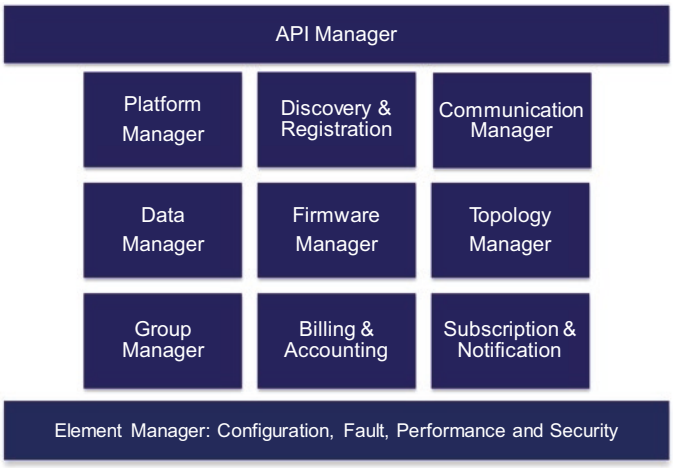

Fig. 7.4 Common IoT Services Platform functions

7.2 IoT Platform Manager

The IoT Services Platform Manager, also known as IoT Service Platform's Management Entity in some standards, is responsible for managing the IoT Service Platform internal modules and interfaces. It works with the Communication Manager (Sect. 7.4) and the Element Manager (Sect. 7.6) to monitor, configure, troubleshoot, and upgrade the Services Platform modules. It is really the "manager of managers" responsible for providing the overall management of the entire Services Platform functions.

The Platform Manager is used for the overall control and management of the common management functions. It allows the system administrator, or an application in the Application Layer, to manage IoT Services Platform components and interfaces. This includes initiating an action (e.g., discovery) and receiving results (e.g., discovered elements) within a specific amount of time.

The Platform Manager is expected to have a full user interface, allowing the system administrator to initiate requests and review reports, and providing interfaces to receive and send information. It must be noted that user and application's authorization (specifying access rights level) and authentication (verifying the user's credentials) is a top requirement.

The Platform Manager may be a physical system/server or virtual system with functions distributed among the common management components.

The IoT Platform Manager is responsible for:

- Performance Monitoring and Fault Management of the Services Platform functions. This includes continuous monitoring, troubleshooting, fault identification, fault correction, and diagnostics. This requires constant collection of logs,

performance and fault parameters from the platform functions (e.g., system logs, alarms).
- Lifecycle software management allowing the IoT Platform Manager to manage any software packages related to the above Services Platform functions. This includes upgrading, updating, installing, uninstalling/removing, and downloading software packages. Complete configuration backups with roll-back capabilities must be supported (Why? See Problem 24).
- Configuring any of the platform functions when they are first installed. This includes the configuration of the services offered to Application Entities.
- Supporting multiple levels of IoT Platform Managers operating in a hierarchical environment. For instance, supporting two Platform Managers, representing two separate networks, and a third "Supper Platform Manager" with full read and write access to the first two. Consequence, Platform Managers should have the ability to establish relationships among each other including establishing parent–child and Read–Write relationships.

The concept of Super Platform Manager is needed to address high availability requirements.

7.3 Discovery: Entities, Services, and Location

Discovery is the process of identifying and transferring information regarding existing IoT entities and/or resources with their locations. Accurate discovery is essential for most IoT management tasks such as asset management, network monitoring, network diagnostics and fault analysis, network planning, capacity expansion, high availability, and others.

One of the key discovery requirements is for IoT entities (e.g., sensors, gateways, routers) to uniquely identify themselves via a common registration process. Hence, each entity needs to be uniquely identifiable through its embedded computing system. It also needs to be able to interoperate within the existing IoT infrastructure via IoT access protocols as we defined in Chap. 5.

An essential requirement for discovery is entity registration. In this section, we will first introduce the registration function and then provide the key requirements for discovery.

7.3.1 Registration

IoT device registration can be defined as the process of delivering the device information to the Management Entity (or to another server) in order for IoT devices to communicate and exchange information. Most IoT devices will be identified and tracked by their IP addresses. However, as we mentioned in Chap. 2, not all IoT

devices are IP-enabled. In such case, devices (e.g., basic sensors) may be tracked by their local (typically non-unique) addresses (e.g., local identifier) in combination with their corresponding gateway IP address. Gateways are expected to have unique IP addresses and are responsible for providing a means to uniquely identify their associated sensors and actuators.

In order for the IoT registration process to work, the following key capabilities are necessary:

- IoT devices must have the capability to register to an associated Platform Manager entity. This procedure may be self-registration (preferred solution) where a new IoT device identifies itself to the management entity as soon as it joins the IoT network or identifies itself during the discovery process as will be discussed in the next section. The registration requirements must be addressed in all IoT domains, i.e.,
  - Ability for new sensors and actuators to register themselves with their associated gateways.
  - Ability for new gateways to register themselves with their associated Platform Manager entities.
  - Ability for Platform Managers to register themselves with a super (or another) Platform Manager(s) as defined by the network administrator.
- Once the registration is complete,
  - The IoT Platform Manager must be able to access the IoT gateway and retrieve information (i.e., Read Access is granted). In other words, IoT gateways must grant full access privilege to the associated IoT Platform Manager(s). Hence, all resource information must be available to the IoT Platform Manager.
  - The IoT gateways must be able to access their associated sensors and actuators and retrieve information. In this case, sensors and actuators resource information must available to the associated IoT gateway(s).
  - Super IoT Platform Manager(s), if present, must be able to access their corresponding IoT Platform Managers and retrieve information. Hence, all resource information must be available to the super management entities where applicable.

7.3.2 Discovery

Based on some filtering criteria (typically specified by a management entity such as the Platform Manager, IoT Gateway, or a northbound application) in the discovery request, the discovery function is responsible for discovering, identifying, and retuning matching information regarding entities and/or resources. The discovery function sends matching information to the requester's system. The discovery request may include the IP or MAC address (obtained from device registration), set of addresses, or range of IP addresses of the resource where the discovery is to be

performed. Full discovery, without any specified addresses, may also be supported. In such case, all entities (based on some filtering criteria in the discovery request) are discovered. Example: Discover all entities in a given enterprise network.

In IoT, the location of the physical entities (e.g., sensors, gateways) is also essential. The discovery function also supports obtaining geographical location information.

It is assumed, therefore, that IoT entities have the capability of identifying, storing, and updating their geographical location information. This may be accomplished with a GPS module in the entity, a location server responsible for tracking and storing location information, or information for inferring location stored in other nodes. The location technology (e.g., Cell-ID, assisted-GPS, and fingerprint) used by the underlying network depends on its capabilities. Sensors with no geolocations are identified by their corresponding gateways.

We will use an example of CoAP (Constrained Application Protocol) to illustrate discovery.

Discovery Request: Assume the IP Address of the Management Server is 192.15.10.5. Also assume the Management Server is interested in discovering sensors within 500 m from the location of (37.76724070774898, −122.37890839576721)[2] GPS Coordinates. The management server will send a CoAP GET request to

```
Coap://192.15.10.5:5784/.well-known/core?
& ro=SSN-XG-IRI&sd=yyyyyy=&at30004&lg=-122.37890839576721
&lt=37.76724070774898&md=500&st=2&sr=70
```

Discovery Reply: Upon receiving the request, the CoAP server will start a matching process comparing the request with all stored information in its local data store. Let us assume that the returned set consists of two sensors matching the request. The CoAP server response payload will be

```
</Hts2030HumidSens>;ct=41;    at30004;    lg=-122.37890839576721;
lt=37.76724070774898&md=310;      ro=SSN-XG-IRI;      sd=aaaaaa;
tittle="Humidity-Sensor-2030",
</BitLineAnemomSens>;ct=0; ct=41;at=30004; lg=-122.37890839576721;
lt=37.76724070774898&md=276;      ro=SSN-XG-IRI;      sd=bbbbbb;
tittle="Anemometer-Sensor-111",
```

Table 7.1 summarizes the Registration and Discovery requirements.

[2](37.76724070774898, −122.37890839576721) are the GPS Coordinate for a northern California area.

Table 7.1 Summary of IoT Registration and Discovery requirements

Function	Responsibility	Results/outputs
Discovery	Identify IoT sensors, actuators, gateways, and devices via attributes and search protocols	IoT entities, gateways, sensors, and actuators based on filtering criteria
	Identify the location of physical entities	GPS location
	Identify access control policies across management servers and clients (see Sect. 7.5)	Access Control Policy information
	Identify IoT services via attributes and collected data	IoT configured services (outside the scope of this book)
Registration	The process of delivering IoT device information (sensors, actuators, gateways, and IoT entities) to the Management Entity, or to another server, in order for IoT devices to communicate and exchange information	Ability for IoT device (sensors, actuators, gateways, and IoT entities) to register with their associated gateways

Finally, IoT software services may also be discovered by collecting configuration and operational parameters (e.g., using YANG,[3] SNMP MIBs, CLI Outputs). IETF defined a set of requirements for standard-based device (configuration and operational data) management. Key functionalities include:

- Ability to collect configuration and operation data from all IoT devices (e.g., running configuration files) where applicable.
- Ability to extract and then structure/model data from configuration and operation files via an information model.
- Ability to distinguish between configuration data and operational data (i.e., data that describes operational state and statistics).
- Ability for operators to configure the entire network and not just individual devices.
- Ability to check configurations consistency between devices in the network.
- Ability to use text processing tools such as diff and version management tools such as CVS.
- Ability to distinguish between the distribution of configurations and the activation of a certain configuration.

Detailed requirements for discovery of software services are outside the scope of this book.

[3] YANG is a tree-structured data modeling language (defined by IETF) used to model configuration and state data [6].

7.4 Communication Manager

The Communication Manager is responsible for providing communications with other platform functions, applications, and devices. This includes supporting the following functionality:

- Ability to provide a global view of the state of the entire underlying platform network. This is needed to address the next requirement.
- Ability to determine the optimal time to establish the communication connection to deliver information between at least two platform entities. Such decision is based on the source delivery request as well as traffic/congestion control optimization techniques within the platform. Data may be stored/buffered for future delivery time per the provisioned Communication Manager policies.
- Ability to deliver required information within the delivery request time.
- Ability to publish its own polices to external systems.
- Ability to provide information to external systems to drive policies describing details of the usage of network resources (i.e., 5% of bandwidth on link X at time T was utilized for service Y).
- Ability to communicate, select paths for a given amount of time, and manage buffers based on communication manager polices.

7.5 Data Management and Repository

Collecting, storing, and exchanging information among various platform entities is one of the key requirements for the IoT Service Platform. Data Storage and Mediation functionalities must include:

- **Data Retrieval**: Data may be retrieved from various sources including IoT devices (e.g., sensors and getaways), IoT network elements (e.g., super-gateways and switches), IoT subscribers or IoT applications. IoT device and network element data is assumed to be collected by collection systems or by collection agents.
- We are using the term "Collection System" to refer to a physical hardware machine (e.g., server, PC) mainly used for data collection. And the term "Collection Agent" refers to a software unit (agent) that resides on a gateway/router blade (or on a computer along with other applications). Hence, Collection System may be the same as Collection Agent (see Problem 30).
- **Data Aggregation**: Data aggregation implies grouping data from similar or diverse sources for further processes. Typically, data from various IoT sources need to be grouped together based on a well-defined data model (e.g., physical locations, device types, subscribers with their assigned devices, etc.). The aggregation syntax should be defined by the data model. Also, data from multiple data collection systems (for the same IoT entity) need to be filtered and aggregated accordingly.

- **Data Parsing**: Data parsing normally implies reading the data, using software, and extracting useful information. Stages of data parsing are hard to define without a concrete use-case but typically include running code to extract specific parameters and writing the extracted data to a database.
- **Data Storing**: The Data Storage and Mediation Function supports taking data from various sources and storing it based on pre-defined policy. Raw data, aggregated data, and parsed data may be stored with different polices (e.g., store raw data for 6 months, store parsed data for 2 years). Associated contextual information is also stored with the data. Examples of contextual information include: data type (e.g., Temperature), data format (e.g., −100 °C to +100 °C) data source (e.g., Sensor ID and Associated Gateway ID), retrieval time and date (e.g., 03:45:00 PM EST on 12/12/2016), retrieval location (e.g., lg = −122.37890839576721; lt = 37.76724070774898).
- **Access to data based on defined access control policy**: The Data Storage and Mediation needs to have the capability of providing local or remote data access based on a well-defined access control policy. The policy, which is typically defined by the network administrator, needs to capture what types of functions a specific user or application can perform on the data (read-only write-only, read/write). The policy may include temporal access restrictions, and may be role based (e.g., administrator vs. user, etc.).

7.6 Element Manager (Managing IoT Devices and Network Elements)

The element management function is expected to manage IoT sensors, actuators, gateways as well as other devices residing within the platform boundaries. The element management function, as shown in Fig. 7.5, typically utilizes the client-server distributed model where a single management server may manage multiple management clients. In this model, tasks are partitioned between the management server (provider of the service) and the management client (service requester). The management client establishes a connection to the management server over the network to accomplish a particular task (e.g., sending performance results of the last 5 min). Once the management client's task is fulfilled, by the management server, the connection is terminated.

In IoT environment, the management server may be residing in a data center while management client may be residing on the IoT Gateway in an offsite location.

A key function of element management includes:

- Ability for the management client and management server to communicate at any time. Hence, real-time communication is required to send time-sensitive data.
- While it is recommended to use a standardized protocol so that any management server can communicate with any management client, any existing client-server

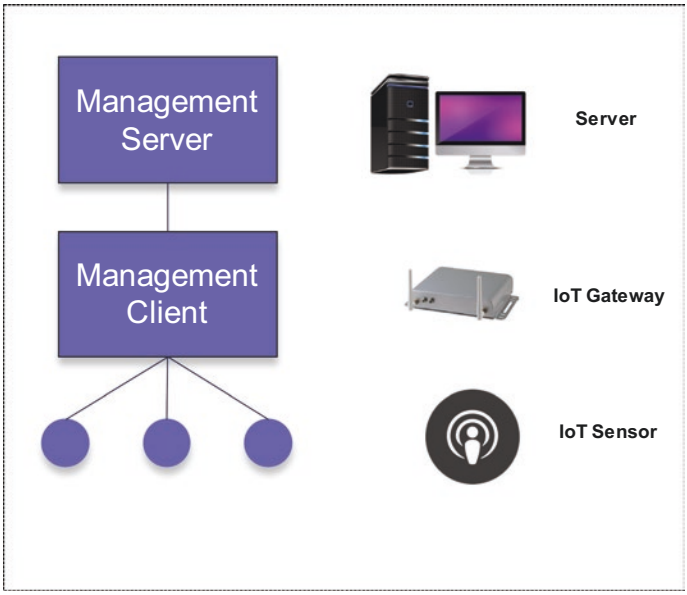

Fig. 7.5 Example of element management function

communication protocol may be utilized. Key examples include TR-069[4] and LWM2M.[5]

- Ability for the management servers (or adaptors) to receive and fully understand (based on an agreed upon protocol) management client requests and/or notifications. For example, air pressure measurements of the oil rig vale.
- Ability for the management clients to receive requests and/or notifications from the management servers (or their adaptors). The management clients may have the ability to fully understand such events and deliver them to targeted sensors, actuators, or device as required. For example, requesting the actuator to shut down a valve.
- Ability for the management server and management clients to address the security requirements as defined later in this chapter and in Chap. 8 including Authorization, Authentication, Access Control, Non-reputation, Data confidentiality, Communication Security, and Data Integrity and Privacy.
- Ability for the super management server to assign different levels of access control privileges when multiple management servers and/or clients exist.

[4] TR-069 as a bidirectional SOAP/HTTP-based protocol that was originally for remote management of end-user devices. It was published by the Broadband Forum and entitled CPE WAN Management Protocol (CWMP).

[5] LWM2M (Lightweight Machine-to-Machine) protocol is defined by the Open Mobile Alliance for M2M/IoT, as an application layer communication protocol between a LWM2M Server and a LWM2M Client (located in a LWM2M Device).

- Ability for the super management server to provide read access (with the appropriate access control requirement) to the discovery or other functions to discover access control policy information.
- Ability for the management server to provide read access (with the appropriate access control requirement) to the discovery or other functions to discover managed elements with their latest collected information (e.g., metadata, values) including gateways, sensors, and actuators.
- Ability for the management server to create a new element to be managed (e.g., gateway, sensor), delete an existing element, update any parameters of any existing elements, update the firmware of any element, and to retrieve information of any existing elements.

7.6.1 Configuration (and Provisioning) Management

Configuration management is one of the most important element and network management functions. Configuration management is the process of enabling (or disabling) a service. Before providing the overall requirements for IoT configuration management, it is worthwhile to discuss the main differences between configuration and provisioning management.

The Provisioning function is concerned with the basic process of preparing and equipping an IoT network to provide proper and effective services, while the Configuration function is concerned with the actual enablement or disablement of an IoT service. Provisioning is often equated to initiation of a service or capability, whereas configuration is the final set of touches to deliver the actual service to a particular customer.

Hence, an IoT network is first generically provisioned (e.g., by installing libraries or services on servers) to provide a set of services to any customers. Such provisioning does not imply that a service can simply be launched without additional instructions on which particular server or set of servers to use, which specific set of already provisioned parameter to employ, how to distribute the load when demand increase, etc.

Figure 7.6 shows an example of Device Remote Management/Configuration to address the machine-to-machine (M2M) environment with OMA (Open Mobile Alliance) lightweight M2M protocol, which focuses on constrained cellular and sensor network M2M devices.

Key configuration requirements include:

- Ability to identify IoT devices and their associated management objects and attributes.
- Ability to enable or disable a device capability.
- Ability to update device parameters.
- Ability to roll-back applied changes in the configuration at least to five back versions (tracked by time and date).

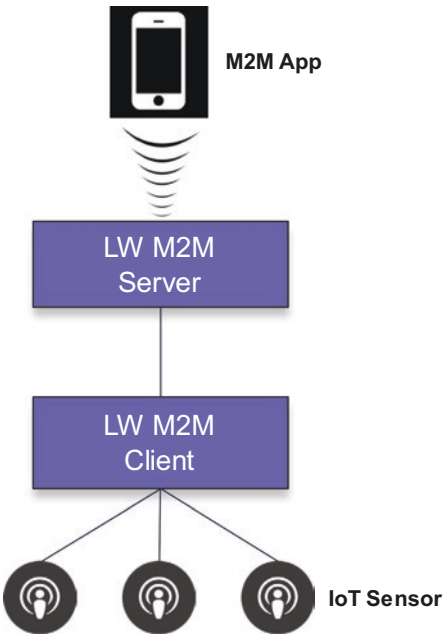

Fig. 7.6 Example of configuration management using LW M2M protocol

- Ability to reset IoT device parameters to original factory values.

On the IoT network side, an example of network element protocol is the Network Configuration Protocol (NETCONF). It provides mechanisms to install and update the configuration of network elements such as a router or switch using XML to encode the configuration data and the protocol messages.

7.6.2 Fault Management

At the minimum, IoT service providers need to be able to configure new service (turn-on a service for a customer) and then identify any problem or potential problem and have the tools to fix it quickly. No service provider will survive in the market if they do not have the capabilities and processes to discover problems promptly (before they occur in most cases) and take quick action to prevent service interruption or service degradation that could result in Service-Level Agreement (SLA) violation.

Fault management is among the most challenging and important management function of IoT networks. This is due to the fact that large-scale deployment of inexpensive sensors (i.e., with very limited processing capability, storage capacity, and limited energy) means that failures from various defects will not be uncommon. It is also due to the fact that managing IoT devices in remote locations and often

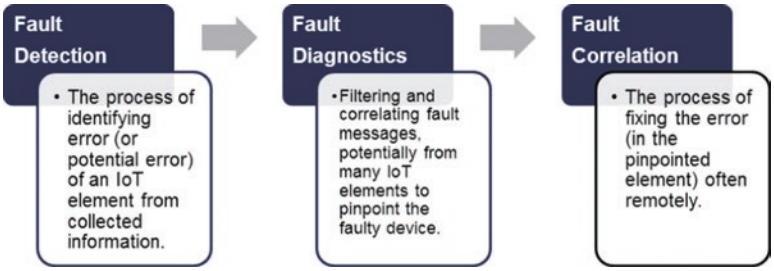

Fig. 7.7 Main stages of fault management function

harsh environments will be demanding, especially when dealing with various IoT topologies and verticals.

Fault Management typically consists of three main functions: fault detection, fault isolation (or diagnostic), and fault correction as shown in Fig. 7.7. In this section, we will first describe these three functions. Then we will introduce fault tolerance and fault or diagnostic signature. Finally we will list the overall fault management requirements for IoT devices and services.

- **Fault Detection** is the process of identifying error (or potential error) of an IoT element typically using collected statistics. The collected data may be time-based (e.g., fault-related data collected from the IoT element by the fault manager function every *t* seconds) or event-based (e.g., IoT element notifies the fault manager only if pre-defined fault-related conditions are met). When a fault or event occurs in the event-based case, an IoT element will send an alarm or notification to the fault manger (and often notify the network administrator) immediately. An alarm is a persistent indication of a fault that clears only when the triggering condition has been resolved.

- An example of fault-related data is the Simple Network Management Protocol (SNMP) Entity Sensor Management Information Base (MIB) as described by IETF RFC 3433. The Entity Sensor MIB provides generalized access to information related to sensors that are often found in network equipment. The complete list of the MIB information is shown in Table 7.2. One of the key variables of the Entity Sensor MIB is "Entity Sensor Status" with three defined possible values:

  – Entity Sensor Status = 1: indicates that the sensor data value can be obtained (normal operation).
  – Entity Sensor Status = 2: indicates that the sensor data value is unavailable (operational but no data was collected).
  – Entity Sensor Status = 3: indicates that the sensor is broken and cannot collect the sensors data value (failure). Once the failure status is received by the network administrator/operator, S/he needs to investigate the issue further to determine if the failure is due to disconnected wire, out-of-range, violently fluctuating readings, or something else.

Table 7.2 Overview of entity sensor MIB

MIB variable	Description	Examples of potential value
EntitySensorDataType	Entity Sensor measurement data type associated with a physical sensor value	3 = Volts AC 4 = Volts DC 5 = Amperes 6 = Watts 7 = Hertz 8 = Celsius
EntitySensorDataScale	A data scaling factor, represented with an International System of Units prefix	6 = Nano 10 = Kilo 11 = Mega 12 = Giga 13 = Tera 14 = Exa
EntitySensorPrecision	Sensors Precision Range	1 = One decimal place in the fractional part 2 = Two decimal place in the fractional part
EntitySensorValue	Sensor Value	From −999,999,999 To +999,999,999
EntitySensorStatus	Operational Status of Physical Sensor	1 = Ok 2 = Unavailable 3 = Nonoperational
TimeStamp	The time the status and/or value of this sensor was last obtained	10:00:00 AM PST

Fault detection will be triggered if the value of "Entity Sensor Status" variable is 3.

- **Fault Diagnostic and Isolation** (also referred to as Fault root cause analysis) is the process of hierarchal filtering and correlating of fault messages, typically from hundreds of IoT elements or systems, to pinpoint the faulty element to a stage where corrective action can be taken. Such process is often based on artificial intelligence, pattern recognition combined with models of abnormal behavior and/or intelligent rule-based systems.
- Pattern recognition with abnormal behavior models is frequently used in the industry to construct the so-called **Diagnostic Signatures** as a form of accumulated and documented knowledge. Fault Diagnostic and Isolation will then take place at run-time based on matching observed information to the nearest Diagnostic Signature.
- Fault managers may use complex filtering systems to assign alarms to severity levels. Alternatively, they could use the ITU X.733 Alarm Reporting Function's perceived severity field: cleared, indeterminate, critical, major, minor, or warning.
- Fault Isolation (or Fault Diagnostic) in IoT-based network is a challenging problem because of the interactions between different network entities (e.g., wireless sensors, gateways) and protocols.
- **Fault correction** is the process of fixing the error/fault problem, often remotely. A fault manager allows a network administrator to monitor events and perform

actions based on received information. Ideally, the fault manger system should be able to not only correctly identify faults but also to automatically take corrective action, such as to activate the notification system to notify a pre-defined list of administrators (i.e., send e-mail or SMS text to a mobile phone) for intervention when needed, or to launch a program or script to take corrective action.

Critical IoT systems should be designed around the concept of fault tolerance. In principle, they must be able to continue working at least to some acceptable level in the presence of faults. Network element redundancy (e.g., multiple sensors performing identical tasks, dual modular sensing engines in the same sensor, fail-over power supply) is a very common fault tolerance example that is designed to prevent failures due to hardware components.

It should be noted that fault tolerance is not just a property of individual IoT elements; it may also impact IoT communication protocols as discussed in Chap. 5. For example, the Transmission Control Protocol (Chap. 2) was designed as a reliable two-way communication protocol, even in the presence of failed or overloaded communications links. It achieves this by requiring the endpoints of the communication to expect errors such as packet loss, packet reordering, packet duplication and corruption.

The element Diagnostics and Fault Management Function in IoT allows network engineers to troubleshoot sensors and actuators (typically over their gateways) or any other IoT entity remotely. Service troubleshooting (i.e., when devices are working correctly but the service-level parameters are not being meet) is also addressed through this function.

The Diagnostics and Fault Management function supports the following areas:

- Ability to connect and uniquely identify any device in the network including sensors, actuators, gateways, etc. Sensors and actuators are often identified by their corresponding gateways.
- Once the connection is established, Fault Management function requires the ability to retrieve device information that identifies a device, its model and manufacturer. E.g., Device Universal ID, Device Product ID, Device Serial Number, SKU.
- Ability to retrieve device information for the software and firmware installed on the device, e.g., embedded software version.
- Ability to retrieve information related to a battery embedded within the device.
- Ability to retrieve information related to memory in use by a device.
- Ability to reconfigure/change (Write option) device specific parameters to diagnose or fix an identified problem.
- Ability to compare results from main system and backup system (if backup system is deployed and operational) and provide error messages for different results.
- Ability to provide the current list of problems occurring on the network to the fault manager/network management systems/system administrator. Such list is cleared only when the triggering condition has been resolved. Or cleared by the network administrator.
- Ability to retrieve the event logs from any IoT device.

- Ability to allow a system administrator to monitor events from multiple systems/locations and perform actions.
- Ability to assign alarms to severity levels. E.g., cleared, indeterminate, critical, major, minor or warning.
- Ability to notify administrators of critical and/or other alarms (based on pre-defined rule-based list) via e-mails, text message, call to mobile phones.
- Ability to launch a program or script to take corrective action for critical and/or other alarm types.
- Ability to reboot diagnostic operation.
- Ability to roll-back any changes at any stage.
- Ability to rest IoT device parameters to original factory values.

7.6.3 Performance Management

The Performance Management function can be defined as a mechanism to quantity "how the underlying IoT infrastructure (e.g., IoT network and device layers) is doing?" Is the infrastructure operating under heavy load (e.g., over 90% utilization) and about to run out of bandwidth or is there substantial extra free capacity so a service provider can offer discounted services?

As was mentioned in Chap. 2, IoT is more than just devices at rest; there are also many mobile IoT devices that include wearables, connected vehicles, and even flying drones. A more formal definition of performance management is a set of processes to measure and monitor the quality and grade of the services that are offered to customers. Quality of Service (QoS) typically refers to performance measures from one element (e.g., delay of one link), whereas Grades of Service (GoS) typically refers to a performance measure of the end-to-end service (e.g., delay of the end-to-end path that a service is taking).[6]

Consequently, a practical description of IoT network performance incorporates three main elements:

- What to measure? Determining what to measure is conceivably the most critical question for IoT management. Smart performance algorithms are useless unless required measurements that drive such algorithms can be collected. In Chap. 3 (Things in IoT), we have identified over a dozen sensor types. Knowing that these sensors are performing correctly is very important. Key sensor performance measures include: Operating range of input-to-output signals, acceptable noise level produced by sensors, acceptable resolution, and acceptable response time to instantaneous change in input signal.
- Generic measurements for all IoT devices (e.g., gateways, routers) will include device and transport link utilization (based on available bandwidth and capacity),

[6] Some researchers use the term QoS to refer to both QoS and GoS as defined above.

end-to-end delay and jitter, packet lost ratios, packet error rates, and any other parameters that impact services carried on the network. These will continue to be important for IoT-based networks.
- Where to measure? Theoretically performance should be measured through the network at all time. Practically, performance should be measured at least between the network end points where the service is delivered. E.g., sensor to gateway, gateway to platform and platform to application.
- How to measure the above parameters and then construct QoS and GoS measures to perform the actual minoring?

Similar to Fault Management, Performance Management supports the following areas for IoT network elements and devices:

- Ability to connect and uniquely identify any device in the network including sensors, actuators, gateways, etc. Sensors and actuators are often identified by their corresponding gateways.
- Once the connection is established, Performance Management function needs to have the ability to ID the device by retrieving device information.
- Ability to retrieve device information for the software and firmware installed on the device, e.g., embedded software version.
- Ability to retrieve information to measure the performance of a device or a module within the device (e.g., battery).
- Ability to measure any performance related parameter including, but not limited to, element utilization, delay, jitter, packet lost, packet arrives with error, amount of memory in use by a device.
- Ability to allow a system administrator to monitor events from multiple systems/locations.
- Ability to notify administrators of critical and/or other performance related activities (based on pre-defined rule-based list) via e-mails, text message, calling mobile phones.

7.6.4 Important Performance Measures for IoT Devices (E.g., Sensors)

The following sensor (and actuators where applicable) performance requirements/characteristics measures are considered important for IoT solutions:

- IoT Sensor's Transfer Function should be plotted (e.g., testing the various ranges of inputs, vendor documentations) to ensure it meets the specific IoT solution requirements. The Transfer Function represents the functional relationship between input signal (physical signal captured by the sensor) and output signal (electrical signal converted by the sensor). Frequently, this relationship is represented by a graph constituting a comprehensive depiction of the sensor characteristics.

- IoT Sensors' Sensitivity should be evaluated and within the minimum acceptable range for the specific IoT solution (e.g., 0.1 variation in temperature sensors may be acceptable for smart homes but not for more critical solutions). The sensitivity is generally the ratio between a small change in electrical output signal to a small change in physical signal. It may be expressed as the derivative of the transfer function with respect to physical signal.
- IoT Sensor's Dynamic Range should be established and documented. Dynamic range is defined as the range of input signals which may be converted to electrical signals by the sensor. Outside of this range, signals cause unsatisfactory accuracy in output.
- IoT Sensor's Accuracy should be established and documented. Accuracy is defined as the maximum expected error between measured (actual) and ideal output signals. Manufacturers often provide the accuracy in the datasheet, e.g., 1% error may be acceptable for some IoT solutions.
- IoT Sensor's Noise Level should be established and documented. As was stated in Chap. 3, all sensors produce some level of noise with their output signals. A sensor's noise is only an issue if it impacts the performance of the IoT system. Smarter sensors must filter out unwanted noise and be programmed to produce alerts on their own when critical limits are reached. Noise is generally distributed across the frequency spectrum. Many common noise sources produce a white noise distribution, which is to say that the spectral noise density is the same at all frequencies.
- IoT Sensor's Resolution should be established and documented. The resolution of a sensor is defined as the smallest detectable signal fluctuation. It is the smallest change in the input that the device can detect. The definition of resolution must include some information about the nature of the measurement being carried out.
- IoT Sensor's Bandwidth (the frequency range) should be established and documented. Some sensors do not operate properly outside their defined bandwidth range.
- IoT Sensor should produce a performance alert and notify its IoT gateway once service issues or interpolation is detected outside its normal operational range (e.g., outside the defined bandwidth, resolution).
- Finally, IoT Sensors should have some ability (depending on the sensors' sophistication level) to work with its IoT gateway to measure the Throughput (actual rate at which the information is transferred), Latency (the delay between the sender and the receiver), Jitter (variation in packet delay at the receiver of the information), and Error Rate (the number of corrupted bits expressed as a percentage or fraction of the total sent) during a specific period of time (e.g., 1 h).

7.6.5 *Security Management*

Security management is extremely important for IoT. Any security management solution must comprehensively address sensitive data handling, data administration, service subscriptions, data transfer (especially over the Internet), data access control, and identity protection. Given the importance of this area, we have dedicated an entire chapter (Chap. 8) to this critical topic. In this section we will simply list the high level security requirements.

IoT high level security requirements include eight main areas:

- ***Data Confidentiality***: ensures that the exchanged messages can be understood only by the intended entities.
- ***Data Integrity***: ensures that the exchanged messages were not altered/tampered by a third party.
- ***Secure Authentication***: ensures that the entities involved in any operation are who they claim to be. A masquerade attack or an impersonation attack usually targets this requirement where an entity claims to be another entity.
- ***Availability***: ensures that the service is not interrupted. Denial of Service attacks target this requirement as they cause service disruption.
- ***Secure Authorization***: ensures that entities have the required control permissions to perform the operation they request to perform.
- ***Freshness***: ensures that the data is fresh. Replay attacks target this requirement where an old message is replayed in order to return an entity into an old state.
- ***Non Repudiation***: ensures that an entity cannot deny an action that it has performed.
- ***Forward and Backward Secrecy***: Forward secrecy ensures that when an entity leaves the network, it will not understand the communications that are exchanged after its departure. Backward secrecy ensures that any new entity that joins the network will not be able to understand the communications that were exchanged prior to joining the network.

Detailed discussions of the above areas including existing solutions and gaps will be provided in Chap. 8.

7.7 Firmware Manager

In the past, Firmware Management was not even an issue as older devices rarely required operating system updates. In fact, Firmware is not part of the traditional FCAPS capabilities that we described in Sect. 7.1.

Firmware refers to the device's operating system that controls and operates the device. Firmware is a program written into read-only-memory (ROM), rather than simply being loaded into normal device storage, where it may be easily erased in the event of a crash, and initially added at the time of manufacturing. It is called

firmware rather than software to highlight that it is very closely tied to the particular hardware components of a device.

Nowadays, firmware updates are provided by vendors on regular basis, often as a way to fix bugs or introduce new functionality (e.g., Apple's iOS, Cisco's IOS, Samsung's Android).

Key Firmware requirements for IoT solutions include:

- Ability for IoT device to store and maintain multiple firmware images and to manage individual firmware images.
- Ability for IoT management solution to provide a user-friendly device Firmware Management site that provides lifecycle management for firmware associated with a device. This includes
  - Downloadable versions of latest Firmware images.
  - Step by step instructions to download/update images on various supported devices that guarantee full migration of existing settings and applications on an IoT Device.
  - Step by step instructions to remove a Firmware image and roll-back into an older image if needed with full device backup of existing applications and settings.
  - Support for downloading and updating within the same action.
  - Download, update, and removal of Firmware process should be done within a reasonable amount of time (typically less than 10 min) with clear progress bar visible to the user.
  - Q&A and troubleshooting support.
- Ability for IoT management solution to support both wire-line and mobile (the so-called FOTA (Firmware Over-The-Air) firmware upgrade. FTOA is a Mobile Software Management (MSM) technology in which the operating firmware of a mobile device is wirelessly upgraded and updated by its manufacturer. FOTA-capable devices download upgrades directly from the service provider.

7.8 Topology Manager

IoT network topology refers to the arrangement of the various elements (sensors, gateways, switches, links between gateways and switches, etc.). Topology may be physical or logical and is often presented explicitly in a structured graph. Physical Topology is the placement of the actual IoT elements on a graph (e.g., map) as they are connected with physical information (e.g., locations). Logical Topology, on the other hand, displays virtual information such as network virtualization data, data flow on the network.

Key requirements for topology management include:

- Ability to display IoT Physical network that includes all IoT devices (e.g., sensors, actuators) and IoT network elements (gateways, switches, routers). User should have the ability to filter which devices to display.
- Ability to display IoT Virtual network (often on top of a physical view).
- Ability to display specific Element Management parameters (e.g., utilization, devices at faults) based on user selection criteria.
- Ability to filter/configure the topology.
- Ability to retrieve information related to any IoT element.
- Ability to retrieve information related to an IoT protocol.

7.9 Group Manager

Unlike traditional networks, a typical IoT network often contains a large number of IoT devices (e.g., sensors). Hence, it is important to allow network administrators to group IoT elements of the same characteristics into groups instead of managing each element separately.

Group Management is responsible for handling group related requests. The request is sent to manage a group and its membership as well as for any bulk operations, including broadcasting/multicasting, that are supported by the group. Group management security is handled by the element management system.

When facilitating access control using a group, only members with the same access control policy for a resource are included in the same group. Also, only application entities, which have a common role with regard to access control policy, are included in the same group. This is used as a representation of the role when facilitating role based access control.

Group Management Key requirements include:

- Ability to create, retrieve, update, or delete groups. Groups are created by selecting IoT elements of similar characteristics. An IoT element may belong to multiple groups. New members may be added and/or deleted at any time. When new members are added to a group, the group manager should validate if the member complies with the purpose of the group. Requests to create, retrieve, update, or delete are assumed to be initiated by an application.
- Ability to create super group (group of a group). In this case, operations (e.g., Forwarding) are done recursively.
- Ability to initiate and execute a request for the entire members of a group. The request may be a simple notification or read operation (i.e., retrieve information form sensors), or write operation (changing a common parameter).
- Ability to support subscriptions to individual groups.
- Ability to notify group members when they are added to or deleted from a group, or when the group is updated.

7.10 Billing and Accounting

Billing and Accounting management is used to calculate and report the charges based on subscription and/or usage of a service. It supports different charging models including online real-time credit control by interacting with the charging system in the underlying IoT network. Billing polices include the ability to trigger a charge based on specified events and to charge even when the billing system is offline. The system may record information for other purposes such as for event logging. The main charging models include:

- Subscription-based charging (flat rate): Typically a service layer per subscription.
- Event-based charging (per event or task): Charging based on service layer chargeable events. For example, an operation on data (Create, Update, and Retrieve) can be an event.
- Time-based charging: Chargeable events are configurable to initiate information recording. More than one chargeable event can be simultaneously configured and triggered for information recording.
- Usage-based charging: Charge based on bandwidth (or other parameters) consumptions. Users are allowed to change usage level within a task (e.g., high bandwidth for first hour and then switch to lower bandwidth).

Key Billing and Accounting requirements include:

- Ability to bill based on subscription (flat rate), event (per event), time (charge per hour), or usage.
- Ability to allow an application (or network administrator) to develop billing related policies. Further, the Billing and Accounting Module has the ability to start and end the actual billing by applying charging related policies, configurations, and communicating with the charging system in the underlying network.
- Ability to start and end charges based on the defined charges policies. Such charges must be recorded in a billing system/DB.
- Ability to handle offline billing related operations. The offline billing function generates service charging records based on billing polices and recorded information. A service charging record is a formatted collection of information about a chargeable event (e.g., amount of data transferred) for use in billing and accounting.

7.11 Subscription and Notification Manager

Subscription and Notification service provides notifications concerning subscription events. It allows authorized devices and applications to subscribe to a set of notification services, typically from a predetermined list. A notification event may be generic (e.g., a recent security alert) or subscriber-specific (e.g., security alert

related to an IoT service and/or device such as end of life date). Subscription and Notification service also provides notifications concerning subscriptions that track event changes on a resource (e.g., deletion of a resource, important change in the resource's events such as a major increase in the temperature reading). The subscription may be provided by the platform itself or by a northbound application communicating with the platform via the API Manager, as shown in Fig. 7.4.

Key requirements for the Subscriptions and Notification Modules include:

- Ability to allow devices and/or applications to subscribe to specific set of services based on right level of authorization. Hence, authorization information may be obtained from the authorization service as we mentioned in Sect. 7.6.5 under Element Management system.[7]
- Ability to allow authorized devices and/or applications to subscribe to a set of notification services from a drop down list.
- Ability to support generic notifications as well as subscriber-specific notifications where notifications are correlated with the subscriber's IoT device or service as mentioned above.
- Ability to support subscription and notification services related to event changes on a resource as mentioned above.
- Ability to provide subscription and notification service in the platform itself and/or in a northbound application. In the latter case, subscription selection is made in an application that communicates with the platform via the API Manager. Notification may also be sent to such application (if so is selected) via the API Manager.
- Ability to notify devices and/or applications based on subscription and authorization level (e.g., subscribe and notify only for security-related alerts).
- Ability to create and store subscription profile information including device ID, notification address, notification type, notification policies (e.g., notify any time for priority 1 issues, notify from 8 AM to 5 PM for priority 2, etc.).
- Ability to subscribe to a single or multiple resources.
- Ability to store subscription profiles as well as directed notifications along with date, time, and delivery mechanism.

7.12 API Manager

The main function of the API Manager is to manage communication with IoT network and devices, for obtaining network service functions in a common way. It is intended to shield other platform modules from developing their own technology and mechanisms supported by the Underlying Networks.

Key functions of the API Manager include:

[7] Alternatively, an Authorization, Authentication and Accounting (AAA) server may be used for device authorization.

- Ability to provide adaptation for different sets of network service functions supported by various Underlying Networks.
- Ability to maintain the necessary connections between the platform entities and the Underlying Network.
- Ability for the API Manager to provide information to the Communication Manager related to the IoT Network so the Communication Manager can include that information determine proper communication handling.

7.13 Commercially Available IoT Platforms

Tens of IoT Platforms exist in the marketplace today. Examples include AWS IoT Platform, Google Cloud IoT Platform, Microsoft Azure IoT Suite Platform, IBM Watson IoT Platform, Salesforce IoT Cloud Platform, Cisco IoT Cloud Connect Platform, Oracle IoT Intelligent Applications Platform, PTC ThingWorx IoT Platform, OpenRemote IoT Platform (open-source focusing on helping engineers creating a range of IoT applications), IRI IoT Voracity Platform (focusing on data discovery, integration, migration, governance, and analytics), Particle Platform, and Altair IoT SmartWorks Platform.

As we mentioned earlier in this chapter, IoT platforms are used to address one or more of the following functions.

1. Rapid and consistent development and deployment of IoT devices and services.
2. Middleware connecting IoT devices and applications to other devices and applications.
3. Streaming data from IoT devices.
4. Profiling customer context data.
5. Device management addressing the FCAPS (Fault, configuration, Accounting, Performance, and Security) functions. See Sect. 7.6 for additional information.
6. Real-time reporting and advanced analytics, e.g., using artificial intelligence algorithms for advanced prediction, service optimization, diagnostics, and trending analysis.
7. Sandbox allowing subject matter experts to test business or technical ideas without (or with limited) programming.
8. Provide API library allowing engineers to import data from other sources (e.g., gateways, Websites, controllers, end application service) and platforms (e.g., using RESTful API).
9. Handle huge data volume from devices, users, applications, websites, and sensors and take actions to give a real-time response.

Selecting the right IoT Platform is challenging and depends greatly on the requirements of the specific solution for hardware, real-time access, custom reports, budget, development skills, and business model.

The purpose of this section is to introduce students and engineers into examples of known IoT platforms and related functionalities. It is not intended to provide

recommendations, nor provide feature by feature comparisons. The selected platforms, as shown in Table 7.3, include AWS IoT Platform, Google Cloud IoT Platform, Microsoft Azure IoT Platform, and PTC ThingWorx IoT Platform. The first three are typically considered general-purpose platforms addressing various IoT applications while the last platform (i.e., PTC ThingWorx) is more focused on addressing industrial IoT requirements.

Again, it is important to note that the feature description (Table 7.3) is snapshots at the time of the writing. Such features and capabilities are expected to change over time. Students/engineers are encouraged to log into each platform to understand the latest capabilities.

7.14 Putting All Together

As we mentioned in pervious section, IoT platforms can be divided into two categories: product-centered with a stronger focus on specific products for industrial companies, and general-purpose platform for developers. In many cases, general-purpose platforms are complemented by an accompanying marketplace.

Marketplace is an e-commerce platform owned and operated by a specific vendor (e.g., Amazon). It enables third-party sellers to offer products and/or services online alongside the vendor's regular offerings. This allows the vendor (platform owner) to earn commissions and to create more comprehensive solutions.

Marketplaces have several advantages. First, they bring together offers from multiple suppliers or service providers with minimum investments. Second, they relieve marketplace owners from owning the inventory that their platform sells. Third, they allow platform owners to choose a revenue stream that best fits their market position and business goals. Finally, marketplace owner leaves the more operational side of the business to vendors while focusing on promoting their marketplace brand. Marketplace owners can create a rating and review systems allowing their customers to make informed purchase decisions.

Let us imagine that you are developing an IoT security solution for your own home. In this case, you will need to install and connect your home cameras and sensors to the Internet, select data sources and protocols, and then develop an application for data visualization. You also need to make sure that your data is secure at all time (e.g., data is not altered by third party, your devices are never hacked, and your credentials are always secure) and that your network is reliable and available. You may also chose to combine your data with additional available information (e.g., Weather conditions, Fire and Crime alerts along with Locations) for advanced monitoring especially when you are traveling.

In this case, you will need to subscribe to a platform allowing you to connect your devices, collect data in real-time, and then build (or utilize and existing) interactive dashboards to visualize and track your home data. Such capabilities may be offered by the platform or the associated marketplaces.

Table 7.3 Examples and glimpses of commercially available platforms

	AWS IoT Platform	Google Cloud IoT Platform	Microsoft Azure IoT Platform	PTC ThingWorx IoT Platform
Overview	Almost all IoT platforms allow users to connect their IoT devices and data sources, select supported protocols, build applications, enable security, and define the communication between devices and the Internet.			
Protocols Snapshot	Supports wide variety of communication protocols including custom ones which enable communication b/w devices from different manufacturers, e.g., MQTT. HTTP and WebSockets for asynchronous communication.	Supports wide variety of communication protocols to enable communication between devices from different manufacturers including, e.g., MQTT, HTTP.	Supports wide variety of communication protocols to enable communication between devices from different manufacturers, e.g., AMQP, HTTPS and AMQP. IoT hub Supports SASL and AMQP claim based security in conjunction with AMQP protocol.	Supports wide variety of communication protocols to enable communication between devices from different manufacturers, e.g., MQTT, HTTP, OAuth2, and WebSockets.
Element Management: Fault Management Snapshot	**AWS IoT Device Manager** allows users to troubleshoot device functionality and query the state of IoT devices.	**Google Cloud IoT Core** supports trouble management, e.g., predicting when equipment needs maintenance.	**Microsoft Azure IoT Monitor** provides guidance to reduce the time in diagnosing and troubleshooting.	Supports various functions for troubleshooting including connections to the platform.
Element Management: Configuration Management Snapshot	Users can query the state of device(s) on demand and provide the functionality to apply firmware updated over-the-air.	The device manager allows devices to be configured (in group) through a console or programmatically.	Azure IoT Hub Device Provisioning Service enables zero-touch provisioning to the right IoT Hub.	Includes utilities to provision devices. Allows users to create rule-based Workflows to execute across multiple devices.
Element Management: Accounting and Billing Snapshot	Basic connectivity fee (for platform access) and then usage-based billing (bay for what you use).	Usage-based: Cloud IoT Core is priced according to the data volume.	Basic and standard tier-based billing model. e.g., $0.123 per 1000 operation for device provisioning.	Subscription based with a pay-as-you-go model is supported.

(continued)

Table 7.3 (continued)

	AWS IoT Platform	Google Cloud IoT Platform	Microsoft Azure IoT Platform	PTC ThingWorx IoT Platform
Element Management: Performance Management Snapshot	**AWS IoT Device Manager** monitors, organizes, and provides an interface to manage IoT devices. It provides functionality to register an individual device or in bulk and manage security permissions/ policies.	**Google Cloud IoT Core** provides a solution for collecting, processing, analyzing, and visualizing IoT data in real time. E.g., automatically optimize device performance in real time while predicting downtime.	**Azure Monitor and Resource Health** provides monitoring capabilities with data about the operations of Azure IoT Hub, for instance. Advanced analytics features that can turn connectivity and workflow data into actionable insights.	**ThingWorx Platform** allows user to select data and use it to create specific charts and workflow alerts. Advanced analytics features that can turn connectivity and workflow data into actionable insights.
Element Management: Security Management Snapshot	Data to and from AWS IoT is sent securely over Transport Layer Security (TLS). AWS cloud security mechanisms protect data as it moves between AWS IoT and other AWS services.	Allows users to securely connect, manage, and ingest data using TLS.	Uses TLS based handshake and encryption. Support various security functions including security information and event management, security orchestration, and automation.	Provides transport security, identity management (device and platform), and content & asset management.
	Supports device authentication and authorization (via custom schemes).	Supports device authentication and authorization (via keys and JSON web tokens).	Supports device authentication and authorization (via certificates and keys).	Supports device authentication and authorization.
	Supports various compliance management for security audits.	Supports various compliance management for security audits.	Supports various compliance management for security audits.	Supports various compliance management for security audits.

In general, the following steps are followed:

1. Install your devices.
2. Connect devices to the platform.
3. Select data sources and formats.
4. Add a custom data source via Developer Console (if applicable).

5. Use standard dashboard capabilities (or add a custom dashboard via a developer console) to create your view. Connect dashboard to data source using an offer interface.
6. Continue adding your devices and data sources to enable complete visualization.
7. Customize your dashboard if needed (e.g., Drag and drop widgets into the desired dashboard location, add custom colors).
8. Share dashboard with family members, e.g., adding users with read-only or edit access.
9. Add additional advanced capabilities (if needed).

7.15 Summary

Without a doubt, the IoT Services Platform creates the cornerstone of successful IoT solutions. It is responsible for many of the most challenging and complex tasks of the solution. The Services Platform automates the ability to deploy, configure, troubleshoot, secure, manage, and monitor IoT entities ranging from sensors to applications in terms of firmware installation, patching, debugging, and monitoring just to name a few. The Service Platform also provides the ability for data management and analytics, temporary caching, permanent storage, data normalization, policy-based access control and exposure.

Given the complexity of the services platform in IoT, this chapter grouped the core capabilities into 11 main areas: Platform Manager, Discovery and Registration Manager, Communication (Delivery Handling) Manager, Data Management and Repository, Firmware Manager, Topology Management, Group Management, Billing and Accounting Manager, Cloud Service Integration Function/Manager, API Manager, and Element Manager addressing Configuration Management, Fault Management, Performance Management and Security Management across all IoT entities.

Problems and Exercises

1. This chapter categorized the IoT Services Platform into 11 functions. (a) Name and define each of the 11 functions. (b) List and define the Element Manager functions.
2. What are the traditional FCAPS management functions? Do they also apply to IoT? If so, Are they sufficient?
3. List six reasons why the overall management functions of IoT solutions are more multifaceted than traditional networks.
4. IoT solutions are considered much more complex to manage than traditional networks?

   (a) Why?—List top five factors.
   (b) Why does the Fog Layer introduce new changes for IoT?

5. This chapter mentioned that not all IoT entities will be IP address enabled.
   (a) Why is that? Provide an example of IoT devices that are not IP addresses enabled.
   (b) How do management system track such devices?
6. What is device registration on IoT? Why is it needed?
7. List the key responsibilities of the Discovery Function.
8. It was mentioned in Sect. 5.1 that for non-IP addressed enabled sensors, IoT sensors may be tracked by the combined (a) IP Address of the Gateway and (b) Sensor address. Why both addresses do are needed?
9. Why IoT device self-registration is preferred over the method where a new IoT device have the capability to be identified during the discovery process?
10. The IETF has released NETCONF and YANG which are standards focusing on Configuration management. Name two other older methods that can be used for configuration management? What are their shortcomings?
11. Section 7.7 indicated that Accurate discovery is essential for many management tasks including asset management, network monitoring, network diagnosis and fault analysis, network planning, high availability, and others.
    (a) Provide short definitions of asset management, network monitoring, network diagnosis and fault analysis, network planning and high availability.
    (b) Why is accurate discovery essential for each of the above functions?
12. What are the key differences between Provisioning and Configuration functions? Which one is done first?
13. What are key differences between deployment, Provisioning, and Orchestration?
14. What are the most basic two management functions to provide a new services?
15. Provide an example of Service-Level Diagnostics and Fault Management Function in IoT where all devices are working correctly but the service-level parameters are not being met.
16. Why Fault management is considered by many experts to be most challenging and important management function of IoT-based networks?
17. What are the three main functions of Fault Management? Provide detailed description of each term.
18. What are the concepts of fault tolerance in IoT networks? Give three examples of failures that should be handled by fault tolerance function in IoT-based networks.
19. Fault tolerance is not just a property of individual IoT element; it may also impact the IoT communication protocol. For example, the Transmission Control Protocol (TCP) was design as reliable two-way communication protocol, even in the presence of failed or overloaded communications links. How is this achieved in TCP?
20. There are special software and instrumentation packages designed to detect failures. A good example is a fault masking system. How does Fault Masking system detect failure?
21. What is Diagnostic Signature? Where it used?

22. In priority order, what are the top three IoT management functions that a service provider needs to provide very basic services? Justify your answer.
23. Why Fault management is considered to be very challenging in IoT network? i.e., What are the main differences between managing IoT network and a traditional network?
24. Why IoT management is considered to be most challenging and complex task of the solution?
25. Section 7.1 indicated the need for a complete configuration backups with roll-back capabilities as a key requirement for the IoT Platform Manager. What is configuration roll-back? Why is it needed? Provide an example?
26. What are the definitions of Sensitivity and Dynamic Range? What are the typical units of Sensitivity and Dynamic Range?
27. What is Hysteresis? What is a typical unit of Hysteresis?
28. What is a Firmware? What does it do? Why is it called so?
29. Why Firmware Images are loaded into ROM and not the device storage?
30. How come Firmware Management was not part of the tradition FCAPS?
31. Data may be retrieved from various IoT sources including IoT devices and network elements (e.g., sensors, gateways, switches), IoT subscribers, and IoT applications. IoT device and network element data is assumed to be collected by collection systems or by collection agents.
    (a) What are the key differences between a collection system and a collection agent?
    (b) What is IoT subscriber data? How is the data collected?
    (c) What is an IoT application data? How is the application data collected?
32. In a table list three Subscription and Notification requirements along with examples of a subscriber and notification message.

References

1. IoT – Converging Technologies for Smart Environments and Integrated Ecosystems, Reviewer Publishers, Online: http://www.internet-of-things-research.eu/pdf/Converging_Technologies_for_Smart_Environments_and_Integrated_Ecosystems_IERC_Book_Open_Access_2013.pdf
2. Internet of Things, Evolving the Manufacturing Industry, Online: http://www.cisco.com/c/en/us/solutions/internet-of-things/iot-products/services.html
3. The Internet of Things: Between the Revolution of the Internet and the Metamorphosis of Objects, Gerald Santucci, Online: http://cordis.europa.eu/fp7/ict/enet/documents/publications/iot-between-the-internet-revolution.pdf
4. From the Internet of Computers to the Internet of Things, Friedemann Mattern and Christian Floerkemeier, Distributed Systems Group, Institute for Pervasive Computing, Online: http://www.vs.inf.ethz.ch/publ/papers/Internet-of-things.pdf
5. Reaping the Benefits of the Internet of Things, Cognizant Reports, May 2014, http://www.cognizant.com/InsightsWhitepapers/Reaping-the-Benefits-of-the-Internet-of-Things.pdf
6. Philip N. Howard (8 June 2015). "How big is the Internet of Things and how big will it get?". *The Brookings Institution*. Retrieved 26 June 2015.

7. Stefan Wallina and Claes Wiksrom, “Automating Network and Service Configuration Using NETCONF and YANG”: Online: http://www.tail-f.com/wordpress/wp-content/uploads/2013/03/Tail-f-NETCONF-YANG-Service-Automation-LISA-Usenix-2011.pdf
8. BJORKLUND, M. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, Oct. 2010.
9. Broadband Forum Technical Report TR-069, CPE WAN Management Protocol, Issue 1 Amendment 5, Version 1.4, Nov 2013.
10. Open Mobile Alliance M2M Device Management Specifications, Online: http://openmobileal-liance.hs-sites.com/lightweight-m2m-specification-from-oma
11. Open Mobile Alliance LightweightM2M Version 1.0, Online: http://technical.open-mobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0
12. Sokullu, R. and Karaca, O., “Fault Management for Smart Wireless Sensor Networks,” Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), Sept 4, 2012.
13. G. Stanley and Associate, White Paper, “A Guide to Fault Detection and Diagnosis”, Online: http://gregstanleyandassociates.com/whitepapers/FaultDiagnosis/faultdiagnosis.htm
14. IETF Network Working Group, Request for Comments (RFC) 3433, Entity Sensor Management Information Base, Online: https://tools.ietf.org/html/rfc3433
15. G. Huston, “Measuring IP Network Performance”, the Internet Protocol Journal, Vol 6, Number 1, Online: http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_6-1/measuring_ip.html
16. H Hui-Ping, X Shi-De, M Xiang-Yin, “Applying SNMP Technology to Manager Sensors in IoT”, The Open cybernetics & Systemic Journal, 2015, pp. 1019-1024, Online: http://ben-thamopen.com/contents/pdf/TOCSJ/TOCSJ-9-1019.pdf
17. L. Adaro, Monitoring 101 eBook, Nov 2015, Online: https://thwack.solarwinds.com/docs/DOC-187523
18. Stanford Sensor Course, Online: http://web.stanford.edu/class/me220/data/lectures/lect02/lect_2.html
19. B. Hedstrom, A. Watwe, S. Sakthidharan “Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions”, University of Colorado, May 2011, Online: http://morse.colorado.edu/~tlen5710/11s/11NETCONFvsSNMP.pdf
20. OMA LightweightM2M v.10, Open Mobile Alliance, Online: http://technical.open-mobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0
21. S. Duquet, Smart Sensors, Enabling Detection and Ranging for IoT and Beyond, Ladder Technology Magazine Elektronik Praxis, April 2015, Online: http://leddartech.com/smart-sensors
22. 50 Sensors Applications for Smarter World, Libelium, Online: http://www.libelium.com/top_50_iot_sensor_applications_ranking/
23. P. Seneviratne, Internet Connected Smart Water Sensors, September 2015, Online: https://www.packtpub.com/books/content/internet-connected-smart-water-meter
24. P. Jain, Pressure Sensors, Prototype PCB from $10, Online: http://www.engineersgarage.com/articles/t
25. D. Merrill, J. Kalanithi, P. Maes, “Siftables: Towards Sensor Network User Interfaces”, Online: http://alumni.media.mit.edu/~dmerrill/publications/dmerrill_siftables.pdf
26. Whatis.com, Online: http://whatis.techtarget.com/definition/firmware
27. Mobileburn, Online: http://www.mobileburn.com/definition.jsp?term=firmware