5.1 Introduction

IoT systems comprise of multiple components and deployment tiers. In Chapter-1, we defined six IoT system levels. Each level is suited for different applications and has different component and deployment configurations. Designing IoT systems can be a complex and challenging task as these systems involve interactions between various components such as IoT devices and network resources, web services, analytics components, application and database servers. Due to a wide range of choices available for each of these components, IoT system designers may find it difficult to evaluate the available alternatives. IoT system designers often tend to design IoT systems keeping specific products/services in mind. Therefore, these designs are tied to specific product/service choices made. This leads to product, service or vendor lock-in, which while satisfactory to the dominant vendor, is unacceptable to the customer. For such systems, updating the system design to add new features or replacing a particular product/service choice for a component becomes very complex, and in many cases may require complete re-design of the system.

In this Chapter, we propose a generic design methodology for IoT system design which is independent of specific product, service or programming language. IoT systems designed with the proposed methodology have reduced design, testing and maintenance time, better interoperability and reduced complexity. With the proposed methodology, IoT system designers can compare various alternatives for the IoT system components. The methodology described in this Chapter is generally based on the IoT-A reference model [75], but is broad enough to embrace other industry efforts as well. Later chapters in this book describe the implementation aspects of various steps in the proposed methodology.

5.2 IoT Design Methodology

Figure 5.1 shows the steps involved in the IoT system design methodology. Each of these steps is explained in the sections that follow. To explain these steps, we use the example of a smart IoT-based home automation system.

5.2.1 Step 1: Purpose & Requirements Specification

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

Applying this to our example of a smart home automation system, the purpose and requirements for the system may be described as follows:

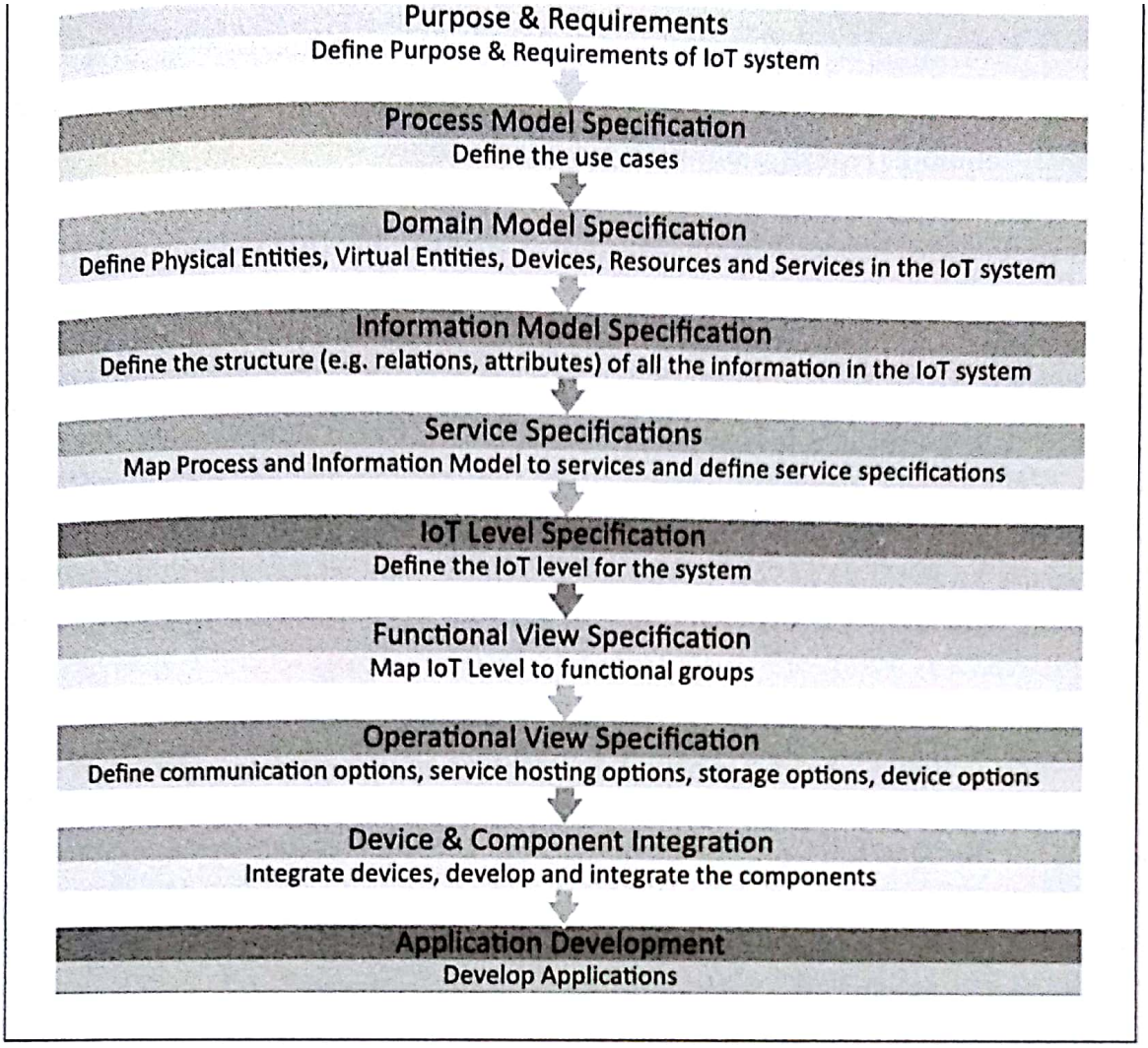

Figure 5.1: Steps involved in IoT system design methodology

- **Purpose :** A home automation system that allows controlling of the lights in a home remotely using a web application.
- **Behavior :** The home automation system should have auto and manual modes. In auto mode, the system measures the light level in the room and switches on the light when it gets dark. In manual mode, the system provides the option of manually and remotely switching on/off the light.
- **System Management Requirement :** The system should provide remote monitoring and control functions.
- **Data Analysis Requirement :** The system should perform local analysis of the data.
- **Application Deployment Requirement :** The application should be deployed locally on the device, but should be accessible remotely.

- **Security Requirement :** The system should have basic user authentication capability.

5.2.2 Step 2: Process Specification

The second step in the IoT design methodology is to define the process specification. In this step, the use cases of the IoT system are formally described based on and derived from the purpose and requirement specifications. Figure 5.2 shows the process diagram for the home automation system. The process diagram shows the two modes of the system - auto and manual. In a process diagram, the circle denotes the start of a process, diamond denotes a decision box and rectangle denotes a state or attribute. When the auto mode is chosen, the system monitors the light level. If the light level is low, the system changes the state of the light to "on". Whereas, if the light level is high, the system changes the state of the light to "off". When the manual mode is chosen, the system checks the light state set by the user. If the light state set by the user is "on", the system changes the state of light to "on". Whereas, if the light state set by the user is "off", the system changes the state of light to "off".

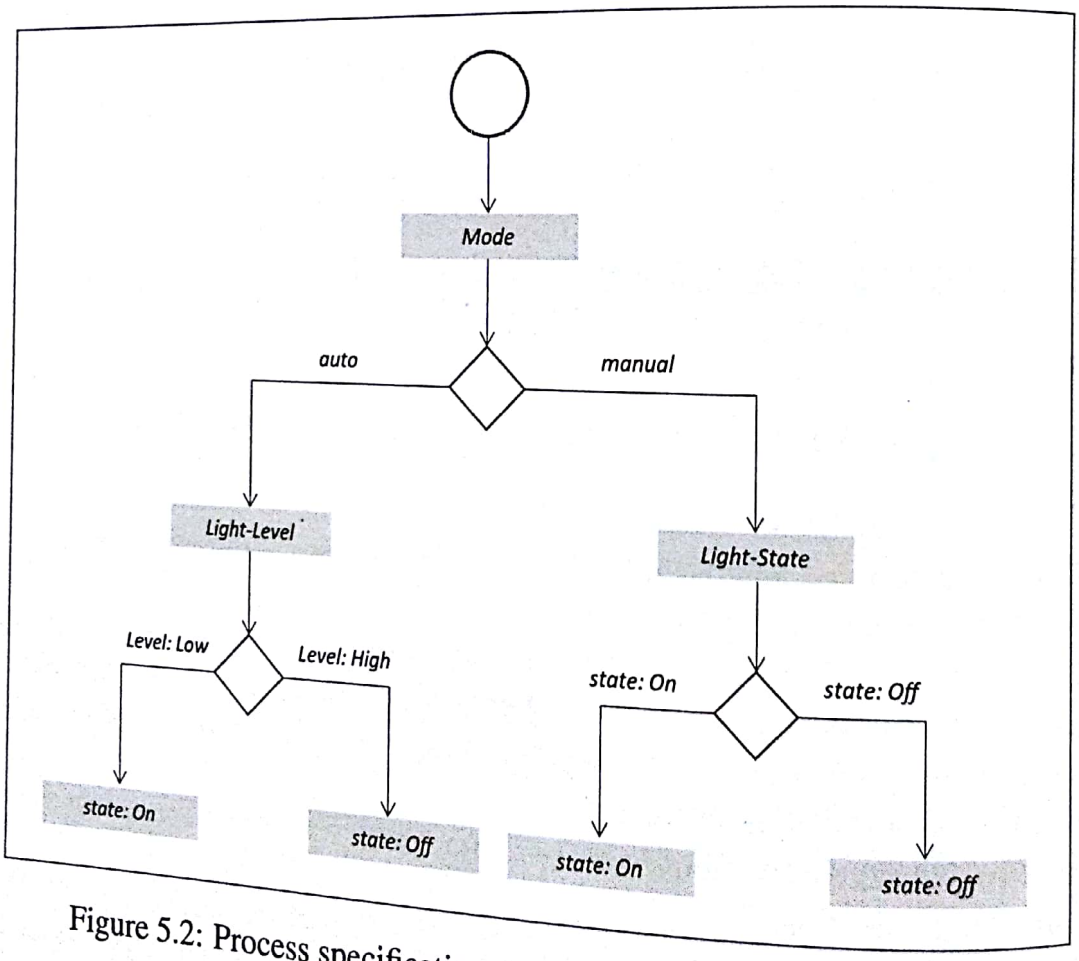

Figure 5.2: Process specification for home automation IoT system

5.2.3 Step 3: Domain Model Specification

The third step in the IoT design methodology is to define the Domain Model. The domain model describes the main concepts, entities and objects in the domain of IoT system to be designed. Domain model defines the attributes of the objects and relationships between objects. Domain model provides an abstract representation of the concepts, objects and entities in the IoT domain, independent of any specific technology or platform. With the domain model, the IoT system designers can get an understanding of the IoT domain for which the system is to be designed. Figure 5.3 shows the domain model for the home automation system example. The entities, objects and concepts defined in the domain model include:

- **Physical Entity :** Physical Entity is a discrete and identifiable entity in the physical environment (e.g. a room, a light, an appliance, a car, etc.). The IoT system provides information about the Physical Entity (using sensors) or performs actuation upon the Physical Entity (e.g., switching on a light). In the home automation example, there are two Physical Entities involved - one is the room in the home (of which the lighting conditions are to be monitored) and the other is the light appliance to be controlled.
- **Virtual Entity :** Virtual Entity is a representation of the Physical Entity in the digital world. For each Physical Entity, there is a Virtual Entity in the domain model. In the home automation example, there is one Virtual Entity for the room to be monitored, another for the appliance to be controlled.
- **Device :** Device provides a medium for interactions between Physical Entities and Virtual Entities. Devices are either attached to Physical Entities or placed near Physical Entities. Devices are used to gather information about Physical Entities (e.g., from sensors), perform actuation upon Physical Entities (e.g. using actuators) or used to identify Physical Entities (e.g., using tags). In the home automation example, the device is a single-board mini computer which has light sensor and actuator (relay switch) attached to it.
- **Resource :** Resources are software components which can be either "on-device" or "network-resources". On-device resources are hosted on the device and include software components that either provide information on or enable actuation upon the Physical Entity to which the device is attached. Network resources include the software components that are available in network (such as a database). In the home automation example, the on-device resource is the operating system that runs on the single-board mini computer.
- **Service :** Services provide an interface for interacting with the Physical Entity. Services access the resources hosted on the device or the network resources to obtain information about the Physical Entity or perform actuation upon the Physical Entity.

In the home automation example, there are three services: (1) a service that sets the light mode to auto or manual, or retrieves the current mode; (2) a service that sets the light appliance state to on/off, or retrieves the current light state; and (3) a controller service that runs as a native service on the device. When in auto mode, the controller service monitors the light level and switches the light on/off and updates the status in the status database. When in manual mode, the controller service retrieves the current state from the database and switches the light on/off. The process of deriving the services from the process specification and information model is described in the later sections.

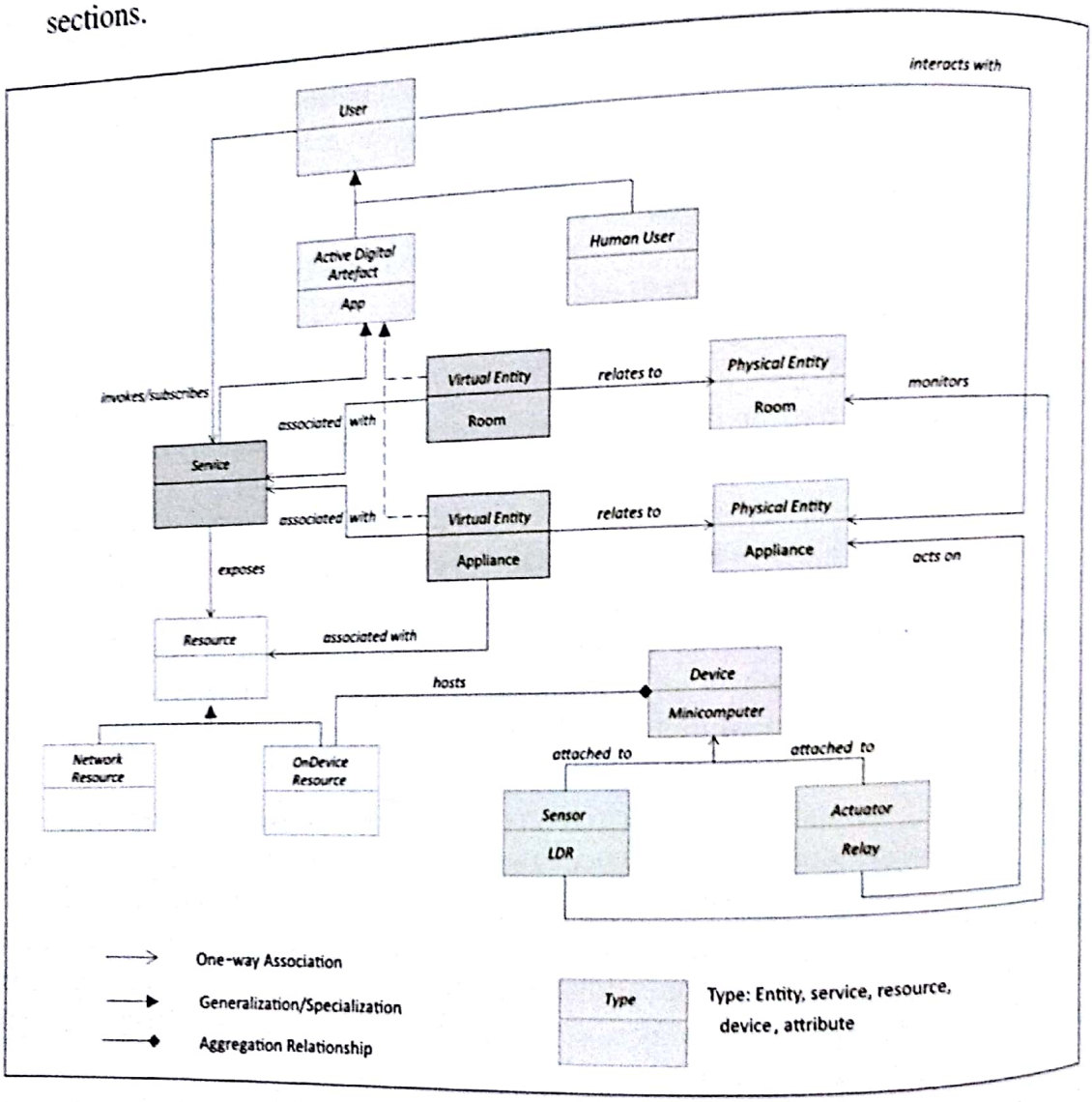

Figure 5.3: Domain model of the home automation IoT system

5.2.4 Step 4: Information Model Specification

The fourth step in the IoT design methodology is to define the Information Model. Information Model defines the structure of all the information in the IoT system, for example, attributes of Virtual Entities, relations, etc. Information model does not describe the specifics of how the information is represented or stored. To define the information model, we first list the Virtual Entities defined in the Domain Model. Information model adds more details to the Virtual Entities by defining their attributes and relations. In the home automation example, there are two Virtual Entities - a Virtual Entity for the light appliance (with attribute - light state) and a Virtual Entity for the room (with attribute - light level). Figure 5.4 shows the Information Model for the home automation system example.

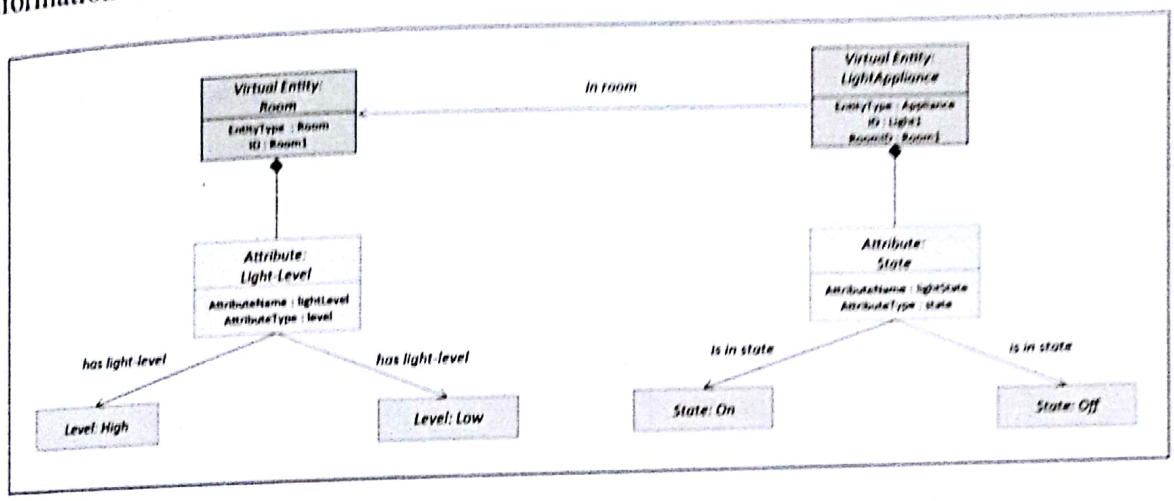

Figure 5.4: Information model of the home automation IoT system

5.2.5 Step 5: Service Specifications

The fifth step in the IoT design methodology is to define the service specifications. Service specifications define the services in the IoT system, service types, service inputs/output, service endpoints, service schedules, service preconditions and service effects.

You learned about the Process Specification and Information Model in the previous sections. Figure 5.5 shows an example of deriving the services from the process specification and information model for the home automation IoT system. From the process specification and information model, we identify the states and attributes. For each state and attribute we define a service. These services either change the state or attribute values or retrieve the current values. For example, the Mode service sets mode to auto or manual or retrieves the current mode. The State service sets the light appliance state to on/off or retrieves the current light state. The Controller service monitors the light level in auto mode and switches the

light on/off and updates the status in the status database. In manual mode, the controller service, retrieves the current state from the database and switches the light on/off.

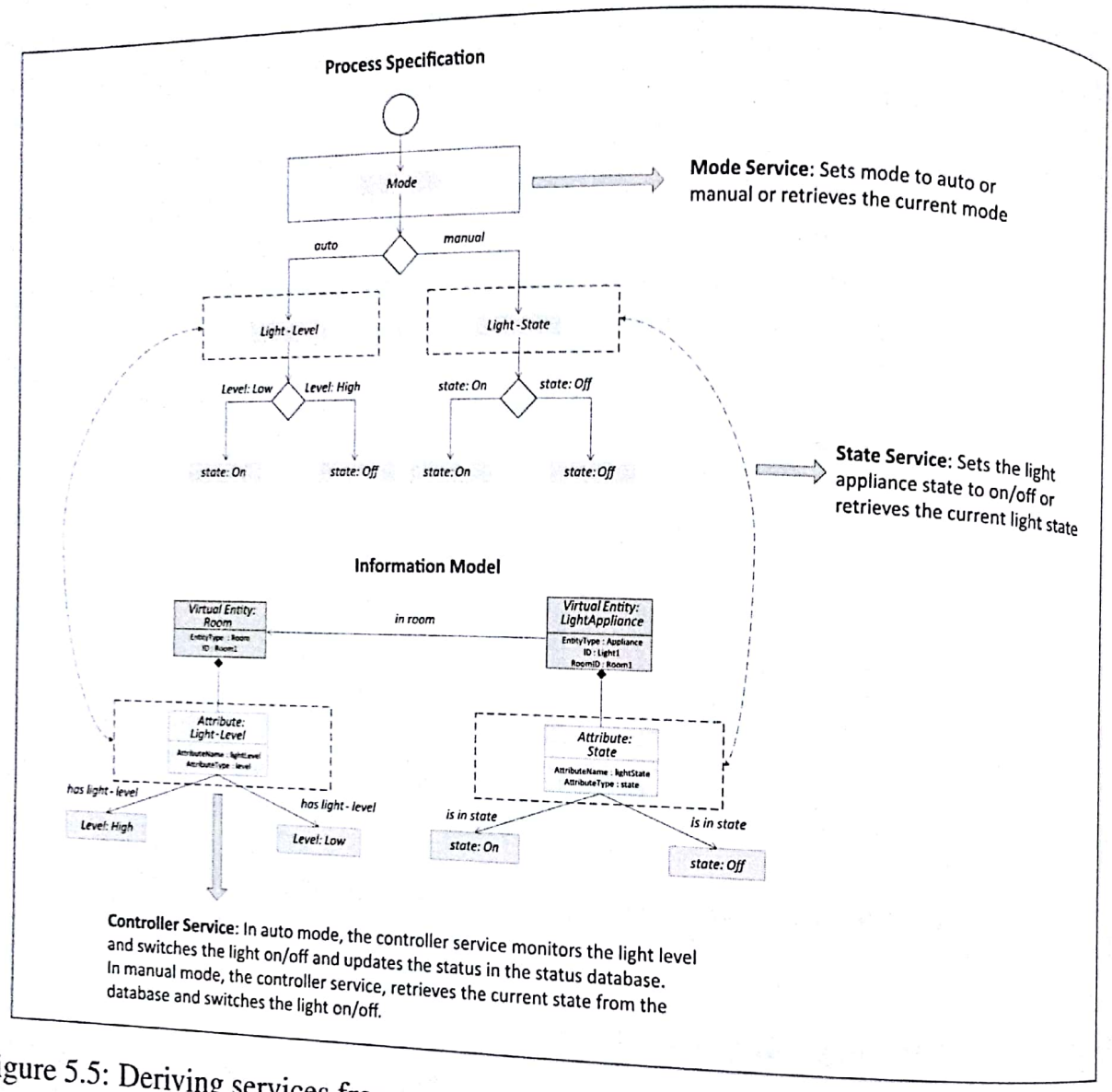

Figure 5.5: Deriving services from process specification and information model for home automation IoT system

Figures 5.6, 5.7 and 5.8 show specifications of the controller, mode and state services of the home automation system. The Mode service is a RESTful web service that sets mode to auto or manual (PUT request), or retrieves the current mode (GET request) . The mode is updated to/retrieved from the database. The State service is a RESTful web service that sets the light appliance state to on/off (PUT request), or retrieves the current light state (GET request). The state is updated to/retrieved from the status database. The Controller service runs as a native service on the device. When in auto mode, the controller service monitors

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

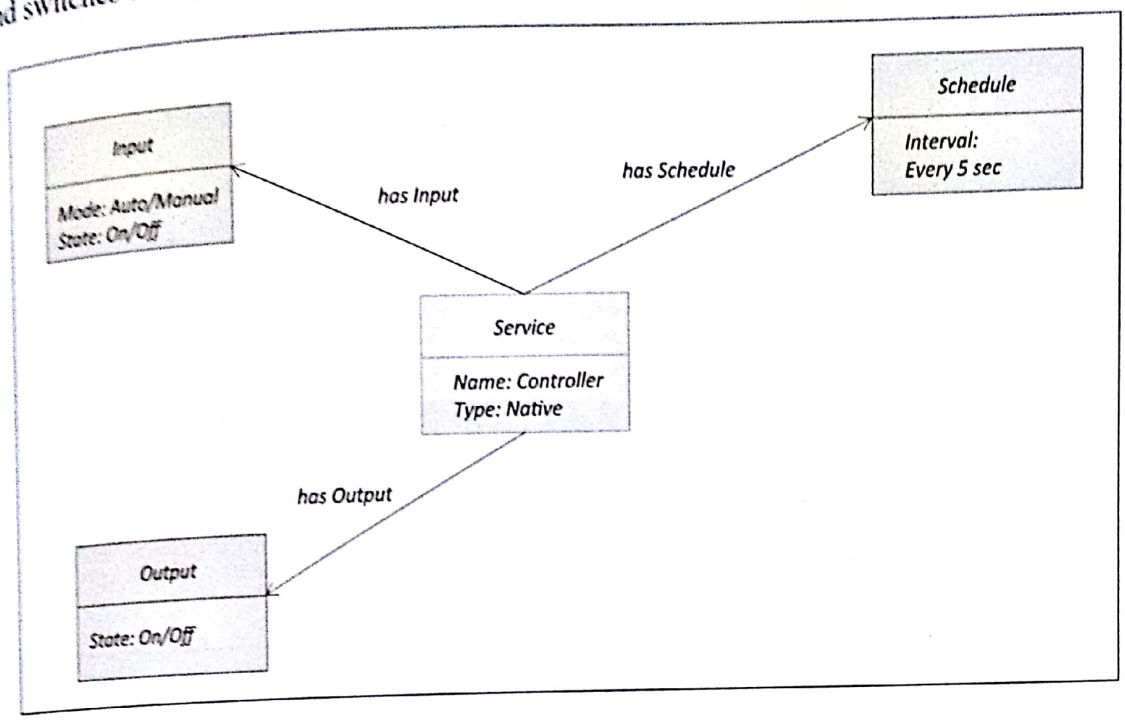

Figure 5.6: Controller service of the home automation IoT system

5.2.6 Step 6: IoT Level Specification

The sixth step in the IoT design methodology is to define the IoT level for the system. In Chapter-1, we defined five IoT deployment levels. Figure 5.9 shows the deployment level of the home automation IoT system, which is level-1.

5.2.7 Step 7: Functional View Specification

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

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

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

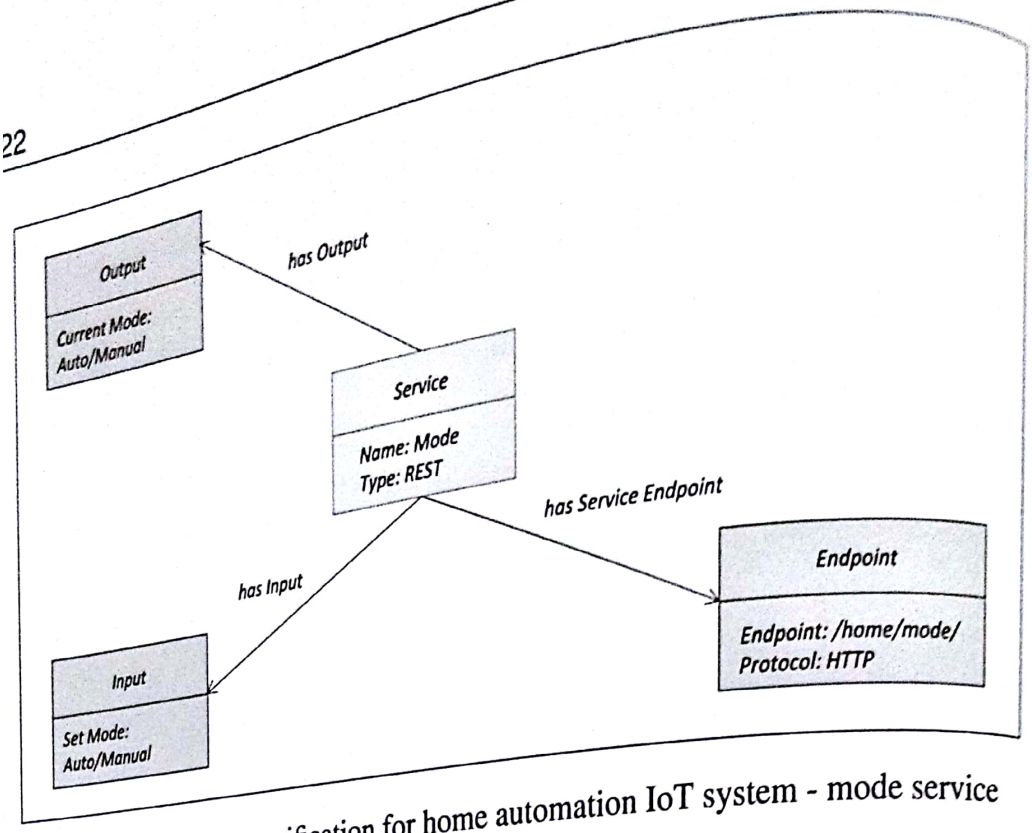

Figure 5.7: Service specification for home automation IoT system - mode service

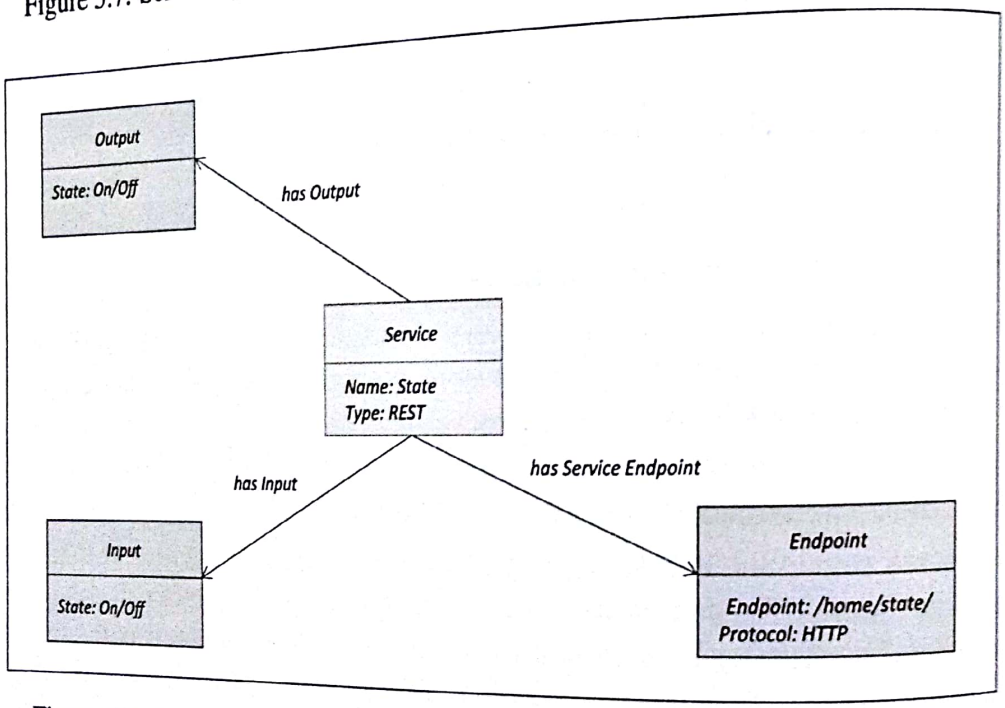

Figure 5.8: Service specification for home automation IoT system - state service

sensor and a relay switch (actuator).

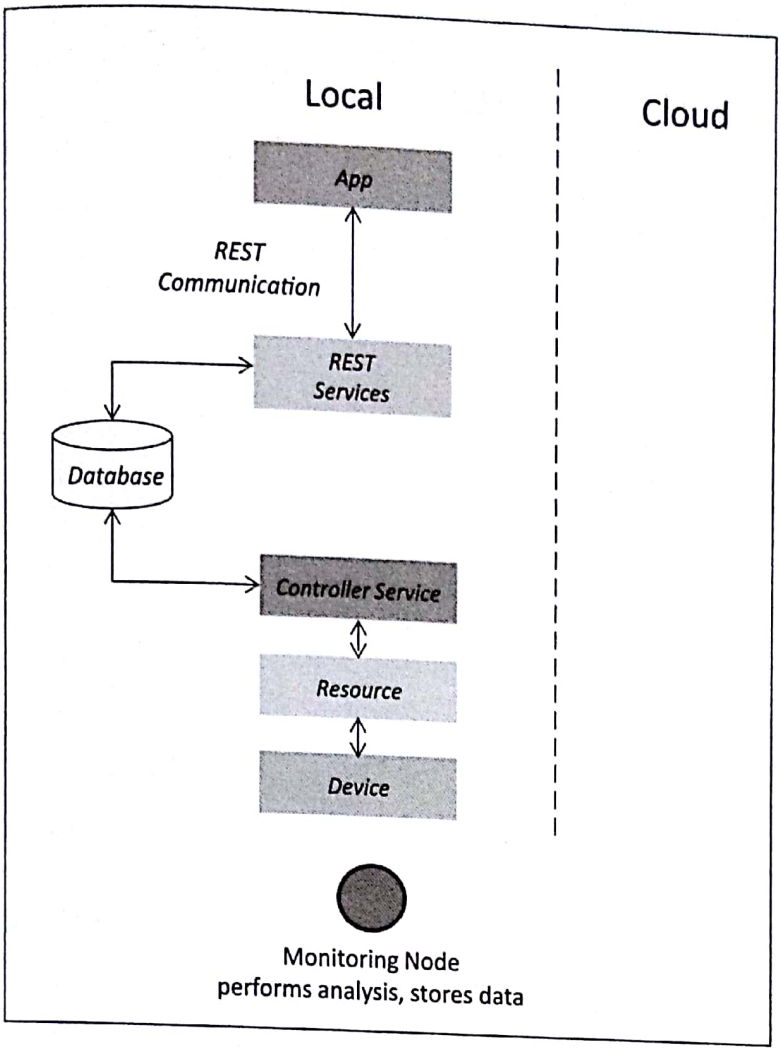

Figure 5.9: Deployment design of the home automation IoT system

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

Figure 5.10: Mapping deployment level to functional groups for home automation IoT system

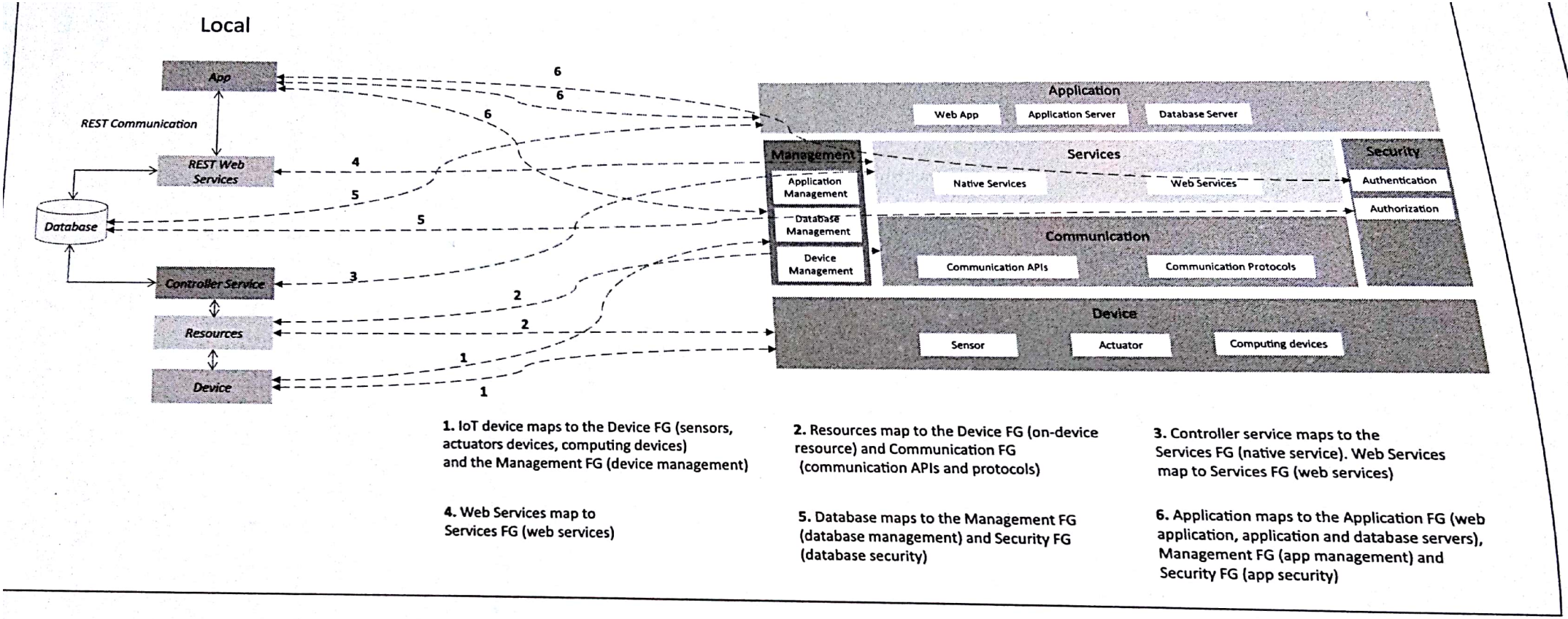

Figure 5.10: Mapping deployment level to functional groups for home automation IoT system

as services for device monitoring, device control services, data publishing services and services for device discovery. In the home automation example, there are two REST services (mode and state service) and one native service (controller service).

- **Management :** The management FG includes all functionalities that are needed to configure and manage the IoT system.
- **Security :** The security FG includes security mechanisms for the IoT system such as authentication, authorization, data security, etc.
- **Application :** The application FG includes applications that provide an interface to the users to control and monitor various aspects of the IoT system. Applications also allow users to view the system status and the processed data.

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

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

5.2.8 Step 8: Operational View Specification

The eighth step in the IoT design methodology is to define the Operational View Specifications. In this step, various options pertaining to the IoT system deployment and operation are defined, such as, service hosting options, storage options, device options, application hosting options, etc.

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

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

- Devices: Computing device (Raspberry Pi), light dependent resistor (sensor), relay switch (actuator).
- Communication APIs: REST APIs
- Communication Protocols: Link Layer - 802.11, Network Layer - IPv4/IPv6, Transport - TCP, Application - HTTP.
- Services:
  1. Controller Service - Hosted on device, implemented in Python and run as a native service.
  2. Mode service - REST-ful web service, hosted on device, implemented with

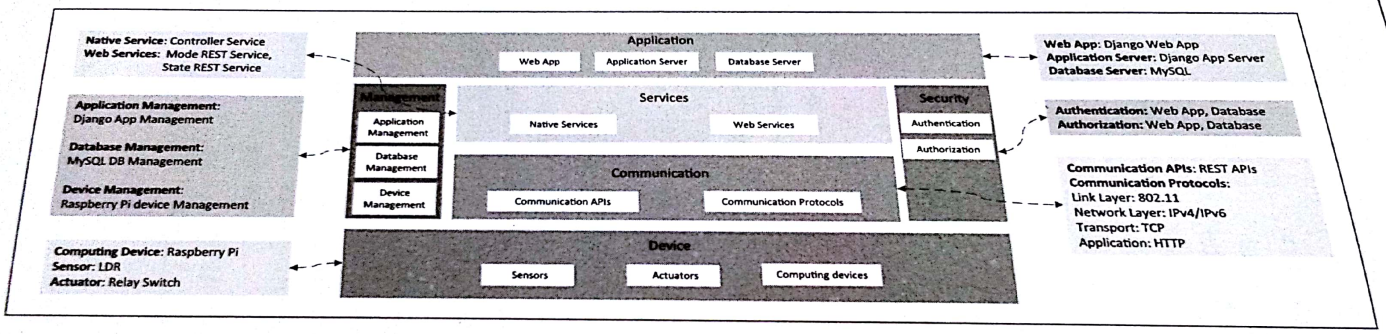

Figure 5.11: Mapping functional groups to operational view for home automation IoT system

Django-REST Framework.

3. State service - REST-ful web service, hosted on device, implemented with Django-REST Framework.

- Application:
  Web Application - Django Web Application,
  Application Server - Django App Server,
  Database Server - MySQL.
- Security:
  Authentication: Web App, Database
  Authorization: Web App, Database

- Management:
  Application Management - Django App Management
  Database Management - MySQL DB Management,
  Device Management - Raspberry Pi device Management.

5.2.9 Step 9: Device & Component Integration

The ninth step in the IoT design methodology is the integration of the devices and components. Figure 5.12 shows a schematic diagram of the home automation IoT system. The devices and components used in this example are Raspberry Pi mini computer, LDR sensor and relay switch actuator. A detailed description of Raspberry Pi board and how to interface sensors and actuators with the board is provided in later chapters.

5.2.10 Step 10: Application Development

The final step in the IoT design methodology is to develop the IoT application. Figure 5.13 shows a screenshot of the home automation web application. The application has controls for the mode (auto on or auto off) and the light (on or off). In the auto mode, the IoT system controls the light appliance automatically based on the lighting conditions in the room. When auto mode is enabled the light control in the application is disabled and it reflects the current state of the light. When the auto mode is disabled, the light control is enabled and it is used for manually controlling the light.

5.3 Case Study on IoT System for Weather Monitoring

In this section we present a case study on design of an IoT system for weather monitoring using the IoT design methodology. The purpose of the weather monitoring system is to collect data on environmental conditions such as temperature, pressure, humidity and light

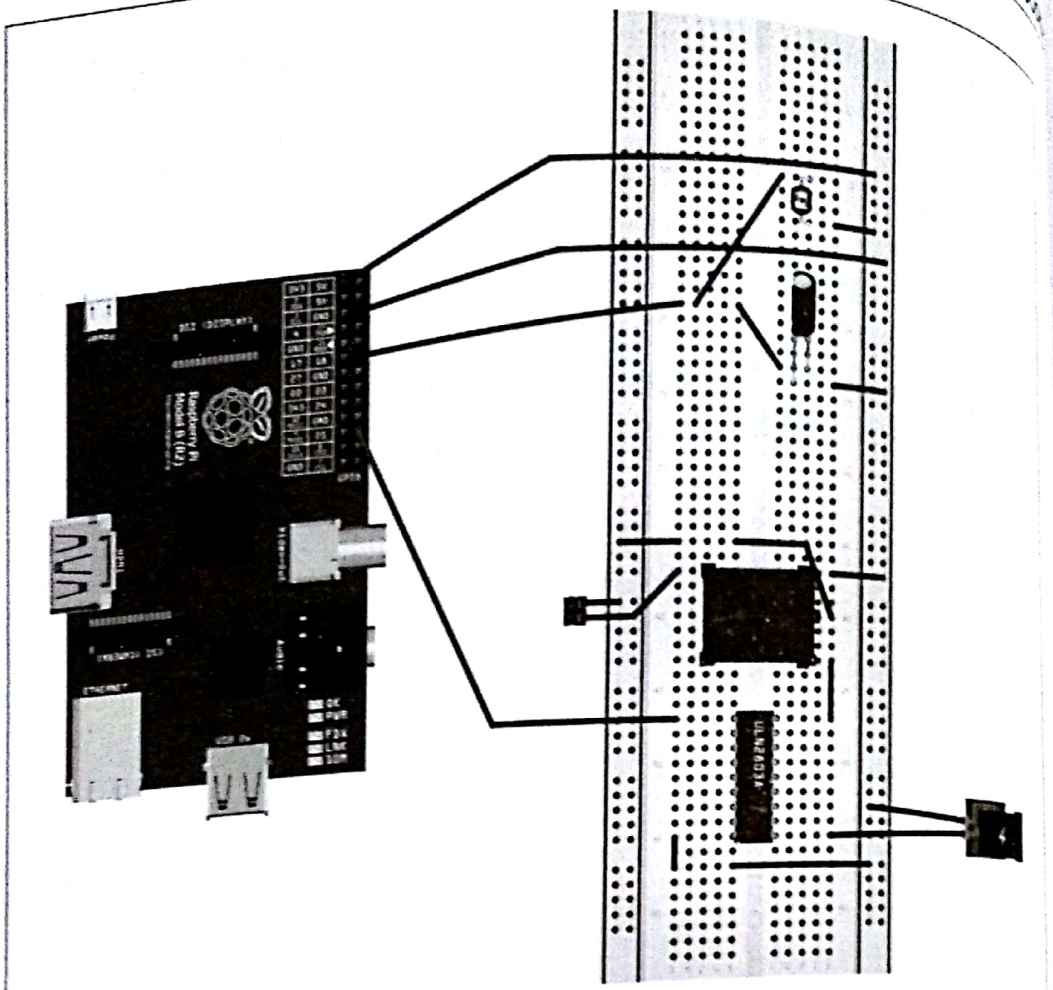

Figure 5.12: Schematic diagram of the home automation IoT system showing the dev
sensor and actuator integrated

in an area using multiple end nodes. The end nodes send the data to the cloud where the
is aggregated and analyzed.

Figure 5.14 shows the process specification for the weather monitoring system. T
process specification shows that the sensors are read after fixed intervals and the sens
measurements are stored.

Figure 5.15 shows the domain model for the weather monitoring system. In this doma
model the physical entity is the environment which is being monitored. There is a virtu
entity for the environment. Devices include temperature sensor, pressure sensor, humidity
sensor, light sensor and single-board mini computer. Resources are software compone
which can be either on-device or network-resources. Services include the controller servic
that monitors the temperature, pressure, humidity and light and sends the readings to

Figure 5.13: Home automation web application screenshot

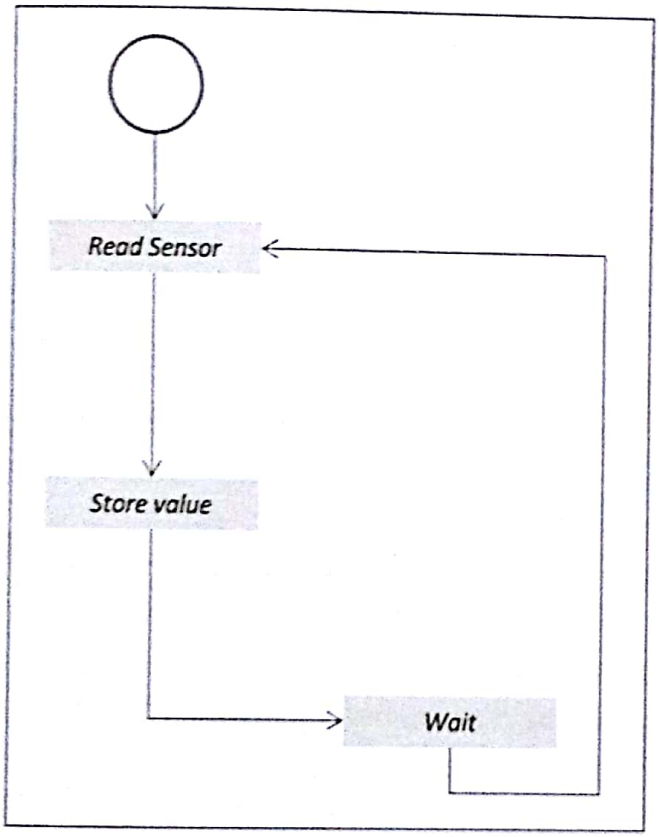

Figure 5.14: Process specification for weather monitoring IoT system

cloud.

Figure 5.16 shows the information model for the weather monitoring system. In this example, there is one virtual entity for the environment being sensed. The virtual entity has attributes - temperature, pressure, humidity and light. Figure 5.17 shows an example of

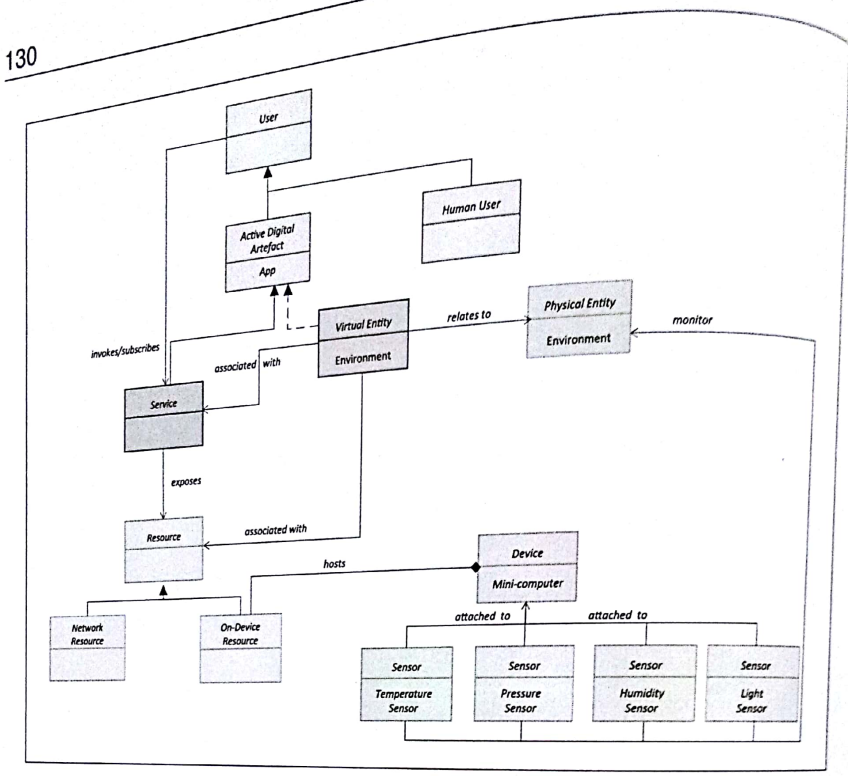

Figure 5.15: Domain model for weather monitoring IoT system

deriving the services from the process specification and information model for the weather monitoring system.

Figure 5.18 shows the specification of the controller service for the weather monitoring system. The controller service runs as a native service on the device and monitors temperature, pressure, humidity and light once every 15 seconds. The controller service calls the REST service to store these measurements in the cloud. In Chapter-8 we describe a Platform-as-a-Service called Xively that can be used for creating solutions for Internet of Things. An implementation of a controller service that calls the Xively REST API to store data in Xively cloud is described in Chapter-9.

Figure 5.19 shows the deployment design for the system. The system consists of multiple nodes placed in different locations for monitoring temperature, humidity and pressure in an area. The end nodes are equipped with various sensors (such as temperature, pressure, humidity and light). The end nodes send the data to the cloud and the data is stored in a cloud

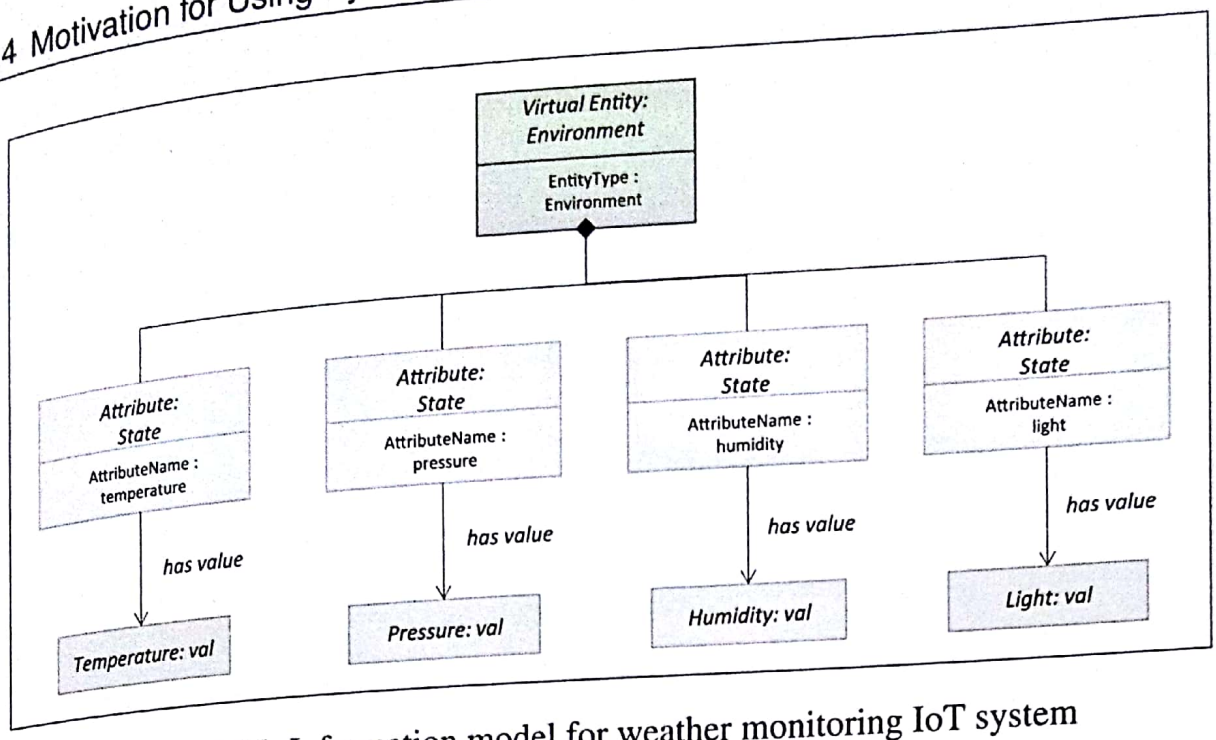

Figure 5.16: Information model for weather monitoring IoT system

database. The analysis of data is done in the cloud to aggregate the data and make predictions. A cloud-based application is used for visualizing the data. The centralized controller can send control commands to the end nodes, for example, to configure the monitoring interval on the end nodes.

Figure 5.20 shows an example of mapping deployment level to functional groups for the weather monitoring system. Figure 5.21 shows an example of mapping functional groups to operational view specifications for the weather monitoring system.

Figure 5.22 shows a schematic diagram of the weather monitoring system. The devices and components used in this example are Raspberry Pi mini computer, temperature sensor, humidity sensor, pressure sensor and LDR sensor.

5.4 Motivation for Using Python

This book uses the Python language for all the examples, though the basic principles apply to other high level languages. In this section we explain the motivation for using Python for developing IoT systems. Python is a minimalistic language with English-like keywords and fewer syntactical constructions as compared to other languages. This makes Python easier to learn and understand. Moreover, Python code is compact as compared to other languages. Python is an interpreted language and does not require an explicit compilation step. The Python interpreter converts the Python code to the intermediate byte code, specific

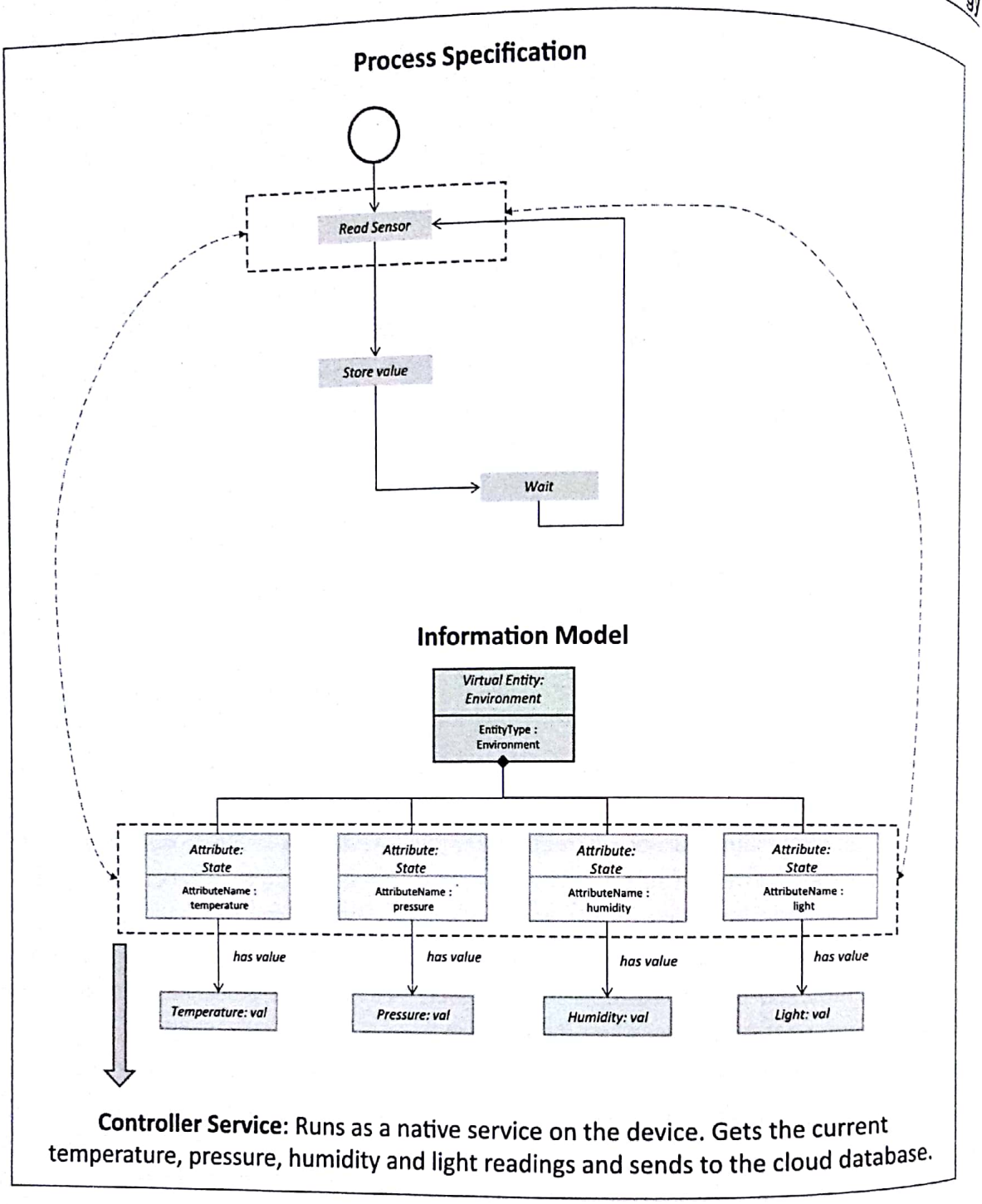

Figure 5.17: Deriving services from process specification and information model for weather monitoring IoT system

to the system. Python is supported on wide range of platforms, hence Python code is easily

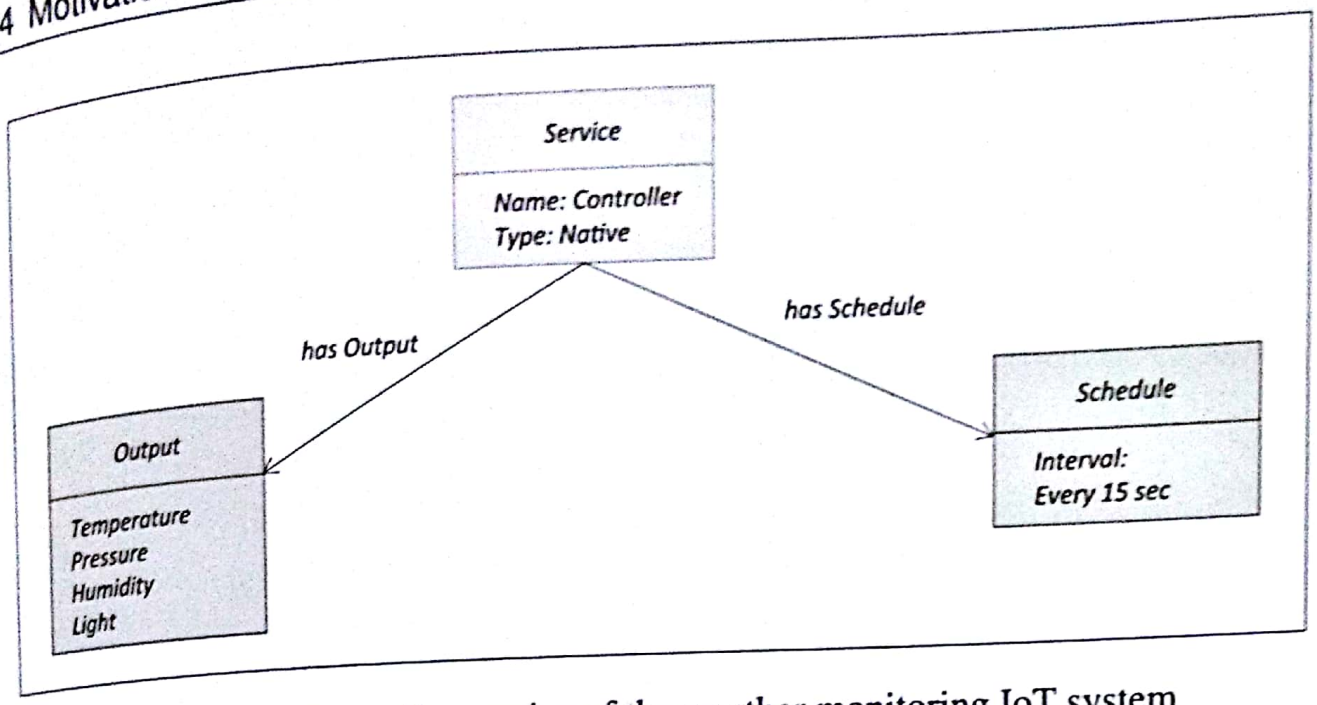

Figure 5.18: Controller service of the weather monitoring IoT system

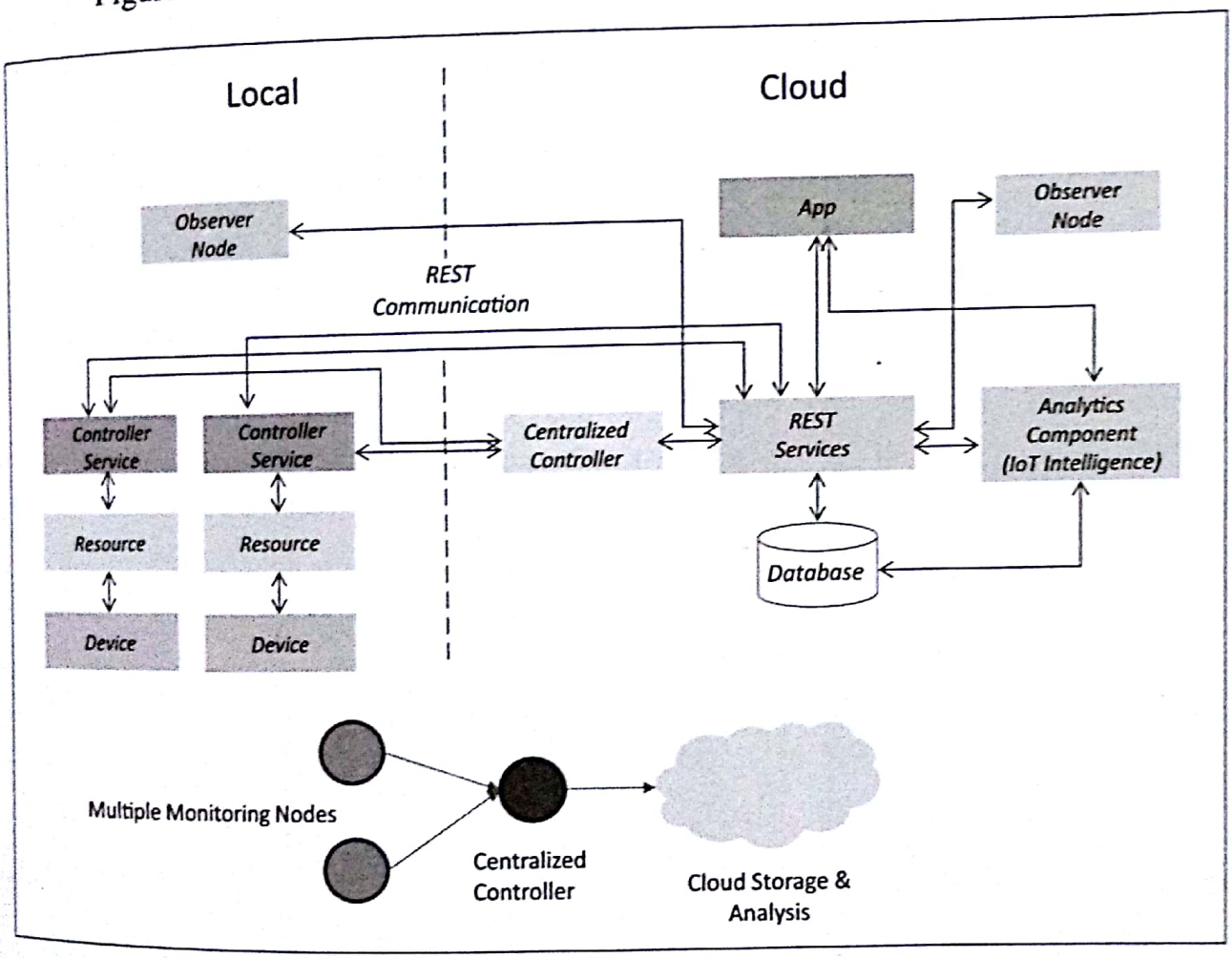

Figure 5.19: Deployment design of the weather monitoring IoT system

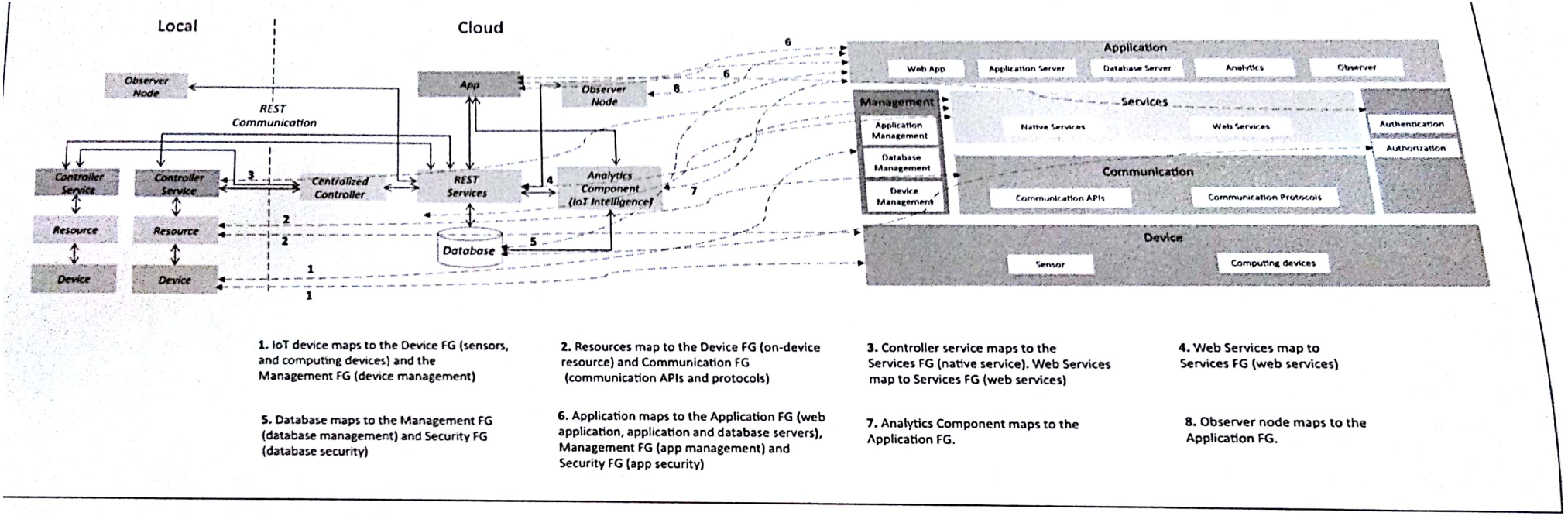

Figure 5.20: Mapping deployment level to functional groups for the weather monitoring IoT system

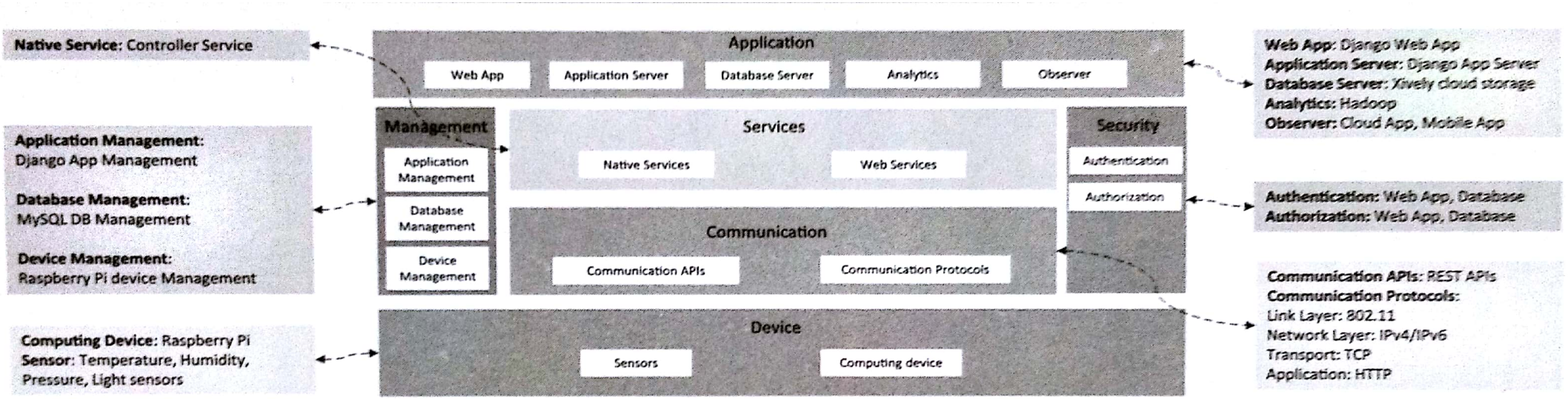

Figure 5.21: Mapping functional groups to operational view for the weather monitoring IoT system

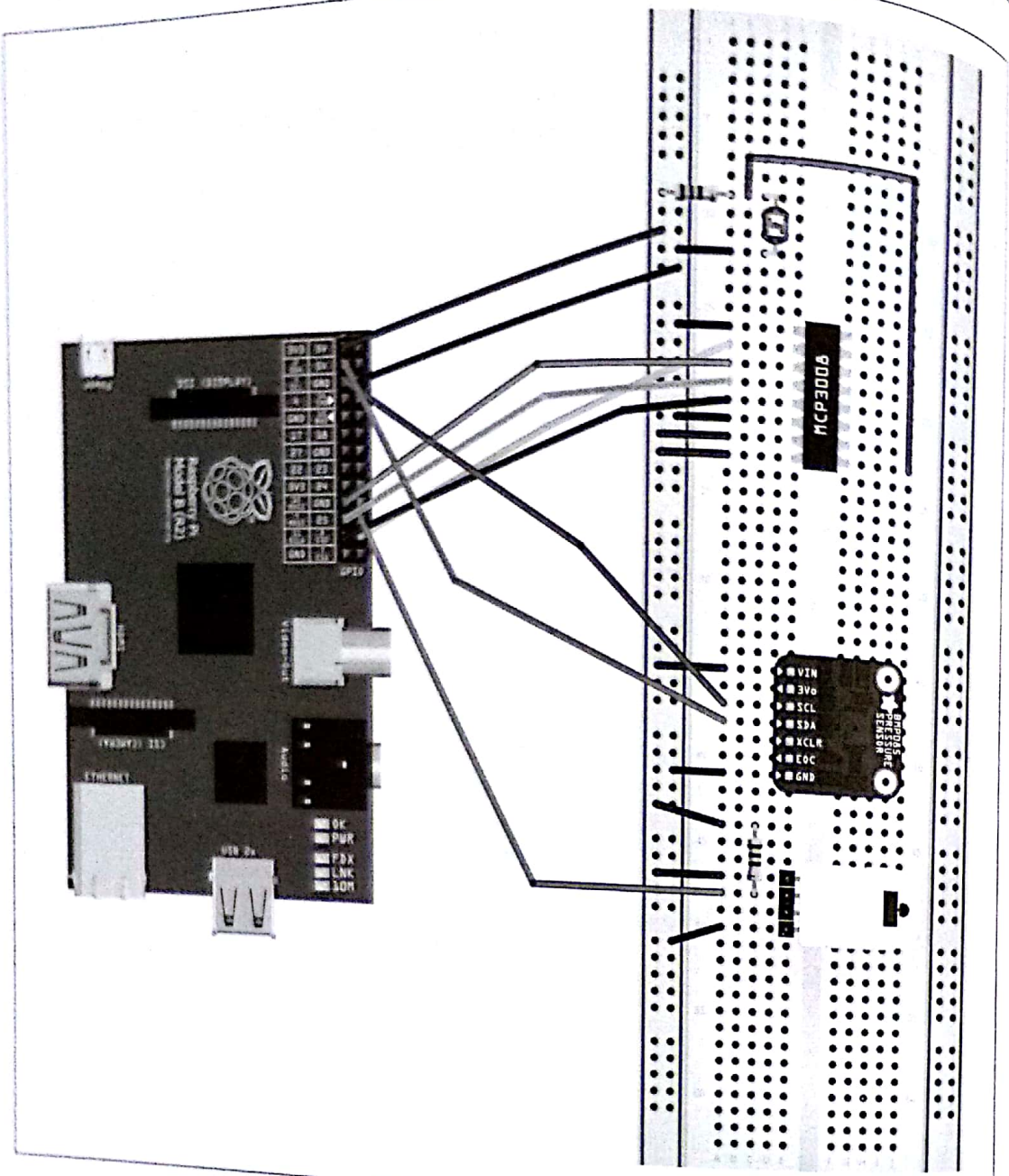

Figure 5.22: Schematic diagram of a weather monitoring end-node showing the device and sensors

portable. The wide library support available for Python makes it an excellent choice for IoT systems. Python can be used for end-to-end development of IoT systems from IoT device code (e.g. code for capturing sensor data), native services (e.g., a controller service implemented in Python), web services (e.g. a RESTful web service implemented in Python),