IoT Design Methodology

Notes taken from IoT Hands-On Approach, A. Bahga and V. Madisetti

Ontario Tech University

Learning Outcomes

- A simple and generic design methodology is presented for IoT applications.
- The example is based on the Django framework.
  - It is more like a web application than truly an IoT application but it contains most of the major components.
- For more information on Django see
  - Python Django Tutorial for Beginners - YouTube

OntarioTech UNIVERSITY

Background

- IoT Applications can be complex to design and build because they involve devices, network resources, web services, analytics components, applications, and database servers.
- In this learning module we go over a generic design methodology for IoT systems.
- The approach is generally based on the IoT-A reference model.

IoT Design Methodology - Steps

1. **Purpose & Requirements**
   Define Purpose & Requirements of IoT system
2. **Process Model Specification**
   Define the use cases
3. **Domain Model Specification**
   Define Physical Entities, Virtual Entities, Devices, Resources and Services in the IoT system
4. **Information Model Specification**
   Define the structure (e.g. relations, attributes) of all the information in the IoT system
5. **Service Specifications**
   Map Process and Information Model to services and define service specifications
6. **IoT Level Specification**
   Define the IoT level for the system
7. **Functional View Specification**
   Map IoT Level to functional groups
8. **Operational View Specification**
   Define communication options, service hosting options, storage options, device options
9. **Device & Component Integration**
   Integrate devices, develop and integrate the components
10. **Application Development**
    Develop Applications

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.

OntarioTech UNIVERSITY

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.

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.

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.

OntarioTech UNIVERSITY

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.

Ontario Tech UNIVERSITY

Step 6: IoT Level Specification

- The sixth step in the IoT design methodology is to define the IoT levels for the system.
- In the IoT Architecture Overview module we covered several IoT deployment levels.
  - Recall IoT Levels 1 to 6 from the IoT Architecture Overview lesson.

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.

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

Step 9: Device & Component Integration

- The ninth step in the IoT design methodology is the integration of the devices and components.

Step 10: Application Development

- The final step in the IoT design methodology is to develop the IoT application.

Home Automation Case Study

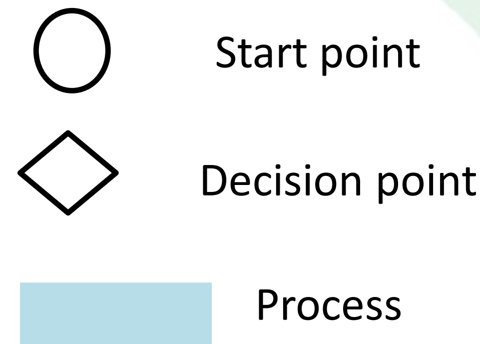
Step:1 - Purpose & Requirements

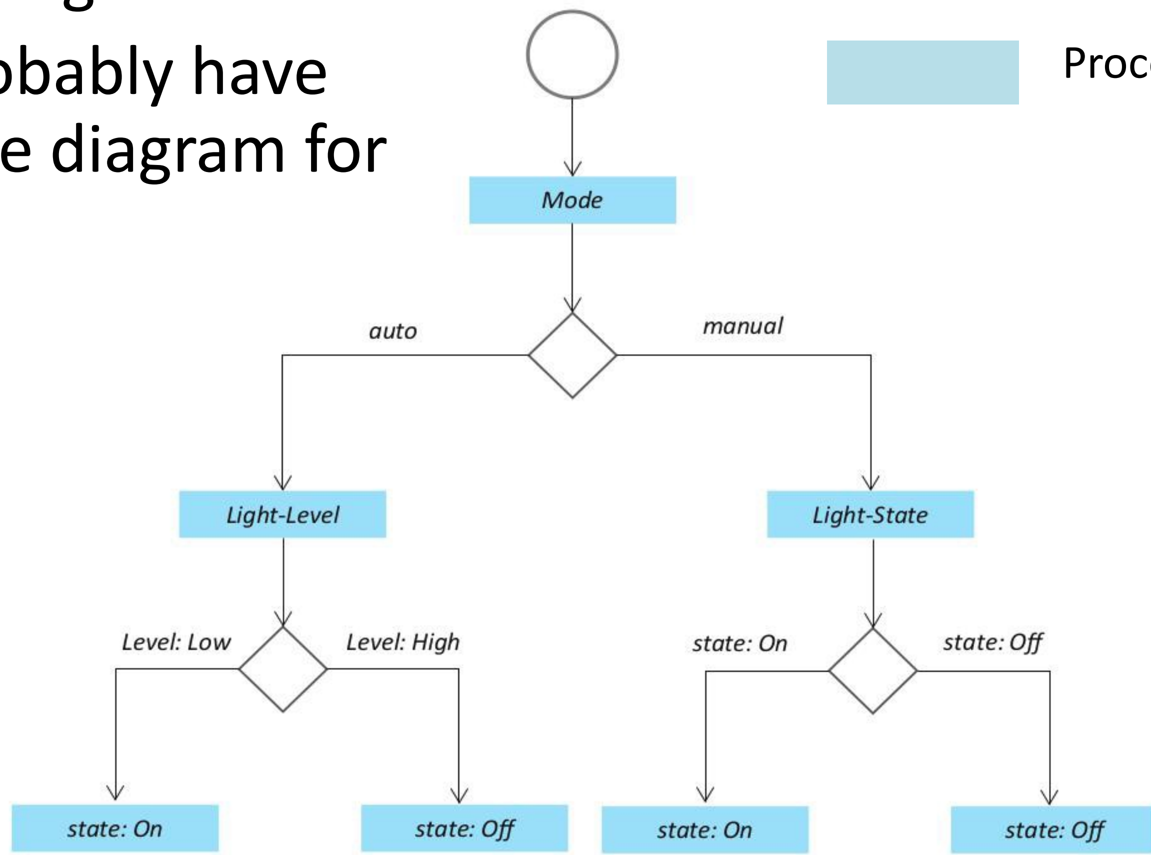
- Applying this to our example of a smart home automation system, the purpose and requirements for the system may be described as follows:
  - **Purpose** : A home automation system that allows controlling of the lights in a home remotely using a web application.
  - **Behavior** : The home automation 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.

OntarioTech UNIVERSITY

Step:2 - Process Specification

- In this stage the Use Cases are captured using a process diagram.
- I would probably have used a state diagram for this.

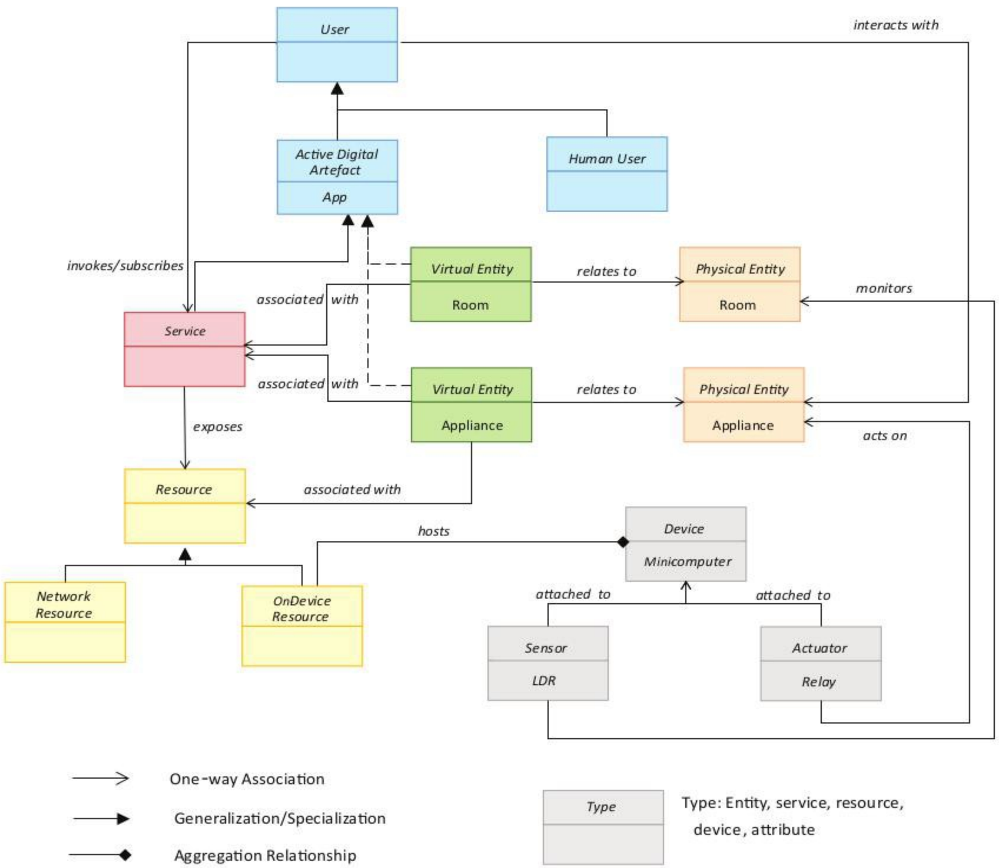




OntarioTech UNIVERSITY

Step 3: Domain Model Specification

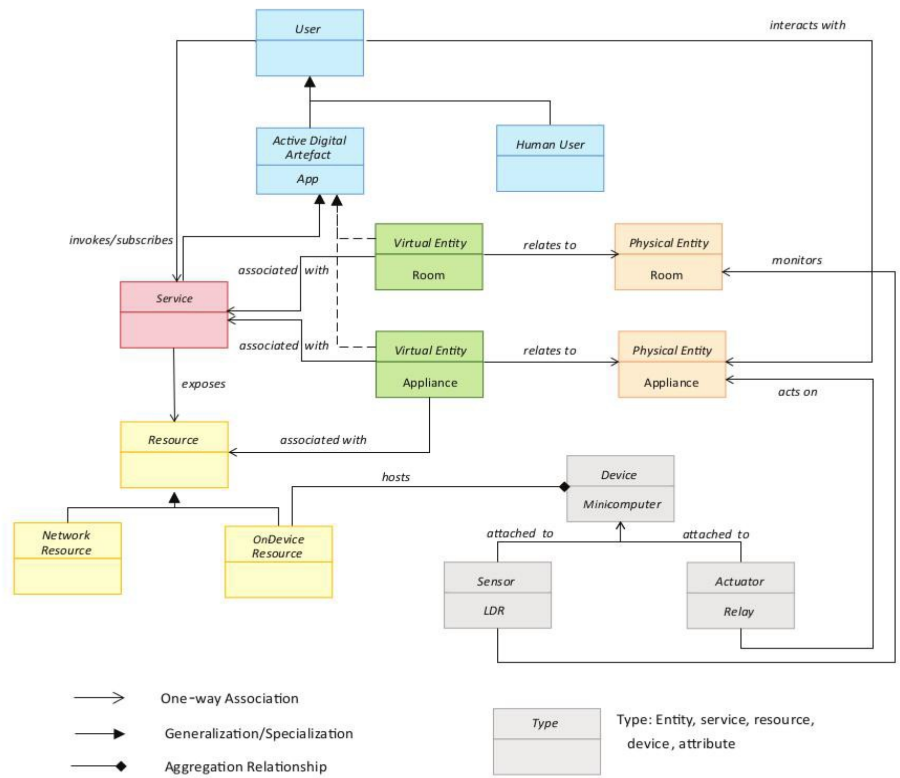
- These Domain models are generally challenging to define.



OntarioTech UNIVERSITY

Step 3: Domain Model Specification

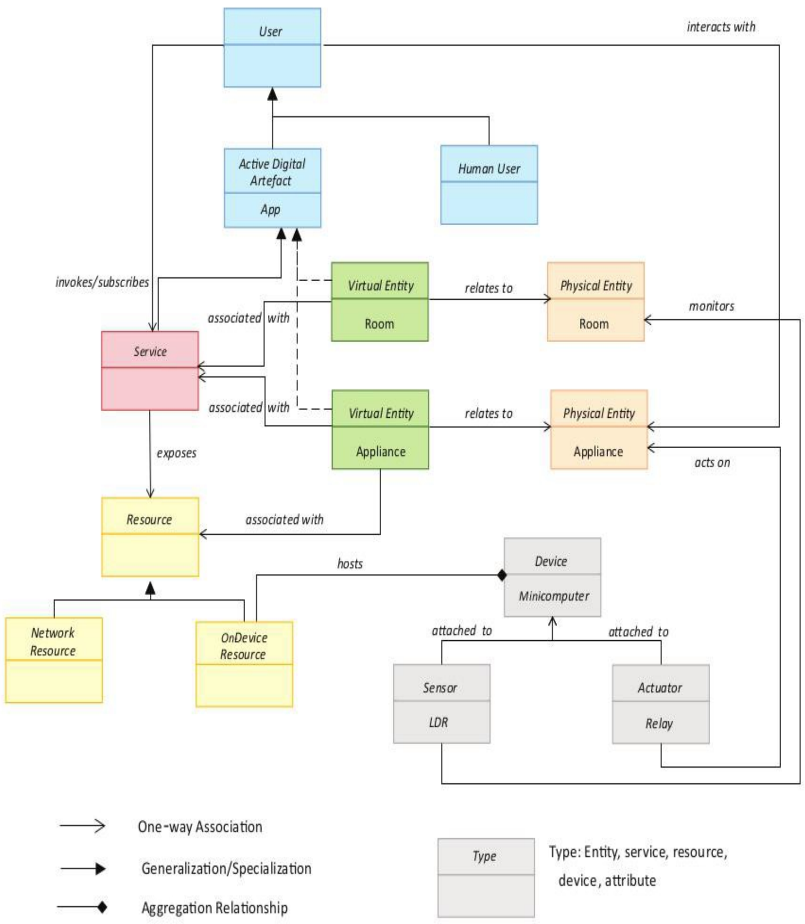
- The Domain model defines the main concepts, entities, and objects on the system.
  - Physical entities (Orange): This is a physical device. A room and light appliance.
  - Virtual entities (Green): Representation of the physical entity in the digital world.



OntarioTech UNIVERSITY

Step 3: Domain Model Specification

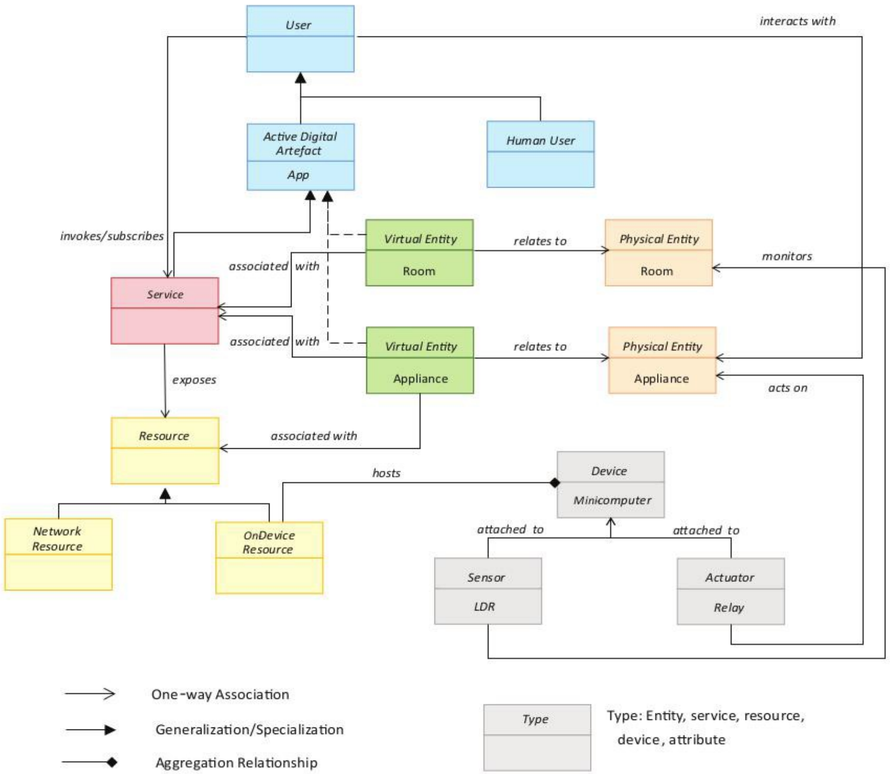
- The Domain model defines the main concepts, entities, and objects on the system.
  - Device (Grey): A medium for interacting between the physical and virtual entities. They are the interfaces to the physical entities.
  - Resource (Yellow): These are representative of Operating system for the Device or networked resources like DB.



OntarioTech UNIVERSITY

Step 3: Domain Model Specification

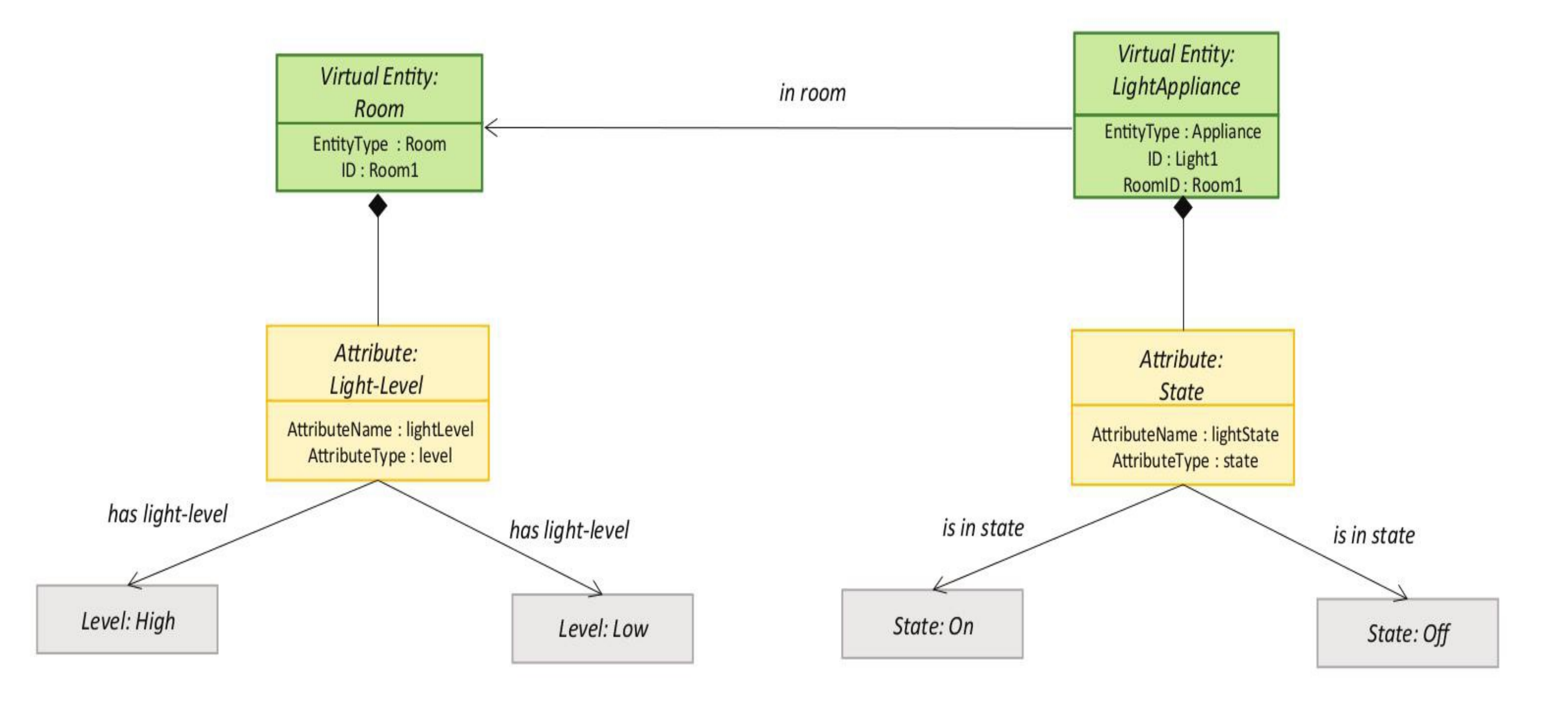
- Service (Pink): These provide an interface for interacting with the Physical entities such as mode setting, light setting, a light controller.
- This will typically be the business logic I am trying to create for the application defined as services.



OntarioTech UNIVERSITY

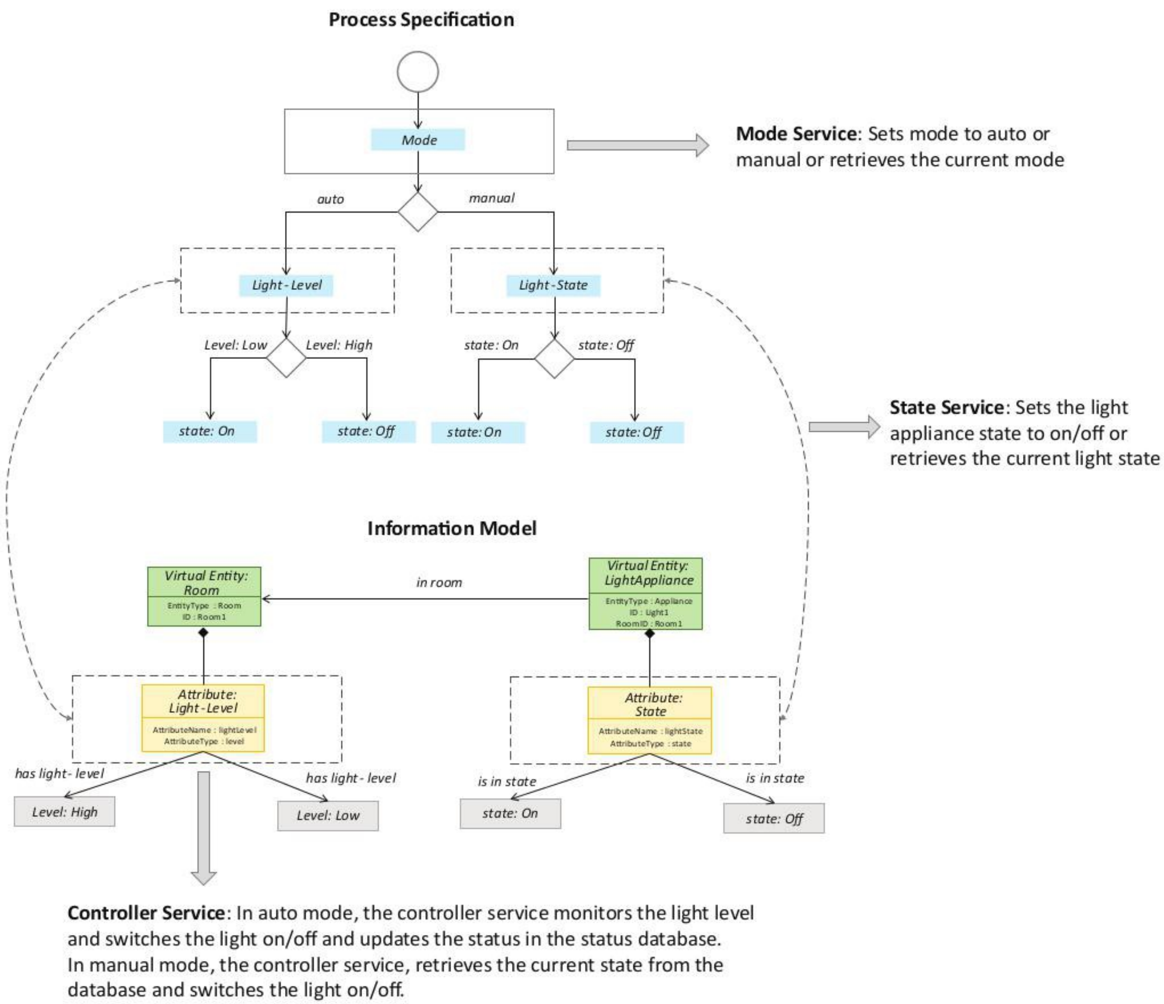
Step 4: Information Model Specification

- Defines a structure of all the information in the system.
  - Start with the virtual entities and add more details by defining attributes and relations.



Step 5: Service Specifications

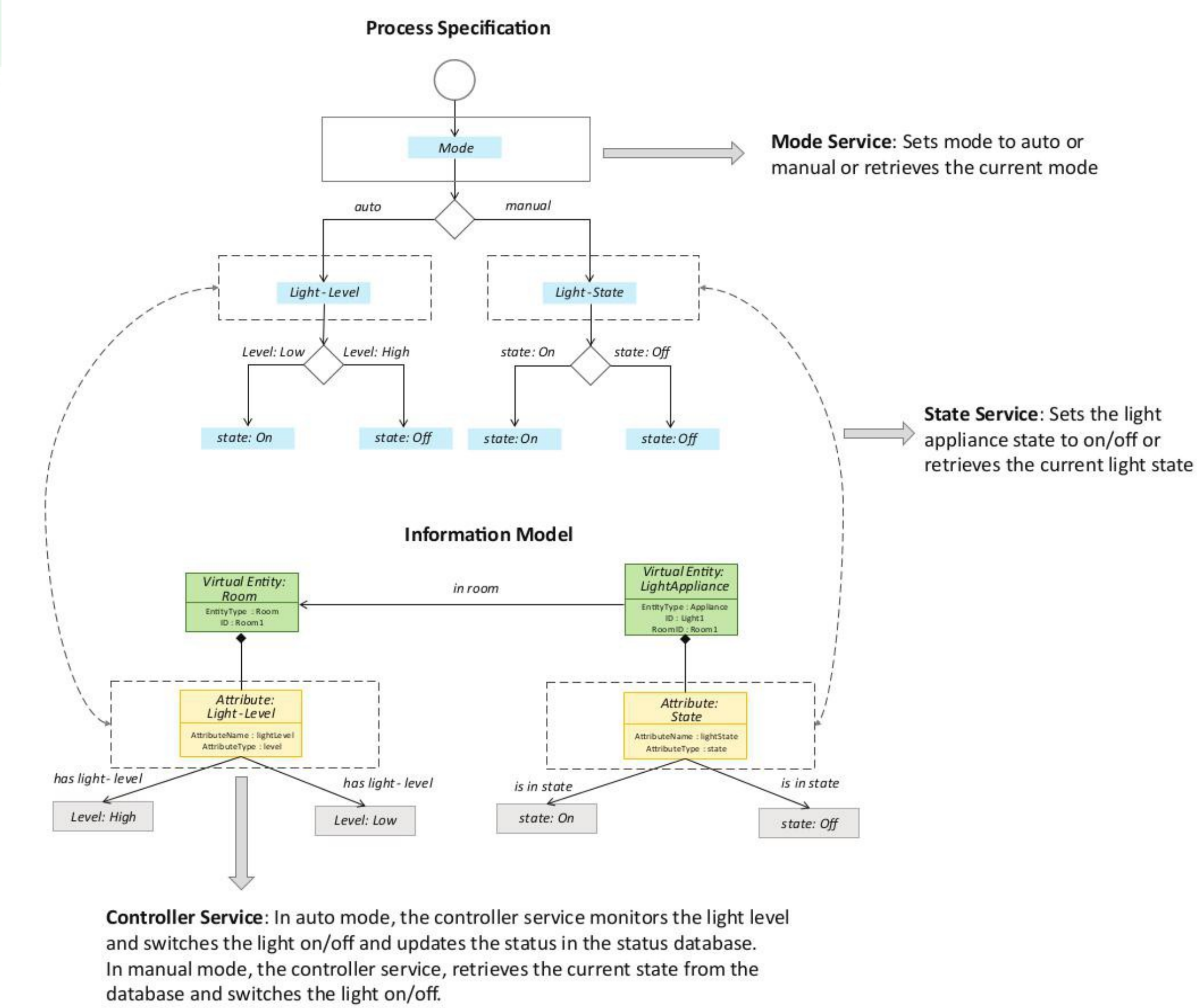
- Because a service model is followed one has to define:
  - Service types, inputs/outputs, endpoints, schedules, pre-conditions, and effects.



OntarioTech UNIVERSITY

Step 5: Service Specifications

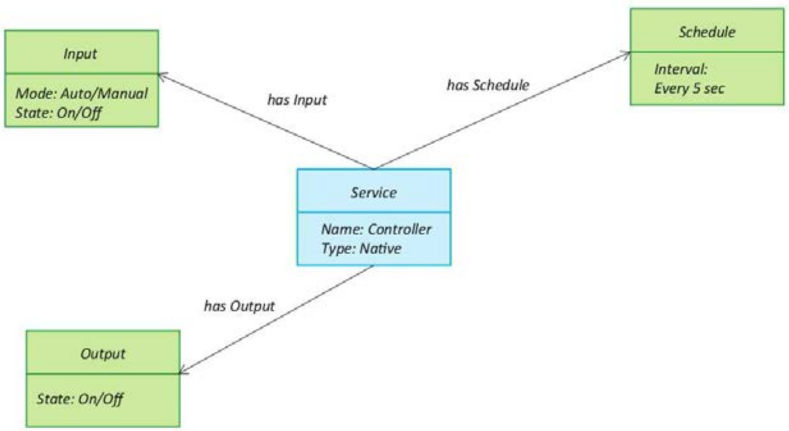
- For each state and attribute a service is defined that can change that state or attribute.
  - Mode changes mode of the system.
  - State changes light state of the light to On/Off
  - Controller monitors light level in the room and turns light On or Off.



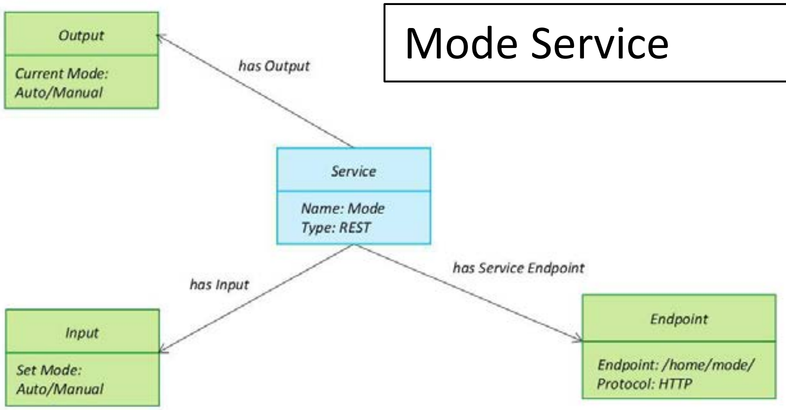
OntarioTech UNIVERSITY

Step 5: Service Specifications

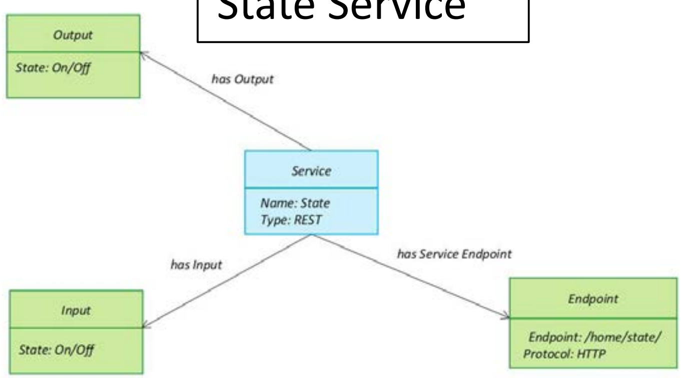
Controller Service



Mode Service



State Service



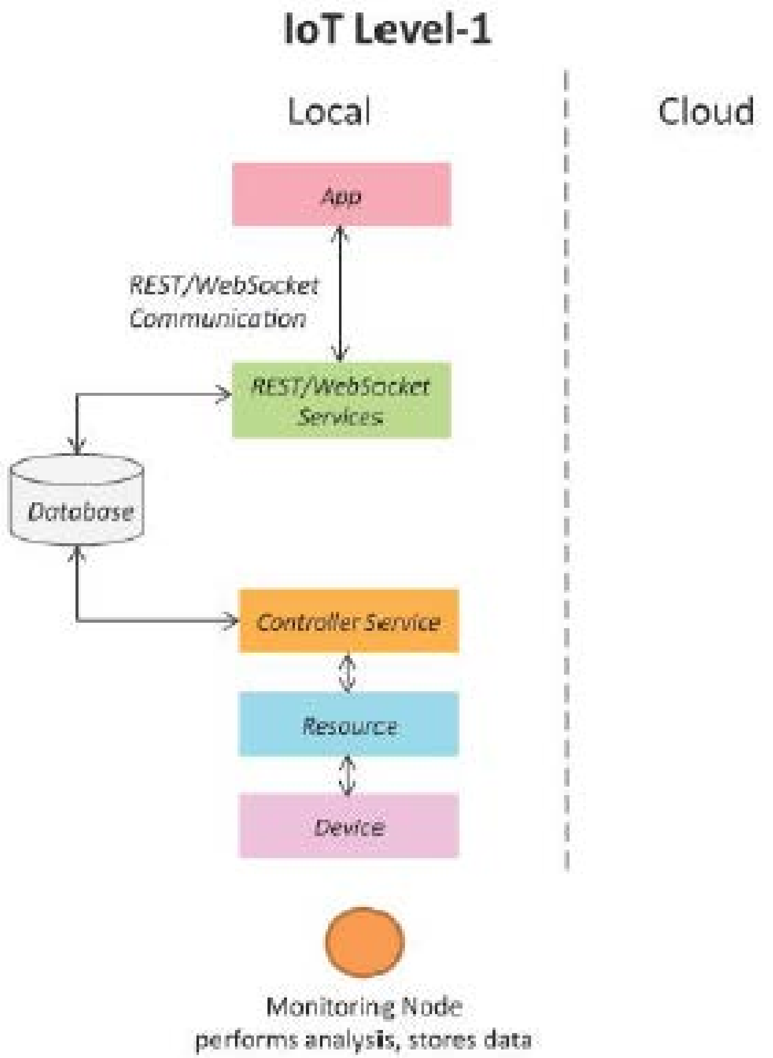
These are defined as RESTful web services

Step 6: IoT Deployment Level Specification

- Prior to discussing deployment implementation we need to present some different deployment levels.

IoT Level-1

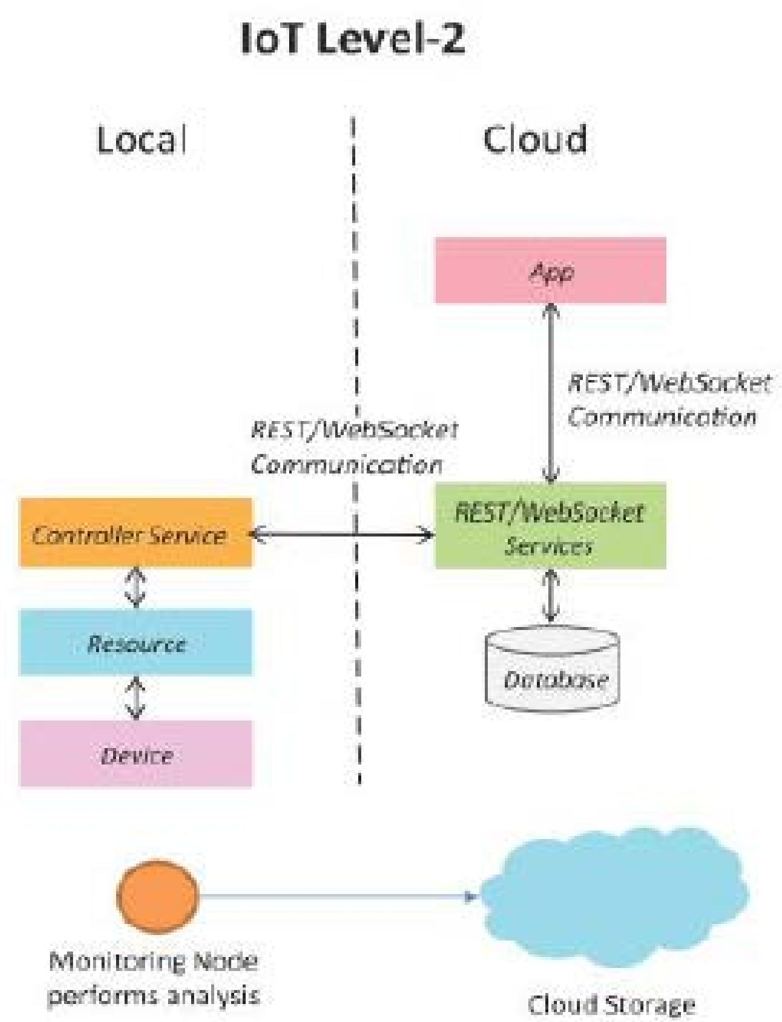
- A level-1 IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application
- Level-1 IoT systems are suitable for modeling low-cost and low-complexity solutions where the data involved is not big and the analysis requirements are not computationally intensive.



Book website: http://www.internet-of-things-book.com

IoT Level-2

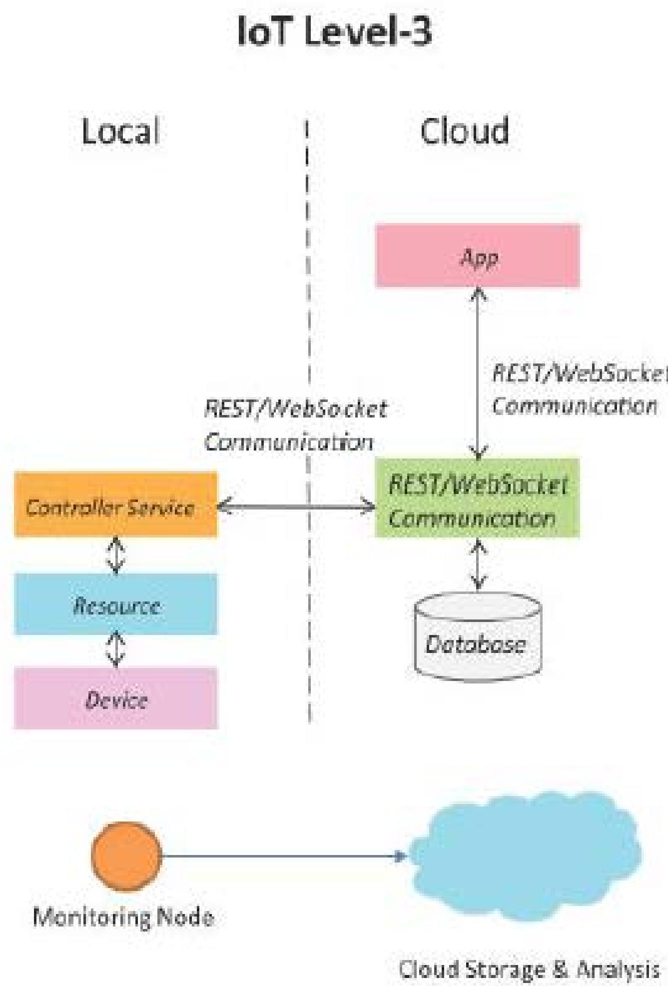
- A level-2 IoT system has a single node that performs sensing and/or actuation and local analysis.
- Data is stored in the cloud and application is usually cloud- based.
- Level-2 IoT systems are suitable for solutions where the data involved is big, however, the primary analysis requirement is not computationally intensive and can be done locally itself.



Book website: http://www.internet-of-things-book.com

IoT Level-3

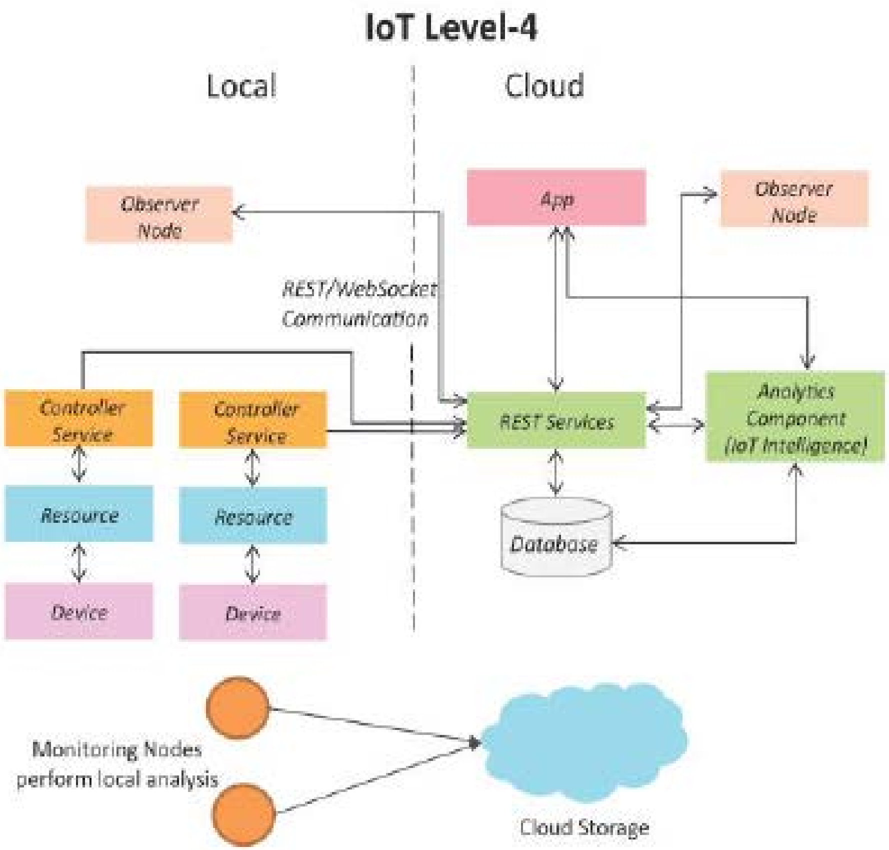
- A level-3 IoT system has a single node. Data is stored and analyzed in the cloud and application is cloud-based.
- Level-3 IoT systems are suitable for solutions where the data involved is big and the analysis requirements are computationally intensive.



Book website: http://www.internet-of-things-book.com

IoT Level-4

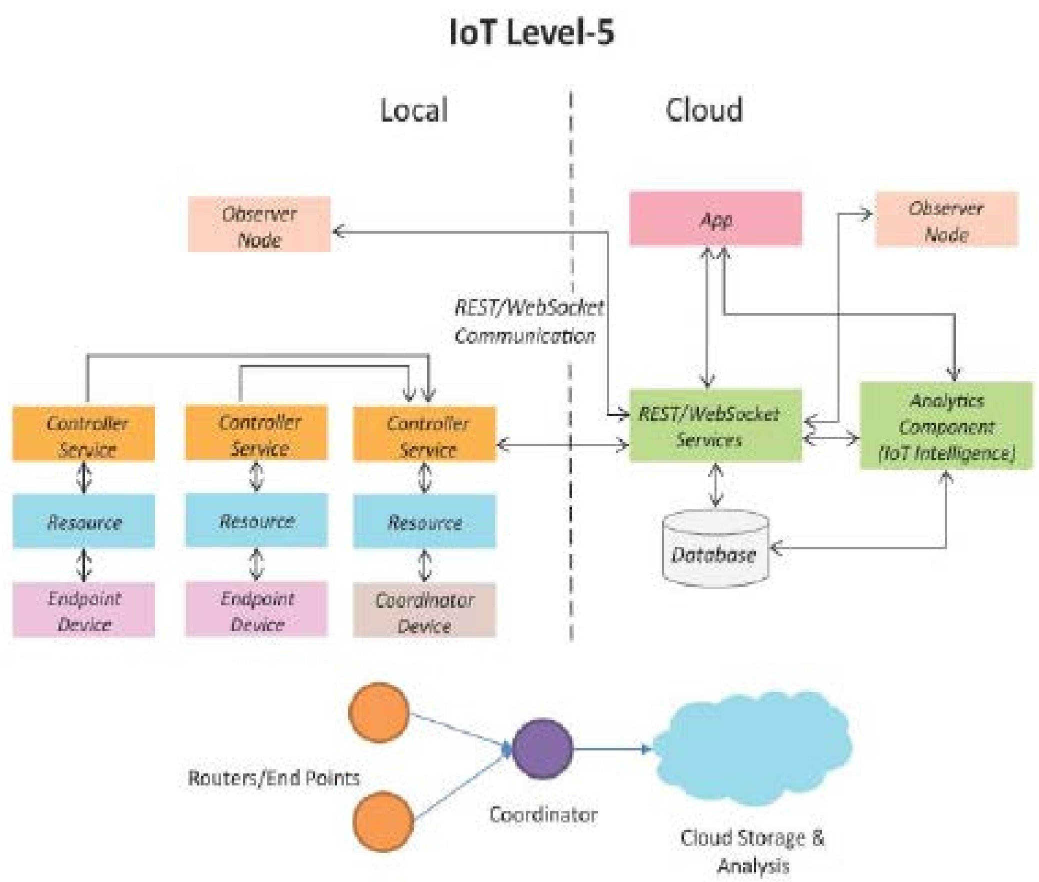
- A level-4 IoT system has multiple nodes that perform local analysis. Data is stored in the cloud and application is cloud-based.
- Level-4 contains local and cloud- based observer nodes which can subscribe to and receive information collected in the cloud from IoT devices.
- Level-4 IoT systems are suitable for solutions where multiple nodes are required, the data involved is big and the analysis requirements are computationally intensive.



Book website: http://www.internet-of-things-book.com

IoT Level-5

- A level-5 IoT system has multiple end nodes and one coordinator node.
- The end nodes that perform sensing and/or actuation.
- Coordinator node collects data from the end nodes and sends to the cloud.
- Data is stored and analyzed in the cloud and application is cloud-based.
- Level-5 IoT systems are suitable for solutions based on wireless sensor networks, in which the data involved is big and the analysis requirements are computationally intensive.

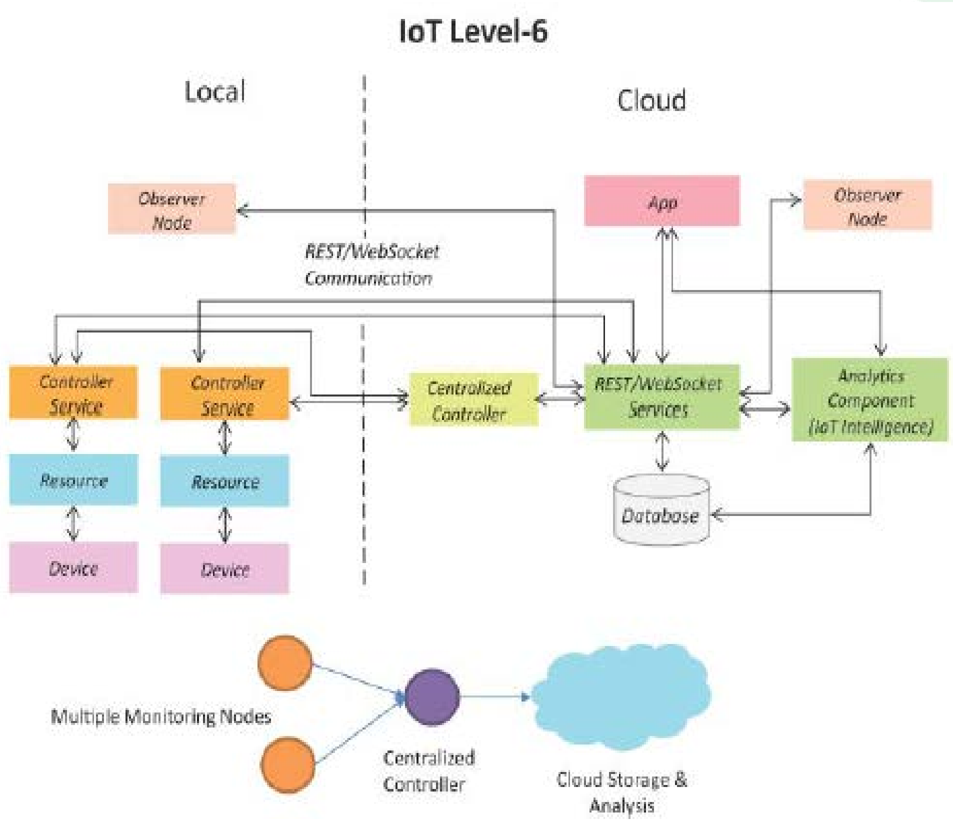


OntarioTech UNIVERSITY

Book website: http://www.internet-of-things-book.com

IoT Level-6

- A level-6 IoT system has multiple independent end nodes that perform sensing and/or actuation and send data to the cloud.
- Data is stored in the cloud and application is cloud-based.
- The analytics component analyzes the data and stores the results in the cloud database.
- The results are visualized with the cloud-based application.
- The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes.

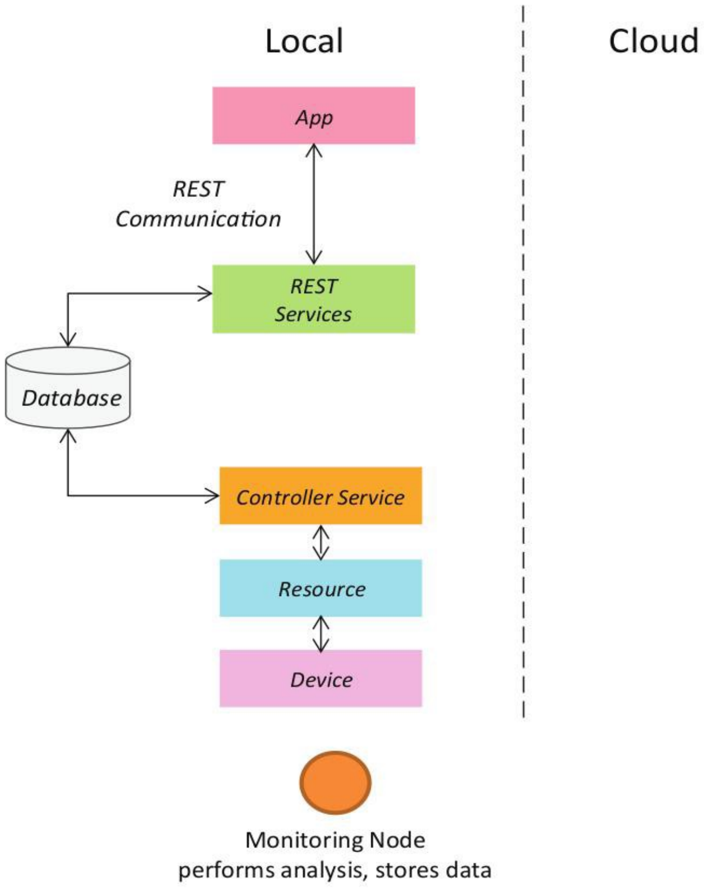


Book website: http://www.internet-of-things-book.com

OntarioTech UNIVERSITY

Step 6: IoT Level Specification

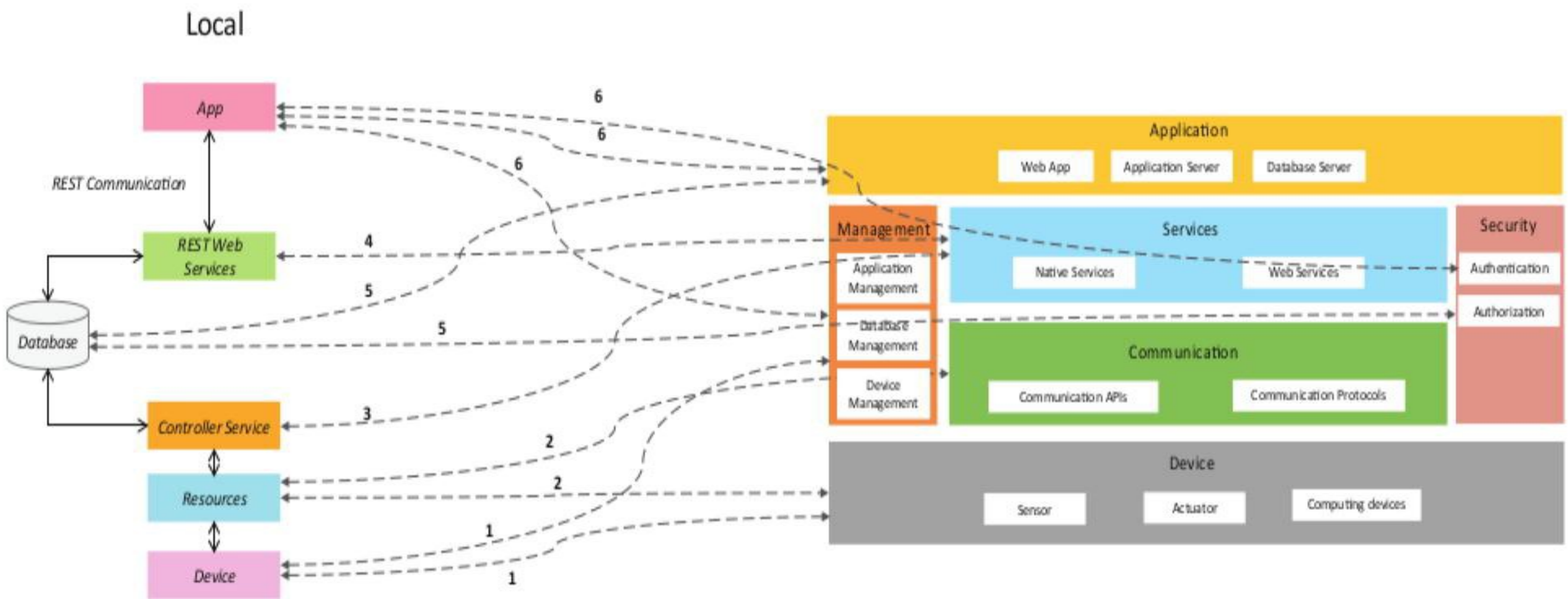
- In our case we would like to deploy this IoT application as a Level-1 deployment, i.e. local deployment.



OntarioTech UNIVERSITY

Step 7: Functional View Specification

- Defines the functions of the IoT application grouped into various Functional Groups (FG)



1. IoT device maps to the Device FG (sensors, actuators devices, computing devices) and the Management FG (device management)

2. Resources map to the Device FG (on-device resource) and Communication FG (communication APIs and protocols)

3. Controller service maps to the Services FG (native service). Web Services map to Services FG (web services)

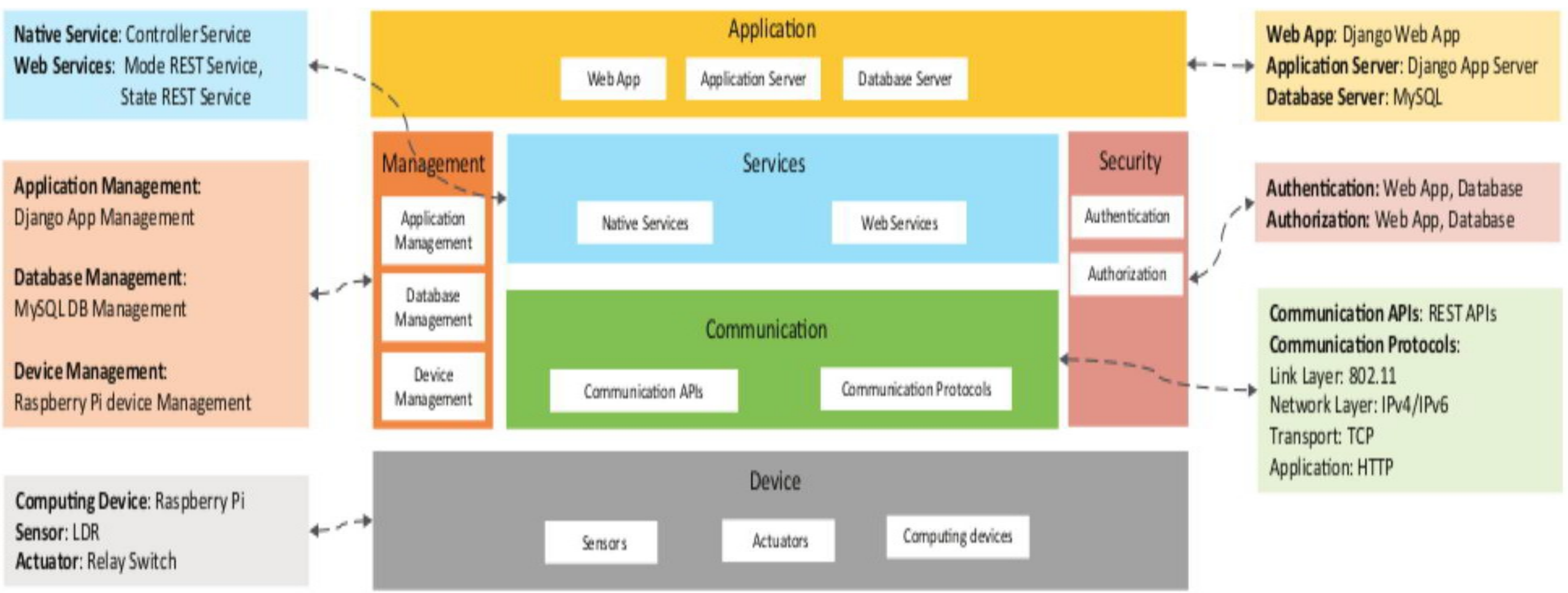
4. Web Services map to Services FG (web services)

5. Database maps to the Management FG (database management) and Security FG (database security)

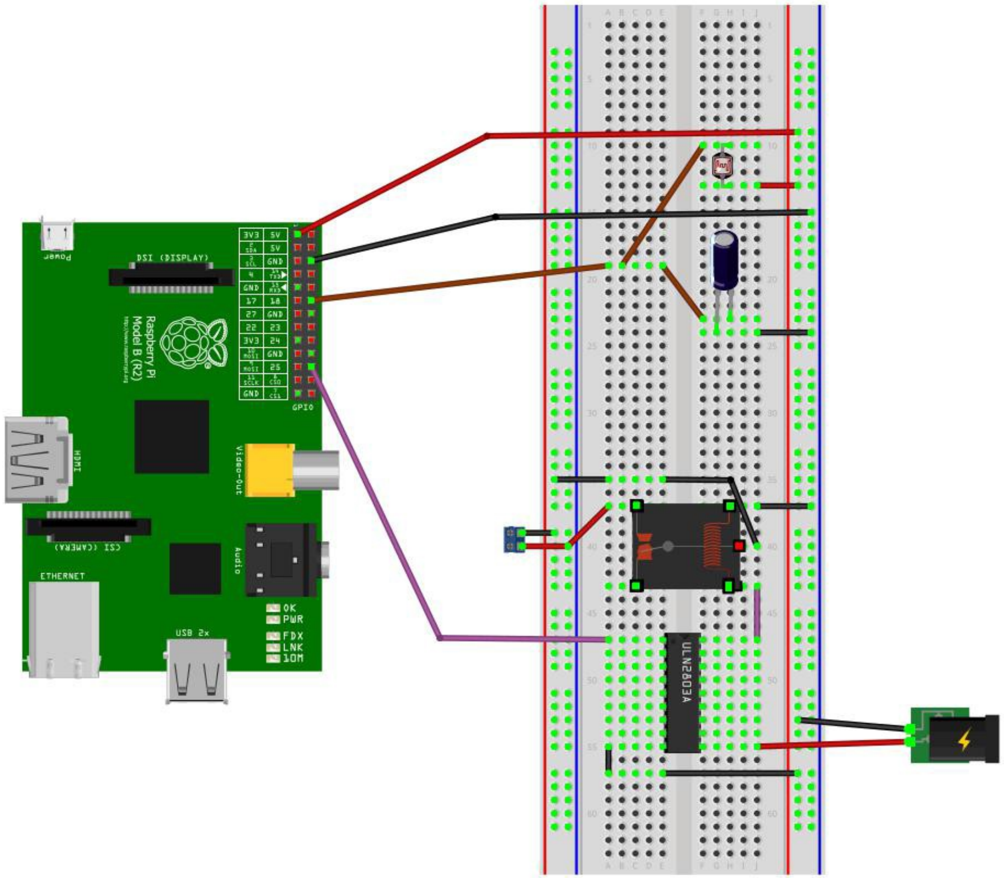
6. Application maps to the Application FG (web application, application and database servers), Management FG (app management) and Security FG (app security)

Step 8: Operational View Specification

- Various options pertaining to the IoT application are defined such as service hosting, storage, device, application hosting, etc.



Step 9: Device & Component Integration



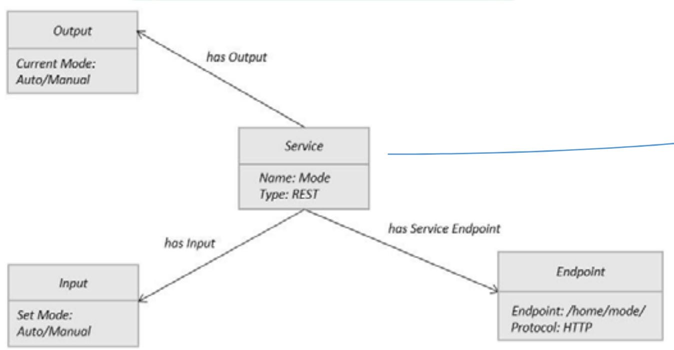
Step 10: Application Development

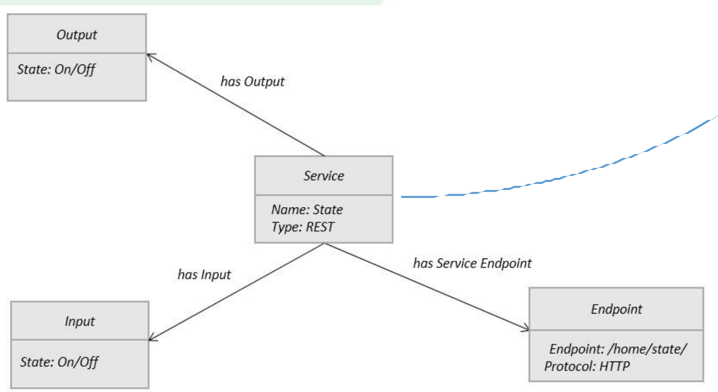
- Auto
  - Controls the light appliance automatically based on the lighting conditions in the room
- Light
  - When Auto mode is off, it is used for manually controlling the light appliance.
  - When Auto mode is on, it reflects the current state of the light appliance.

OntarioTech UNIVERSITY

Implementation: RESTful Web Services

REST services implemented with Django REST Framework





1. Map services to models. Model fields store the states (on/off, auto/manual)

```
# Models – models.py
from django.db import models

class Mode(models.Model):
        name = models.CharField(max_length=50)

class State(models.Model):
        name = models.CharField(max_length=50)
```

2. Write Model serializers. Serializers allow complex data (such as model instances) to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types.

```
# Serializers – serializers.py
from myapp.models import Mode, State
from rest_framework import serializers

class ModeSerializer(serializers.HyperlinkedModelSerializer):
        class Meta:
                model = Mode
                fields = ('url', 'name')

class StateSerializer(serializers.HyperlinkedModelSerializer):
        class Meta:
                model = State
                fields = ('url', 'name')
```

OntarioTech UNIVERSITY

Implementation: RESTful Web Services

```
# Models – models.py
from django.db import models

class Mode(models.Model):
        name = models.CharField(max_length=50)

class State(models.Model):
        name = models.CharField(max_length=50)
```

3. Write ViewSets for the Models which combine the logic for a set of related views in a single class.

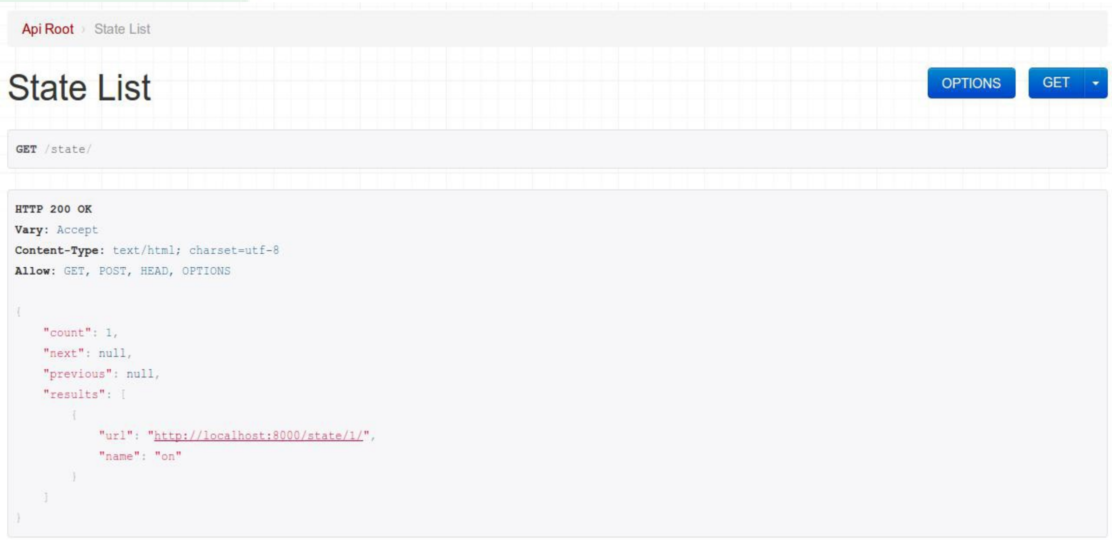
```
# Views – views.py
from myapp.models import Mode, State
from rest_framework import viewsets
from myapp.serializers import ModeSerializer, StateSerializer

class ModeViewSet(viewsets.ModelViewSet):
        queryset = Mode.objects.all()
        serializer_class = ModeSerializer

class StateViewSet(viewsets.ModelViewSet):
        queryset = State.objects.all()
        serializer_class = StateSerializer
```

4. Write URL patterns for the services.
Since ViewSets are used instead of views, we can automatically generate the URL conf by simply registering the viewsets with a router class.
Routers automatically determining how the URLs for an application should be mapped to the logic that deals with handling incoming requests.

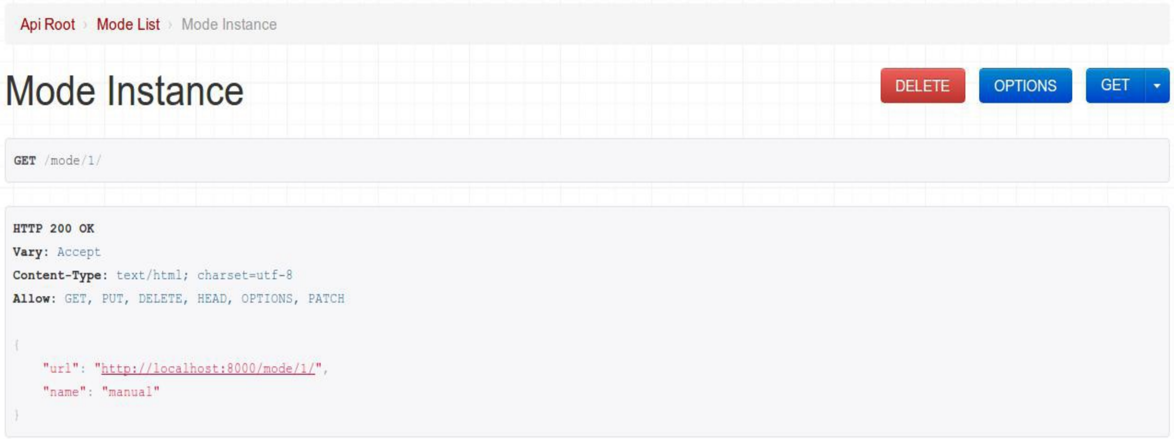
```
# URL Patterns – urls.py
from django.conf.urls import patterns, include, url
from django.contrib import admin
from rest_framework import routers
from myapp import views
admin.autodiscover()
router = routers.DefaultRouter()
router.register(r'mode', views.ModeViewSet)
router.register(r'state', views.StateViewSet)
urlpatterns = patterns('',
    url(r'^', include(router.urls)),
    url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework')),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^home/', 'myapp.views.home'),
)
```

Implementation: RESTful Web Services

Screenshot of browsable State REST API



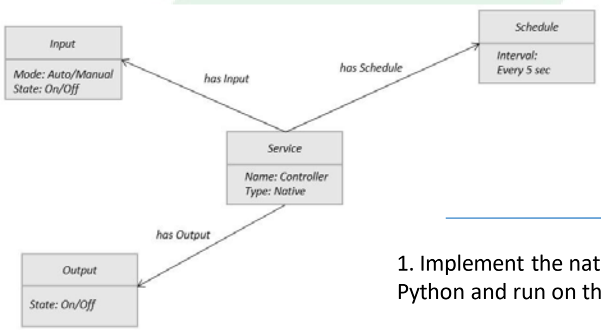
Screenshot of browsable Mode REST API



OntarioTech UNIVERSITY

Implementation: Controller Native Service

Native service deployed locally



1. Implement the native service in Python and run on the device

```
#Controller service
import RPi.GPIO as GPIO
import time
import sqlite3 as lite
import sys

con = lite.connect('database.sqlite')
cur = con.cursor()

GPIO.setmode(GPIO.BCM)
threshold = 1000
LDR_PIN = 18
LIGHT_PIN = 25

def readldr(PIN):
    reading=0
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, GPIO.LOW)
    time.sleep(0.1)
    GPIO.setup(PIN, GPIO.IN)
    while (GPIO.input(PIN)==GPIO.LOW):
        reading=reading+1
    return reading

def switchOnLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, GPIO.HIGH)

def switchOffLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, GPIO.LOW)
```

```
def runAutoMode():
    ldr_reading = readldr(LDR_PIN)
    if ldr_reading < threshold:
        switchOnLight(LIGHT_PIN)
        setCurrentState('on')
    else:
        switchOffLight(LIGHT_PIN)
        setCurrentState('off')

def runManualMode():
    state = getCurrentState()
    if state=='on':
        switchOnLight(LIGHT_PIN)
        setCurrentState('on')
    elif state=='off':
        switchOffLight(LIGHT_PIN)
        setCurrentState('off')

def getCurrentMode():
    cur.execute('SELECT * FROM myapp_mode')
    data = cur.fetchone()        #(1, u'auto')
    return data[1]

def getCurrentState():
    cur.execute('SELECT * FROM myapp_state')
    data = cur.fetchone()        #(1, u'on')
    return data[1]

def setCurrentState(val):
    query='UPDATE myapp_state set name="'+val+'"'
    cur.execute(query)

while True:
    currentMode=getCurrentMode()
    if currentMode=='auto':
        runAutoMode()
    elif currentMode=='manual':
        runManualMode()
    time.sleep(5)
```

Implementation: Application

1. Implement Django Application View

- # Views – views.py
- def home(request): out=‘’
  - if 'on' in request.POST:
    - values = {"name": "on"}
    - r=requests.put('http://127.0.0.1:8000/state/1/', data=values, auth=(‘username', ‘password')) result=r.text
    - output = json.loads(result) out=output['name']
  - if 'off' in request.POST:
    - values = {"name": "off"}
    - r=requests.put('http://127.0.0.1:8000/state/1/', data=values, auth=(‘username', ‘password')) result=r.text
    - output = json.loads(result) out=output['name']
  - if 'auto' in request.POST:
    - values = {"name": "auto"}
    - r=requests.put('http://127.0.0.1:8000/mode/1/', data=values, auth=(‘username', ‘password')) result=r.text
    - output = json.loads(result) out=output['name']
  - if 'manual' in request.POST:
    - values = {"name": "manual"}
    - r=requests.put('http://127.0.0.1:8000/mode/1/', data=values, auth=(‘username', ‘password')) result=r.text
    - output = json.loads(result) out=output['name']

```
r=requests.get('http://127.0.0.1:8000/mode/1/', auth=(‘username', ‘password'))
result=r.text
output = json.loads(result)
currentmode=output['name']
r=requests.get('http://127.0.0.1:8000/state/1/', auth=(‘username', ‘password'))
result=r.text
output = json.loads(result)
currentstate=output['name']
return render_to_response('lights.html',{'r':out, 'currentmode':currentmode, 'currentstate':currentstate},
context_instance=RequestContext(request))
```

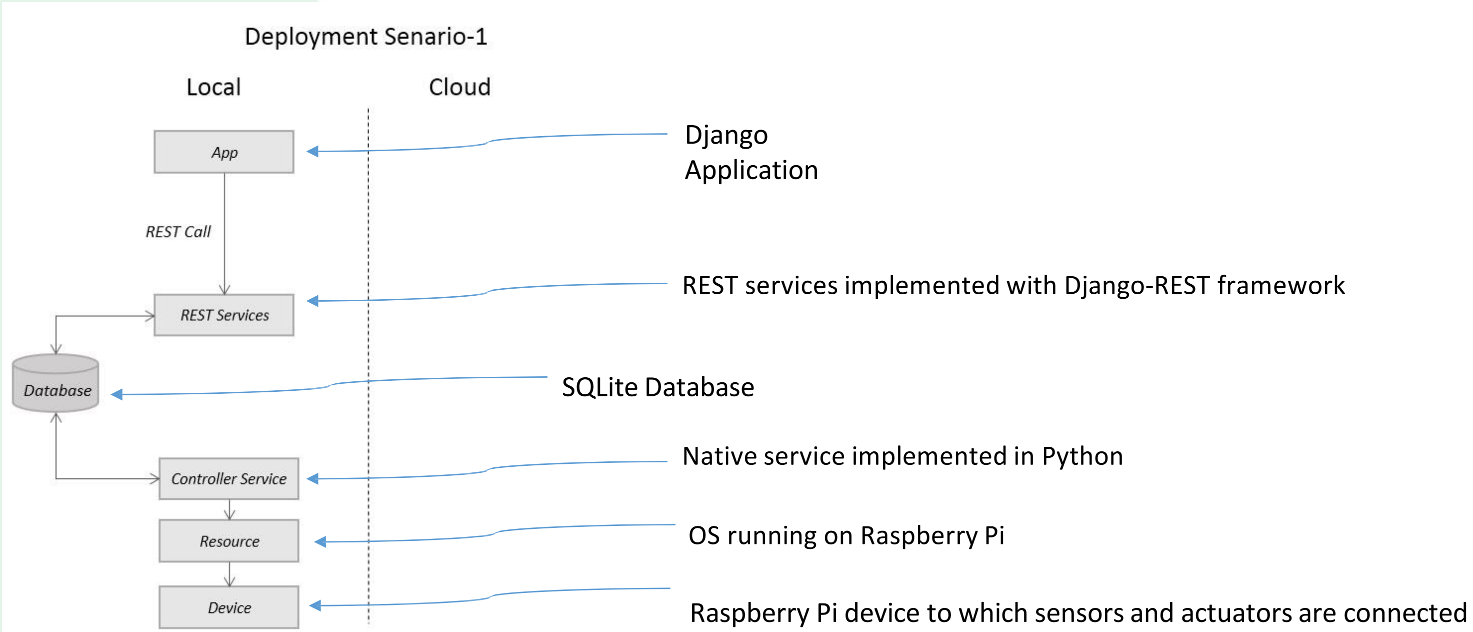
Implementation: Application

2. Implement Django Application Template

```
<div class="app-content-inner">
<fieldset>
<div class="field clearfix">
<label class="input-label icon-lamp" for="lamp-state">Auto</label>
<input id="lamp-state" class="input js-lamp-state hidden" type="checkbox">
{% if currentmode == 'auto' %}
<div class="js-lamp-state-toggle ui-toggle " data-toggle=".js-lamp-state">
{% else %}
<div class="js-lamp-state-toggle ui-toggle js-toggle-off" data-toggle=".js-lamp-state">
{% endif %}
<span class="ui-toggle-slide clearfix">
<form id="my_form11" action="" method="post">{% csrf_token %}
<input name="auto" value="auto" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-off">OFF</strong></a>
</form>
<strong class="ui-toggle-handle brushed-metal"></strong>
<form id="my_form13" action="" method="post">{% csrf_token %}
<input name="manual" value="manual" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-on">ON</strong></a>
</form></span>
</div></div>
<div class="field clearfix">
<label class="input-label icon-lamp" for="tv-state">Light</label>
<input id="tv-state" class="input js-tv-state hidden" type="checkbox">
{% if currentstate == 'on' %}
<div class="js-tv-state-toggle ui-toggle " data-toggle=".js-tv-state">
{% else %}
<div class="js-tv-state-toggle ui-toggle js-toggle-off" data-toggle=".js-tv-state">
{% endif %}
{% if currentmode == 'manual' %}
<span class="ui-toggle-slide clearfix">
<form id="my_form2" action="" method="post">{% csrf_token %}
<input name="on" value="on" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-off">OFF</strong></a>
</form>
<strong class="ui-toggle-handle brushed-metal"></strong>
<form id="my_form3" action="" method="post">{% csrf_token %}
<input name="off" value="off" type="hidden" />
<a href="#" onclick="$(this).closest('form').submit()"><strong class="ui-toggle-on">ON</strong></a>
</form>
</span>
{% endif %}
{% if currentmode == 'auto' %}
{% if currentstate == 'on' %}
<strong class="ui-toggle-on">    ON</strong>
{% else %}
<strong class="ui-toggle-on">    OFF</strong>
{% endif %}{% endif %}
</div>
</div>
</fieldset></div></div></div>
```

Finally - Integrate the System

- Setup the device
- Deploy and run the REST and Native services
- Deploy and run the Application
- Setup the database



Summary

- Presented a simple and generic design methodology for the design of IoT applications.
- Leverages the concept of services deployed in the Django framework