Decentralized IoT Requirements Document

1. Introduction

1.1 Overview

Design an orchestration manager for a decentralized IoT application. Instead of IoT devices relying on cloud for control and actions, they should be able to communicate with each other with the help of a manager on the local network to perform possible actions given a set of inputs/events. The device with spare computation power will be the manager.

1.2 Scope of the Product

The orchestration manager for a decentralized IoT system should be able to accomplish the following:

1. Receive state change notifications from the device that has gone through a state change,
2. Based on the rule set defined in the YAML configuration file (a detailed description of the scenarios. Sample YAML file will be added in the repository during hand off), instruct the devices which need to take action based on the notification received.
3. Register new devices that are added into the cluster.
4. Deregister the devices that are no longer the part of the cluster.

Note: For the purpose of demo, all the scenarios will be simulated on a set of Raspberry Pi Zeros.

1.3 Business Case for the Product

1. Decentralized IoT reduces latency in the actions performed when triggered by predefined events.
2. Reduced cost of on premise deployment of IoT devices and applications for the manufacturers as a server in the cloud no longer required.
3. Faster communication between devices which are on the same network.
4. Less data bandwidth consumption.

2. General Description

2.1 Product Perspective

1. This model for deploying IoT cluster is promising in terms of cost and latency.
2. IoT application developers will benefit from the finished product.
3. The users who will be interacting with the IoT with scenarios in their daily lives will experience a fast, responsive, quick and resilient system at work. The users will also experience less congestion on the egress on their network as the IoT devices no longer send data/information back and forth to the cloud.
4. The production companies will have to comply to a standard set of compatibility constraints if the generic framework that will be proposed has to work on majority of the devices. The manufacturers might have to move from their proprietary platforms to a more open-source/common platform to deploy Decentralized IoT framework.

2.2 Product Functions

- Enable communication between IoT devices without the need of a third server in the cloud. Having said that, the framework does appoint the device with spare compute power as the manager within the local network.
- Enable orchestration of the IoT cluster on the local network.

2.3 User Characteristics

End Users

One of the major consumers of the product would be IoT application developer. They would be using it so that their application can communicate with other devices. Their motivation for the same would be to integrate and orchestrate with the other devices in the network.
Obstacles in order to accomplish the same would be incompatibility of the platforms. The application developer should be able to integrate with the interfaces of the device manager.

Device Manufacturers

Many device manufacturers would also be using and referring to this framework to make the IoT device they manufacture compliant with the framework. Making the devices compliant with the tech stack of the framework should be recommended irrespective of whether the user opt in for

a Decentralized approach. The device manufacturers can define the scenarios using a YAML configuration file and the compiler that will be provided with the handoff repository.

2.4 General Constraints

The device manager is developed for Raspberry Pi Zero which is based on a linux (raspbian) environment. All the devices should support Flask and SQLite.

2.5 Assumptions and Dependencies

1. The devices would be Raspberry Pi Zero.
2. All the devices would have identical compute power.
3. All the devices should be connected on a single network.
4. Any device could be made manager arbitrarily.

3. Specific Requirements

3.1 User Requirements

Any IoT application developer can use the device manager to communicate with the other devices(without cloud). This would enable IoT devices to take actions based on communication/events from other devices within the network via the manager.

3.2 System Requirements

1. The orchestrator/manager needs to maintain current state (real time) of all the IoT devices.
2. The IoT device that ends up being the orchestrator needs to have a predefined rule set that defines actions to be performed on particular state changes.
3. The devices should support running a python file along with a flask server and SQLite database. Also , for the purpose of simulating the scenarios, the devices should have GPIO (General Purpose Input Output) configured.

3.3 Interface Requirements

Each device in the network could either be a device manager or a client.In order to accomplish that, it should be able to support these 2 APIs:

1. **receiveStateChange:** Each device should be able to receive a state change from the underlying hardware. This API decouples the process that is monitoring the state

change. Any new modules to detect the state change can be implemented whilst keeping the API intact.

2. **receiveAction :** Each device should be able to receive action from the manager. The receiveAction API receives a boolean which defined the toggle action for the respective device.

Each manager should be able to support these three APIs:

1. **registerDevice :** To register new devices with the manager.
2. **deregisterDevice:** To deregister a device that no longer wants to ba a part of the cluster.
3. **keepAlive:** A heartbeat that all the devices send to a manager at specific intervals along with the relevant metadata on the devices.

4. Appendices

Hand Off Process

1. Final hand off repository will have the following components:
    a. Fully working scenario.
    b. Sample YAML configuration file.
    c. Python script that will act like a compiler, The script will parse YAML file and create a folder with executables for the devices in the cluster.
2. Defining new scenarios:
    a. Addition of a new device will add a new device in the scenario.
3. The repository that will be handed over will have a README that defines all the components in the system along with an architecture diagram.

5. Glossary

1. **Orchestration:** Orchestration describes automated arrangement, coordination, and management of complex computer systems, and services.
2. **Centralized Application:** Applications/Services that are co-ordinated by a single point for control. (Cloud based server in case of IOT applications)

6. References

ODIN Project Seed - https://odin.cse.buffalo.edu/teaching/cse-662/2018fa/index.html