ESP32 Plant Monitoring Ontology Using Protégé.

Ian wallace
CIS 411 A
Database and Web Application.
Farahnaz Golroo

Month / Year

This project models an ESP32 based plant monitoring system inside Protégé.
An ESP32 board reads temperature, humidity, and soil moisture and sends data to cloud services.

You gain a structured view of hardware, data flow, and plant conditions.

Work in this course focuses on database ideas and web application concepts.
An ontology links those ideas to a real IoT style project with sensors and dashboards.

My project connects class topics to ESP32 hardware, Azure IoT Central, Google Sheets, and Power BI.

Project Objectives and Motivation

Main objective is to describe the ESP32 monitoring system in a clear knowledge model.

The model highlights devices, sensors, measurements, plants, cloud services, and user views.

Your motivation comes from senior design work on plant health tracking and cloud logging.

I wanted a model that supports explanation for presentations and grading. I also wanted a structure that supports future changes and extra sensors or services.

The ontology gives a shared language for discussion between you and your instructor.

Problem Statement

The main problem addresses how to describe all parts of the ESP32 project without confusion. Hardware, network settings, cloud pipelines, and dashboards often feel scattered.

The ontology groups these parts into clear classes and relationships.

Overview of the Domain Chosen

Domain covers an Internet of Things style plant monitoring setup. An ESP32 board gathers readings from a DHT11 sensor and a soil moisture sensor near a plant.

Cloud services store and show data, while dashboards present plant conditions over time.

Overview of the Domain Chosen

Key entities in the domain are System, Device, Sensor, Plant, Environment, Measurement, and Cloud Service.

Together these classes outline both technical parts and user facing results.

Ontology Design Process

I began by listing important nouns from the project description and lab notes.

Those nouns became candidate classes and subclasses in Protégé.

Next, you grouped classes under broader parents such as System, Device, Sensor, Plant, and CloudService.

After class planning I added object properties to link devices, sensors, and measurements.

Then I defined data properties for numeric values, timestamps, identifiers, and messages.

I finished design steps by adding restrictions, individuals, and consistency checks with the Reasoner.

Class Hierarchy Overview

System includes IoTSystem and PlantMonitoringSystem as specialized variants.

Device includes Microcontroller and ESP32Board

Classes form a tree that separates hardware, software services, environments, and user outputs.
Insert image: screenshot of Protégé class hierarchy tree.

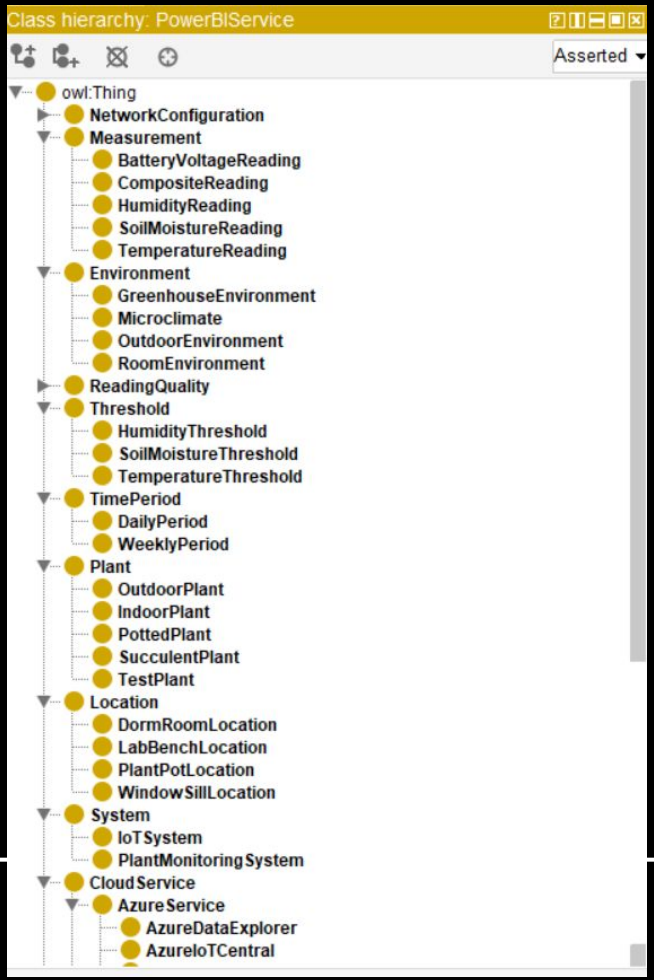
Class Hierarchy Overview

Sensor branch includes Temperature Sensor, Humidity Sensor, Soil Moisture Sensor

Measurement branch includes Temperature Reading, Humidity Reading, Soil Moisture Reading,

Cloud Service branch includes Azure IoT Hub, Azure IoT Central, Azure Data Explorer, Google Sheets Service, and PowerBIService.

Exampl
e

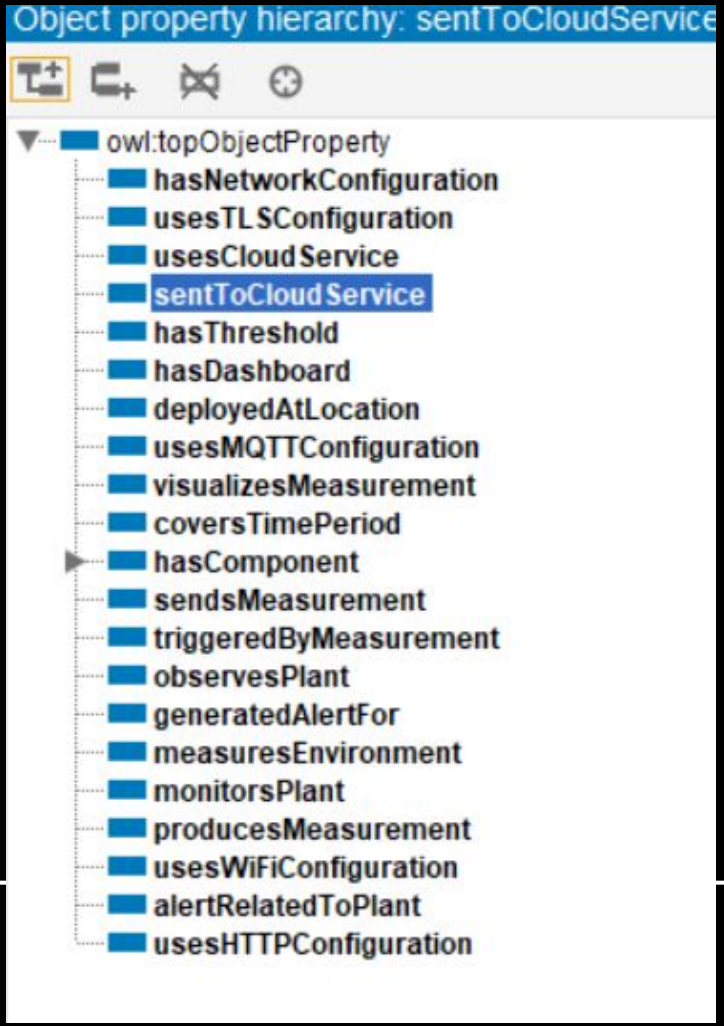

Class hierarchy: PowerBIService
Asserted
owl:Thing
NetworkConfiguration
Measurement
BatteryVoltageReading
CompositeReading
HumidityReading
SoilMoistureReading
TemperatureReading
Environment
GreenhouseEnvironment
Microclimate
OutdoorEnvironment
RoomEnvironment
ReadingQuality
Threshold
HumidityThreshold
SoilMoistureThreshold
TemperatureThreshold
TimePeriod
DailyPeriod
WeeklyPeriod
Plant
OutdoorPlant
IndoorPlant
PottedPlant
SucculentPlant
TestPlant
Location
DormRoomLocation
LabBenchLocation
PlantPotLocation
WindowSillLocation
System
IoTSystem
PlantMonitoringSystem
CloudService
AzureService
AzureDataExplorer
AzureIoTCentral

Property Design, Object Properties

I use object properties to show how classes connect in my ontology.

Key examples are hasComponent, hasSensor, monitorsPlant, deployedAtLocation, producesMeasurement, sendsMeasurement, sentToCloudService, usesCloudService, and hasNetworkConfiguration.

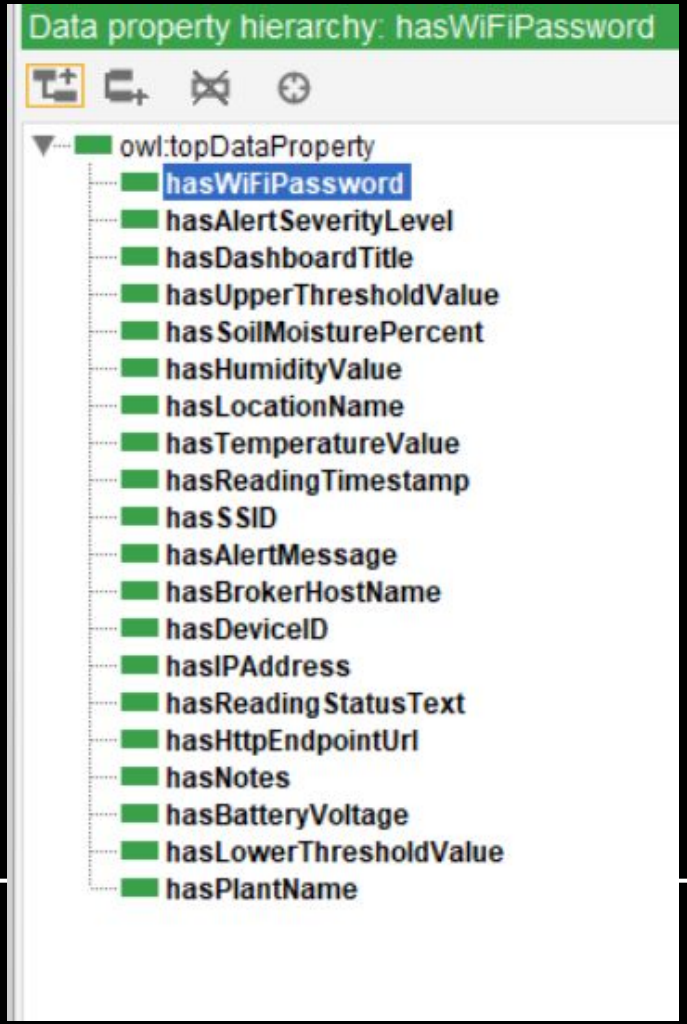
Exampl
e


Object property hierarchy: sentToCloudService
owl:topObjectProperty
hasNetworkConfiguration
usesTLSConfiguration
usesCloudService
sentToCloudService
hasThreshold
hasDashboard
deployedAtLocation
usesMQTTConfiguration
visualizesMeasurement
coversTimePeriod
hasComponent
sendsMeasurement
triggeredByMeasurement
observesPlant
generatedAlertFor
measuresEnvironment
monitorsPlant
producesMeasurement
usesWiFiConfiguration
alertRelatedToPlant
usesHTTPConfiguration

Property Design, Data Properties

I use data properties for strings, numbers, and dates that belong to devices, plants, and readings.

Exampl
e


Data property hierarchy: hasWiFiPassword
owl:topDataProperty
hasWiFiPassword
hasAlertSeverityLevel
hasDashboardTitle
hasUpperThresholdValue
hasSoilMoisturePercent
hasHumidityValue
hasLocationName
hasTemperatureValue
hasReadingTimestamp
hasSSID
hasAlertMessage
hasBrokerHostName
hasDeviceID
hasIPAddress
hasReadingStatusText
hasHttpEndpointUrl
hasNotes
hasBatteryVoltage
hasLowerThresholdValue
hasPlantName

Individuals, Instance Examples

I add individuals to show one real version of my ESP32 plant monitoring setup.

Example individuals are esp32_Prototype1, dht11_Sensor1, soilSensor1, plant_Aloe01. etc

More Individuals and Readings

I also create individual readings such as TempReading_2025_12_04_16_00 and SoilReading_2025_12_04_16_00 with numeric values and timestamps.

Some alert individuals link to these readings and to threshold objects for soil moisture and temperature.

Relationships among Classes

I connect ESP32Board to dht11_Sensor1 and soilSensor1 with hasSensor, and to plant_Aloe01 and dormRoomEnv1 with monitorsPlant and deployedAtLocation.

Sensors use producesMeasurement to point to Measurement individuals, and those measurements use sentToCloudService to point to AzureIoTCentral and GoogleSheetsService.

Use of Reasoner

I use the HermiT Reasoner in Protégé to check that classes and properties stay consistent.

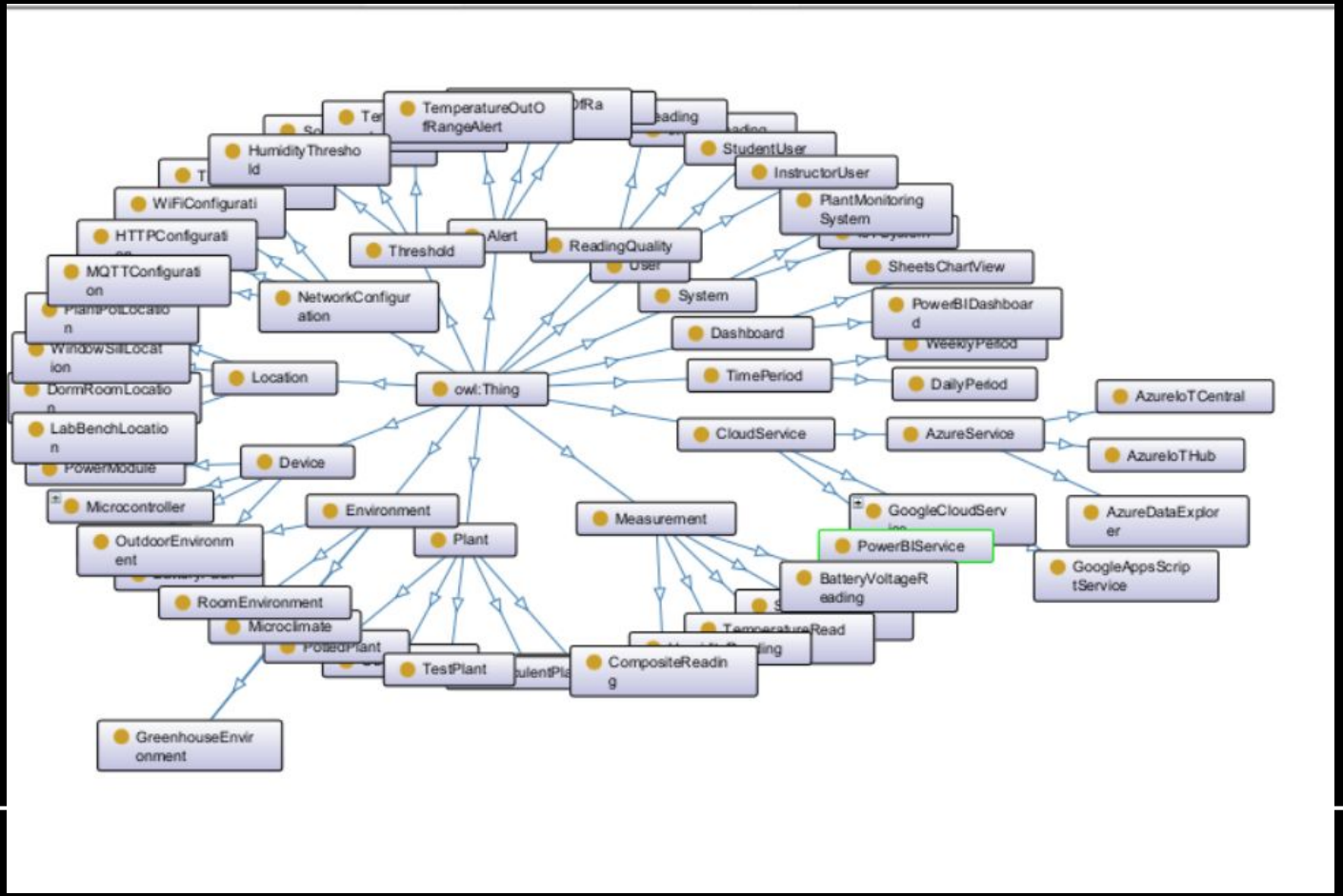
Insert image of the Reasoner window showing a consistent ontology status here.

Visualization with OntoGraf

I use OntoGraf to view classes and properties as a network around my ESP32 based plant monitoring system.

Insert image of an OntoGraf graph centered on ESP32Board, sensors, measurements, and cloud services here.

Exampl
e


TemperatureOutOfRangeAlert
HumidityThreshold
StudentUser
InstructorUser
WiFiConfigurati
PlantMonitoring System
HTTPConfigurati
Threshold
Alert
ReadingQuality
MQTTConfigurati on
SheetsChartView
NetworkConfigur ation
System
PowerBIDashboar d
Dashboard
WeeklyPeriod
Location
owl:Thing
TimePeriod
DailyPeriod
AzureIoTCentral
CloudService
AzureService
AzureIoTHub
Device
Microcontroller
Environment
Measurement
GoogleCloudServ
AzureDataExplor er
Plant
PowerBIService
OutdoorEnvironm ent
BatteryVoltageR eading
GoogleAppsScrip tService
RoomEnvironment
Microclimate
TestPlant
CompositeReadin g
GreenhouseEnvir onment

Challenges Faced and Solutions

I faced challenges choosing class boundaries and placement, especially for Environment and Location.

I fixed early Reasoner errors by adjusting domains, ranges, and restrictions while I tested the model.

Comparison to Real World Applications

My ontology lines up with common ESP32 plant monitoring projects in homes, labs, and greenhouses that use sensors and cloud logging.

Many real projects use ESP32 boards, DHT style sensors, soil probes, and cloud dashboards similar to Azure IoT, spreadsheets, and BI tools.

Future Improvements

I plan to add rule based alert logic that responds when readings cross threshold values.

I also plan to connect the ontology to live ESP32 data and expand plant, care, security, and cloud related classes.

Conclusion and Summary

My ESP32 plant monitoring ontology for CIS 411 A organizes devices, sensors, measurements, plants, cloud services, alerts, and dashboards into one clear structure.

Reasoner checks and OntoGraf views help me explain this structure during presentations and reports.

Additional Visuals

My ESP32 plant monitoring ontology for CIS 411 A organizes devices, sensors, measurements, plants, cloud services, alerts, and dashboards into one clear structure.

Reasoner checks and OntoGraf views help me explain this structure during presentations and reports.