ESP32 Plant Monitoring Ontology Using Protégé

Ian Wallace

Dr Farahnaz Golroo

CIS 411 A

Computer information system

December 4, 2025

Abstract

This report illustrates an ontology that has been developed in Protégé for an ESP32-based plant monitoring system. The domain involves a system wherein an ESP32 board reads temperature, humidity, and soil moisture from sensors and sends data to Azure IoT Central, Google Sheets, and Power BI dashboards. The purpose is to provide a clear model of major concepts and relationships. It contains more than fifty classes and subclasses, over forty object and data properties, and example individuals based on a working prototype. Reasoner checks and OntoGraf views confirm logical consistency and help to present the design.

Introduction

Many IoT projects combine hardware, networking, cloud platforms, and dashboards. Except for structure, the descriptions of those pieces often become muddled. For this assignment, focus on an ESP32 plant monitoring project that records temperature, humidity, and soil moisture and sends readings to cloud targets. The ontology models this project at a concept level rather than source code.

One question is addressed in this report: how to represent this ESP32-based monitoring project in a way that keeps devices, measures, and services clear. Ontology modelling defines classes such as Device, Sensor, Measurement, Plant, CloudService and Dashboard and links among them. Such a structure supports explanations of the project itself and supports further changes.

Literature Review

On the hardware side, the ontology was informed by documentation from ESP32 boards, DHT11 sensors, and soil moisture sensors. Tutorials and reference guides also explained device registration and telemetry flow using Azure IoT Central and MQTT, respectively, as well as common cloud patterns. Finally, introductory material on OWL and Protégé showed typical use of class hierarchies and properties in ontology design.

The simple IoT and sensor ontologies from the online examples provided ideas on how devices are separated from measurements and how to model locations, users, and alerts. These sources shaped the final choice of classes and properties and helped to align the ontology with real projects.

Methodology

Work on the ontology followed a fixed sequence. First, I defined a class hierarchy that covers the main parts of the ESP32 plant monitoring project. Top-level classes include System, Device, Sensor, Plant, Environment, Measurement, CloudService, NetworkConfiguration, Location, User, Alert, Dashboard, Threshold, ReadingQuality, and TimePeriod. Subclasses refine these areas, for instance, ESP32Board under Microcontroller, IndoorPlant under Plant, and AzureIoTCentral under CloudService.

Then I included the object properties to show the relationships. The various examples include: hasComponent between System and Device, hasSensor between Device and Sensor, monitorsPlant between Device and Plant, producesMeasurement between Sensor and Measurement, sentToCloudService between Measurement and CloudService, and hasDashboard between System and Dashboard. Other properties also link devices to network settings and dashboards to time periods, alerts, and thresholds.

After the relationships, I defined the data properties for literal values: hasDeviceID for hardware identifiers, hasPlantName and hasLocationName for labels, hasTemperatureValue, hasHumidityValue, and hasSoilMoisturePercent for sensor readings, and hasReadingTimestamp for date and time; hasAlertMessage and hasAlertSeverityLevel for alert text. Domains and ranges for each property further restrict usage to appropriate classes.

Next, I introduced simple restrictions: for instance, every PlantMonitoringSystem must have at least one Device through hasComponent,t and every Sensor produces only Measurement instances. Finally, I ran Reasoner to test consistency and used OntoGraf to display and refine the structure.

Results and Visualisation

The complete ontology meets the requirements of Milestone 2 for class count and property count and also corresponds to the physical ESP32 prototype. Individuals are represented by esp32_Prototype1, dht11_Sensor1, soilSensor1, plant_Aloe01, dormRoomEnv1, azureIoTCentralInstance, googleSheetsLogger, and powerBI_Dashboard1, which represent real devices, plants, locations, or services. Other individuals correspond to temperature, humidity, or soil moisture readings, whose timestamp and value pairs are also available.

Object properties link these individuals in ways that reflect the real project. The ESP32 board has sensor links to each sensor. The board monitorsPlant for plant_Aloe01 and deployedAtLocation for dormRoomEnv1. Each measurement instance producesMeasurement from a sensor, sentToCloudService to AzureIoTCentral or GoogleSheetsService, and visualizesMeasurement from a dashboard. Data properties attach numeric readings, timestamps, names, and alert text.

OntoGraf views present this structure through node and edge diagrams. The first view focuses on the class tree for System, Device, Sensor, Measurement, Plant, CloudService, and Dashboard. Another focuses on how measurements flow from devices to cloud services and dashboards and how thresholds and alerts relate to those readings. These images support explanation of the design during presentations and reports.

Discussion

This ontology represents the plant monitoring domain in a direct way, reflecting the objectives of CIS 411 A. The separation between hardware, measurements, cloud services, and user views remains clear so that readers may recognize where each part of the project fits in. The model also allows room for extension, as new sensors, plants, or dashboards fit naturally as subclasses of existing classes.

Development was not without its challenges. Choosing class boundaries took several revisions-for example, deciding how far to split environment and location concepts and how many alert types to retain. Setting domains and ranges resulted in some early Reasoner errors, which led to some property and restriction adjustments. Some classes, like Microclimate or WeeklyPeriod, are present only in the hierarchy and do not have individuals yet; however, they support a more comprehensive view of the domain. The alert logic is still modeled at the concept level, and numeric rules for thresholds have not been encoded.

Conclusion

This report covers an ontology in Protégé for an ESP32-based plant monitoring system, which was connected to work from CIS 411 A. The model describes how an ESP32 board with temperature, humidity, and soil moisture sensors sends readings to Azure IoT Central and Google Sheets with Power BI dashboards for visualization. It includes more than fifty classes, more than forty object and data properties, and a set of individuals mirroring the real prototype. It also supports confidence in the structure through reasoner checks and OntoGraf diagrams that will help you explain the project to others.

Other directions for future work might include extending this ontology to include rule-based logic for creating alerts based on threshold values, directly connecting individuals to live data from the ESP32 so that new measurements appear automatically in the model, expanding the plant side of the model with additional species, care schedules, and growing environments, or refining security-related classes and describing configuration for TLS, SAS tokens, and Wi-Fi policies in more detail.

References

https://documentation.espressif.com/esp32_technical_reference_manual_en.pdf

https://learn.microsoft.com/en-us/azure/iot-hub/

https://protege.stanford.edu/