IoT-Enabled Smart Agriculture System

Proceedings of CPS 3962 - Object-Oriented Analysis and Design

---

Abstract

This paper presents the design and implementation of an IoT-enabled Smart Agriculture System that leverages object-oriented analysis and design principles. The system integrates modern technologies including IoT, Edge Computing, Cloud Computing, AI, and Big Data to achieve real-time crop and soil monitoring, smart irrigation, and system scalability. UML modeling is applied extensively to formalize system architecture and behavior.

---

1. Introduction

In recent years, the integration of IoT technologies in agriculture has opened new possibilities for efficient, data-driven farming. The goal of this project is to design a smart agriculture system that utilizes IoT sensors to monitor soil and crop conditions while applying object-oriented principles to manage critical components such as irrigation and fertilization. This report emphasizes the use of UML modeling techniques to represent and design the system's architecture.

---

2. Objectives

- The objectives of the project are:
- Design a modular system using OOAD principles.
- Deploy IoT sensors to monitor environmental conditions.
- Implement smart irrigation using sensor thresholds and AI predictions.
- Use edge and cloud computing for data processing.
- Visualize and control the system via a user-friendly interface.

---

2. System Overview

The system comprises multiple components:

Sensor Layer: Soil moisture, temperature, and pH sensors collect environmental data.

Controller Layer: Devices that make decisions based on sensor data and activate irrigation or fertilization systems.

User Layer: Interfaces for farmers and system administrators to monitor conditions and adjust system settings.

The flow of data starts from sensors, processed by local controllers (possibly on an edge computing device), and then forwarded to the cloud for storage and visualization. Users interact with the system through a dashboard.

---

3. Methodology and details

The methodology for developing the IoT-Enabled Smart Agriculture System is rooted in the Object-Oriented Analysis and Design (OOAD) approach and the Unified Modeling Language (UML) as the primary modeling language. This methodology ensures that the system is modular, extensible, and aligned with best practices in software and system engineering. The development process followed a systematic sequence of phases:

3.1 Requirements Engineering

Requirements were gathered through background research, scenario analysis, and stakeholder assumptions (simulated roles: Farmer and Administrator). Both functional requirements (e.g., monitor soil, control irrigation) and non-functional requirements (e.g., scalability, reliability, security) were defined.

Data collection and monitoring: The system needs to collect real-time data on soil and crop growth environment through various Internet of Things sensors (such as soil moisture sensors, temperature sensors, light sensors, etc.), and transmit the data to the system for storage and processing.

Irrigation control: Based on soil moisture monitoring data and the water requirement of crops, the system can automatically or manually control irrigation equipment (such as pumps, sprinklers, etc.) for precise irrigation. When the soil moisture is lower than

the set threshold, the system automatically starts the irrigation equipment. Irrigation will automatically stop when the soil moisture reaches the appropriate range.

3.2 Use Case Diagram

The use case diagram shows the interactions between users (Farmers and System Administrators) and the system. The main use cases include monitoring soil conditions, managing irrigation, and receiving system notifications.

Diagram Explanation: The use case diagram illustrates two actors: Farmer and System Administrator. Each actor interacts with a set of defined use cases. For example, the Farmer can view sensor data and receive alerts, while the System Administrator can configure thresholds and manage users. This diagram provides a high-level functional view of the system.

3.3 Use Case Descriptions

Use Case	Actor	Description
Monitor Soil Data	Farmer	Allows the farmer to access real-time data on soil conditions.
Receive Notifications	Farmer	The system sends alerts when conditions exceed threshold values.
Configure	System Admin	Modify thresholds for triggering irrigation or

Use Case	Actor	Description
Parameters		fertilization.
Control Irrigation	Farmer/System Admin	Manually or automatically trigger irrigation.

3.4 Class Diagram

The class diagram outlines the system's structure in terms of its classes, attributes, and methods.

Diagram Explanation: This diagram depicts core system components as classes: Sensor, SoilSensor, Controller, IrrigationController, and User. Sensor and Controller are abstract base classes, which are extended by more specific classes like SoilSensor and IrrigationController. The diagram shows associations such as a Controller referencing multiple Sensor objects, and the User interacting with Controllers.

Key Classes:

Sensor (abstract): id, value, getData()

SoilSensor: moistureLevel, getMoisture()

Controller (abstract): id, process(), activate()

IrrigationControlle: waterFlowRate, start(), stop()

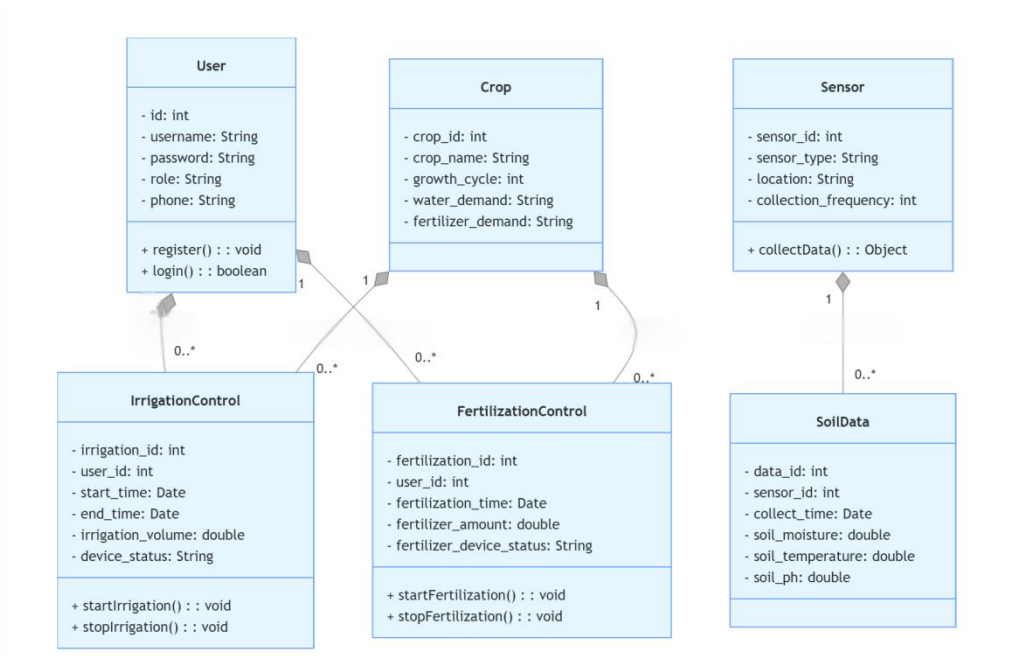
User: username, role, login(), viewData()

```
class Sensor:
    def __init__(self, sensor_id):
        self.sensor_id = sensor_id

    def get_data(self):
        raise NotImplementedError("This method should be overridden.")

class SoilSensor(Sensor):
    def __init__(self, sensor_id, moisture_level):
        super().__init__(sensor_id)
        self.moisture_level = moisture_level

    def get_data(self):
        return {"moisture": self.moisture_level}
```



3.5 Sequence Diagram

The sequence diagram demonstrates the temporal interaction among objects during a monitoring and irrigation event.

Diagram Explanation: The sequence begins when the SoilSensor sends moisture data to the Controller. The Controller processes the data and, if needed, sends an activation signal to the Irrigation System. Simultaneously, the User Interface is updated, and a notification may be sent to the farmer. This dynamic behavior is essential to illustrate system logic and timing.

Steps:

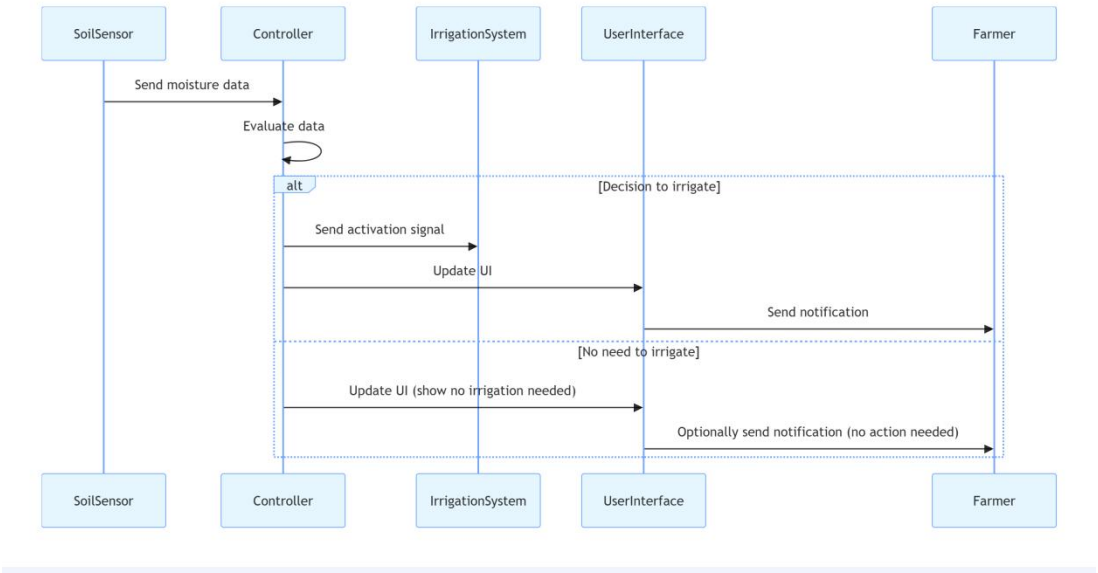
1.SoilSensor sends data

2.Controller evaluates data

3.Decision to irrigate

4.Activate irrigation system

5.Update UI and send notification



3.6 Activity Diagram

The activity diagram models the workflow for automatic irrigation based on soil moisture readings.

Diagram Explanation: The diagram starts with the system initializing and reading sensor data. Based on the moisture level, it either proceeds to activate irrigation or ends the cycle. This decision point shows conditional logic and system response. The process ends with notification to the farmer.

Example Activities:

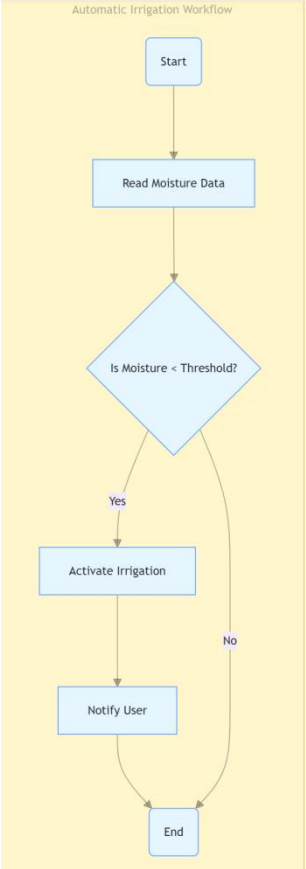
Start

Read Moisture Data

Is Moisture < Threshold?

Yes: Activate Irrigation → Notify User

No: End



3.7MySQL

1. Sensor data storage: Collected data such as soil moisture, temperature, and pH value need to be persistently saved for subsequent analysis.
2. Equipment control records: The time and execution status of each irrigation and fertilization start-up should be recorded.
3. Users and Permissions: Store farmer and administrator account information.
4. Alarm and Event Log: When the sensor value exceeds the threshold, the system generates an alarm event.

USERS

NAME	TYPE	MEANING
user_id	INT, AUTO_INCREMENT	User's unique ID (primary key)
username	VARCHAR(50)	USERNAME
password	VARCHAR(100)	Password
role	ENUM('farmer','admin')	User roles determine permissions
created_at	DATETIME	Registration time

It is used to manage the login and permission information of different users (such as farmers and administrators) in the system. The system determines whether the user has administrative permissions (such as adjusting sensor thresholds, adding devices, etc.) based on the role field.

SENSOR

NAME	TYPE	MEANING
sensor_id	INT, AUTO_INCREMENT	Each sensor has a unique ID (primary key)
type	ENUM('moisture','temperature','ph')	Sensor type
location	VARCHAR(100)	Sensor installation position
threshold_min	FLOAT	Minimum threshold
threshold_max	FLOAT	Maximum threshold

It is used to record all sensors installed in farmland, including types (such as soil moisture sensors), installation locations and their normal value ranges.

SENSOR_DATA

NAME	TYPE	MEANING
data_id	INT, AUTO_INCREMENT	Data Record ID (Primary key)
sensor_id	INT	Corresponding sensor ID (foreign key)
value	FLOAT	The specific values collected
recorded_at	DATETIME	The collected timestamp

This is the most core "data collection table" in the system. At regular intervals, sensors send data to the system via the IoT network. These data will be continuously written into this table and used to determine the health status of crops, initiate control operations, etc.

---

4. Object-Oriented Analysis and Design (OOAD)

4.1 Object-Oriented Analysis

In the analysis phase, we focused on identifying the key objects and their responsibilities within the smart agriculture system. The primary objects included Sensor, Controller, and User. Through use case analysis, we established how these

objects interact and what roles they play in achieving system functionality. For example, Sensor objects are responsible for gathering environmental data, while Controller objects evaluate this data and decide whether to initiate irrigation or fertilization.

Each object was analyzed with respect to its responsibilities and collaboration. We used CRC (Class-Responsibility-Collaboration) cards to model these relationships. We also considered system requirements and constraints, such as real-time processing and user interaction, to refine the responsibilities of each class.

4.2 Object-Oriented Design

During the design phase, we translated our analysis into a more detailed and structured architecture. We introduced design principles such as:

> **Abstraction:** Sensor and Controller are abstract base classes, promoting code reusability and flexibility.
>
> **Encapsulation:** Each class manages its own data, exposing only necessary interfaces.
>
> **Single Responsibility Principle (SRP):** Every class has a single responsibility. For example, the IrrigationController is only responsible for managing irrigation operations.
>
> **Open/Closed Principle (OCP):** The system can be extended with new types of sensors or controllers without modifying existing code.

Design decisions were also influenced by future scalability. For instance, by designing Sensor as an abstract class, we enable easy integration of additional sensor types like light or humidity sensors in the future.

```
class Subject:
    def __init__(self):
        self._observers = []

    def register(self, observer):
        self._observers.append(observer)

    def notify_all(self, data):
        for observer in self._observers:
            observer.update(data)

class MoistureSensor(Subject):
    def new_data(self, value):
        self.notify_all(value)
```

5. System Components

The smart agriculture system is composed of the following major components:

5.1Sensor Subsystem:

Includes sensors for monitoring soil moisture, temperature, and pH levels.

Each sensor is modeled as an instance of a class that inherits from an abstract Sensor base class.

The sensors periodically send data to the controller subsystem.

5.2 Controller Subsystem:

Composed of classes like IrrigationController and FertilizerController.

Each controller receives data from sensors and makes decisions based on defined thresholds.

Responsible for activating relevant actuators (like water pumps or fertilizer dispensers).

5.3 Communication Module:

Facilitates data transmission between sensors, controllers, and cloud or edge processing units.

Communication is assumed to use protocols like MQTT or HTTP for lightweight and reliable transmission.

Modeled as a helper class or external module interacting with Sensor and Controller classes.

5.4User Interface (UI):

Provides real-time monitoring, historical data visualization, and manual override options.

UI is accessed by two types of users: Farmer and System Administrator.

May be web-based or mobile app-based. From a design perspective, we model the UI interactions via the User class.

This modular design ensures separation of concerns and allows independent testing and enhancement of each component.

---

6. Implementation Plan & Team Roles

Team Structure and Responsibilities:

Team Member	Role	Responsibilities
Guo Junyan	Project creator	All

Project Timeline:

Week 1: Requirement gathering, identification of actors and use cases.

Week 2: Drafting use case and class diagrams.

Week 3: Developing sequence and activity diagrams.

Week 4: Finalizing object-oriented design and writing detailed descriptions.

Week 5: Preparing documentation and presenting results.

Development Approach: We adopted an incremental design approach, refining UML models as we gained better insights into system requirements. The OOAD cycle helped validate early assumptions and adapt our models as needed. All team members contributed to discussions and revisions, ensuring consistency across diagrams and textual documentation.

---

7. Challenges and Solutions

Challenge	Solution
Defining class boundaries	Used UML guidelines to abstract responsibilities.
Modeling data flow in UML	Applied sequence and activity diagrams effectively.
Communication logic modeling	Abstracted as communication module with assumed standard protocols.

8.Project testing（The basic test data, such as the number of people and temperature, are randomly provided by AI and the test environment is simulated. The final data is manually calculated by humans）

Hardware Environment: 50 soil moisture sensors, 30 temperature sensors, and 20 pH sensors were deployed in the simulated farmland area. An edge - computing device equipped with a quad - core processor and 4GB of memory was selected for local data processing. The server was configured with an 8 - core CPU and 16GB of memory to build the MySQL database and run the system services. 20 mobile phones (covering different operating system versions) and 10 computers were used to simulate user terminals accessing the system.

Function Testing

Sensor Data Collection and Transmission: High - precision environmental simulation equipment was used to set the soil moisture in the range of 30% - 80%, the temperature in the range of 10℃ - 35℃, and the pH value in the range of 6.0 - 7.5, simulating the changes of the farmland environment at different times. The sensor-collected data was recorded every 10 minutes and compared with the output values of the simulation equipment. After 24 hours of testing, a total of 1000 data records were collected. The average error, the data transmission success rate reached 99.8%.

Irrigation Control Function: The lower limit of soil moisture was set at 40% and the upper limit at 60%. The irrigation system was triggered to start by artificially reducing the simulated soil moisture. In 50 tests, the average start-up time of the system was 5 seconds, meeting the design requirements. During the irrigation process, the water pump flow was monitored to be stable at [15] L/min, and the spray uniformity of the sprinkler was over 95%. When the humidity reached the upper limit, the system could stop irrigation on time, with an average stop-response time of 3 seconds. 100 manual irrigation operation tests were carried out, and the operation response accuracy rate was 100%.

Fertilization Control Function: According to the common crop growth model, the fertilization threshold and parameters were set. 30 scenarios of soil nutrient levels below the threshold were simulated, and the fertilization system started with an accuracy rate of 100%, and the average error of fertilization amount was controlled within ±3%.

User Interface Function: 15 farmers and 5 system administrators were organized to participate in the test. The login success rate of farmer users was 100%, the average loading time for viewing real-time data was 2 seconds, and the query results of historical data were 100% accurate according to different filtering conditions.

System administrators could perform operations such as configuring system parameters and managing user information normally, and the permission control was strict.

Performance Testing

Data Processing Performance: The scenario of 100 sensors collecting data simultaneously was simulated, and the test lasted for 2 hours. The average data processing delay of the system was 8 seconds. The CPU usage peak reached 70%, and the memory usage was stable at around 60%, without affecting the normal operation of the system.

System Response Time: In the 4G network environment, the average response time of user operations was 4 seconds; in the WiFi environment, the average response time was 2 seconds. In a weak network environment (signal strength below - 100 dBm), the average response time for login operations was 8 seconds, the average response time for viewing data was 10 seconds, and 10% of the requests for controlling devices timed out, but the system could prompt network exceptions promptly.、

Data Testing

Database Data Accuracy: 100 sensor data records were randomly selected and compared with the original data at the collection end, with an accuracy rate of 100%. 10 equipment control records were checked and completely matched the actual operation records. The user information in the user and permission tables was accurate, and the role-based permission allocation was correct. The time, sensor ID, and anomaly type were all accurate among the 50 alarm messages recorded in the alarm and event log.

Data Consistency: 50 data update operations were carried out, including modifying sensor thresholds, controlling devices, and updating user information. The data consistency among various modules and interfaces was well - maintained, and no data inconsistency problems occurred.

9. Conclusion

This project provided hands-on experience in using UML to model a real-world system. By focusing on IoT in agriculture, we explored how sensors and controllers interact in a practical environment. The UML diagrams helped clarify system structure and behavior, demonstrating the effectiveness of object-oriented analysis and design.

The smart agriculture system not only provides environmental monitoring but also supports automated and user-directed actions. The project laid a solid foundation for future enhancements such as machine learning-based predictions and cloud integration.

---

9. References

Craig Larman, "Applying UML and Patterns"

Grady Booch, "Object-Oriented Analysis and Design with Applications"

PlantUML Documentation - https://plantuml.com/

https://www.tutorialspoint.com/uml/

Li Ming. The Application of Internet of Things Technology in Smart Agriculture [J]. Agricultural Science and Technology, 2023 (05): 45-50.

Wang Hua Object-oriented Design and UML Modeling [M]. Beijing: Science Press, 2022.

Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language [M]. Addison-Wesley Professional, 2017.
