UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Software Engineering for Autonomous Systems

Prof. Davide Di Ruscio

A.Y.: 2025/2026

Smart Olive Grove Manager (SOGM)

An Autonomic System for Precision Agriculture

Students: Marco Spada, id: 308887, e-mail: marco.spada@student.univaq.it

Github project link: https://github.com/ocraton/smart-olive-grove

1. Introduction: Autonomic Computing in Agriculture

Precision agriculture requires systems capable of adapting to rapidly changing environmental conditions without constant human intervention. The **Smart Olive Grove Manager (SOGM)** is an autonomic system designed to manage the critical operations of an olive grove: optimized irrigation, pest control, and frost protection.

The system is built upon the **MAPE-K** (Monitor-Analyze-Plan-Execute + Knowledge) reference model. Unlike traditional static automation, SOGM features a **Dynamic, Data-Driven Architecture** where control logic is injected at runtime via a centralized Configuration Service. This ensures high flexibility, decoupling, and resilience, adhering to the principles of Self-Adaptive Software Systems.

2. Adaptation Goals and Priorities

The system must fulfill conflicting objectives (e.g., treating pests vs. avoiding chemical drift due to wind). To manage this, we defined a strict hierarchy of goals based on **Numerical Priority**:

1. **Safety (Storm Protection)** - *Priority: 10 (Critical)*
    - **Goal:** Protect the physical infrastructure and prevent resource waste during extreme weather events.
    - **Constraint:** If wind speed exceeds safety limits, all water/chemical emitters must be shut down immediately.
2. **Crop Survival (Frost Protection)** - *Priority: 8 (High)*
    - **Goal:** Prevent irreversible damage to the trees due to freezing temperatures.
    - **Constraint:** Must react proactively to rapid temperature drops before $0°C$ is reached.
3. **Crop Health (Pest Control)** - *Priority: 5 (Medium)*
    - **Goal:** Mitigate pest infestations (e.g., Olive Fruit Fly) when population thresholds are exceeded.
    - **Constraint:** Chemical nebulization is subject to wind conditions to prevent pollution, unless the infestation persists for too long (Override Logic).
4. **Resource Optimization (Hydration)** - *Priority: 1 (Low)*
    - **Goal:** Maintain optimal soil humidity for production.
    - **Constraint:** Irrigation is the baseline operation but is interruptible by any higher-priority event.

3. System Architecture

The architecture follows the **External Approach** pattern, where the Autonomic Manager is distinct from the Managed Resource. The system is fully containerized using Docker to ensure portability and scalability.

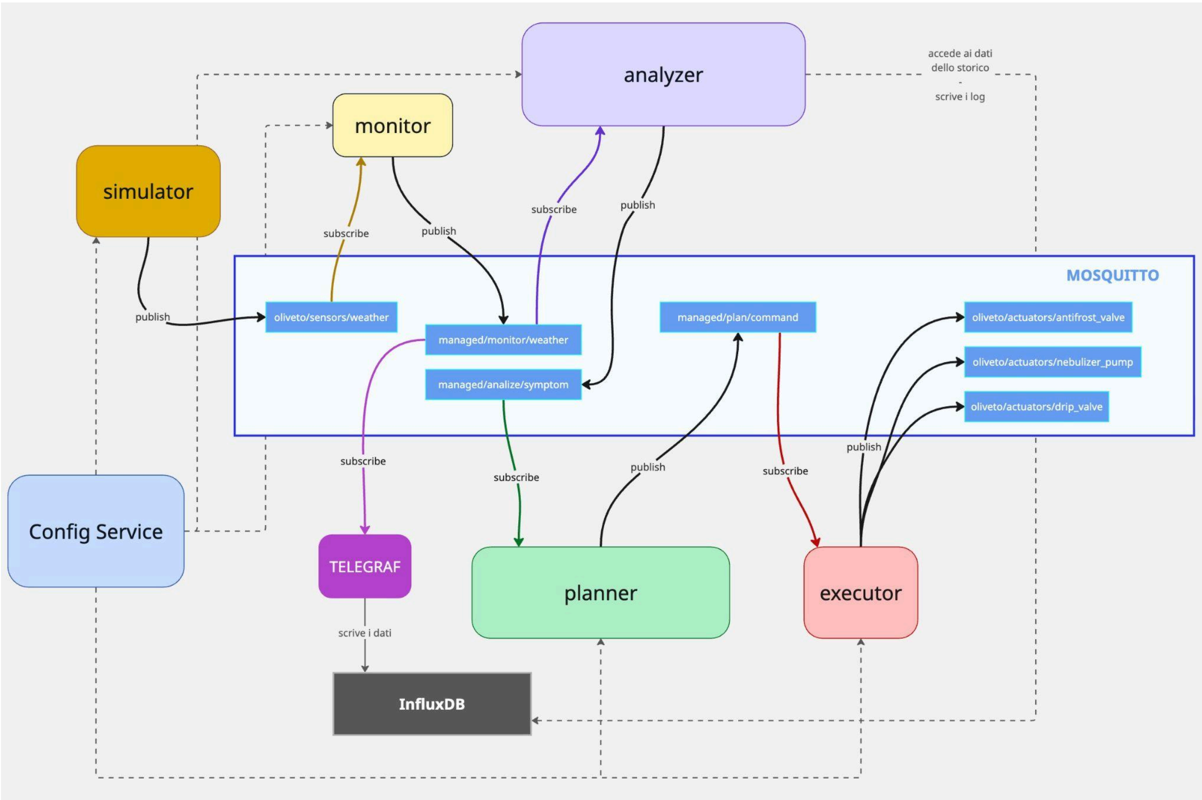

Fig 1: Runtime Architecture of the SOGM system. Solid lines represent reactive MQTT data flows; dashed lines represent initialization via config service and historical knowledge access.

3.1 Component Description

- **Managed Resource (Simulator):** A Node.js digital twin simulating environmental physics (Soil, Weather, Pests) and actuators. It is agnostic and learns its topology from the Config Service.
- **Config Service (Single Source of Truth):** An HTTP service that provides system topology and control policies to all microservices at boot.
- **Monitor:** Collects and standardizes sensor data via MQTT.
- **Analyzer:** The intelligence engine. Evaluates logical rules and performs stateful analysis (e.g., checking historical wind data on InfluxDB).
- **Planner:** The decision maker. Resolves conflicts using the Priority Hierarchy (e.g., $P_{\{Storm\}} > P_{\{Pest\}}$).
- **Executor:** Translates logical plans into physical actuator commands.

- **Knowledge Base:** Composed of **InfluxDB** (Time-series history) and **Mosquitto** (Current state bus).

4. Self-Adaptation Taxonomy

Following the taxonomy for self-adaptive systems, SOGM is classified as follows:

- **Time:**
  - *Reactive:* Immediate response to humidity thresholds or storm winds.
  - *Proactive:* Predictive activation of anti-frost systems based on temperature drop rates (dT/dt).
- **Reason:**
  - *Change in Context:* Adaptation driven by environmental variables (Weather, Pests).
- **Technique:**
  - *Parameter Adjustment:* Modifying actuator states (ON/OFF).
  - *Structure:* The system supports dynamic addition of sensors via the Config Service.
- **Control Approach:**
  - *Centralized with Heterarchical Logic:* A single Autonomic Manager executes multiple parallel control loops defined in the configuration.

5. Adaptation Logic and Scenarios

The system's intelligence is defined by **Control Loops**. Below is the formalization of the logic used in the Analyzer and Planner.

5.1 Scenario A: Hydration Maintenance (Reactive)

Goal: Maintain soil humidity (H) above a minimum threshold ($H_{\{min\}}$).

Logic Formulation:

$$Action_{irrigate}(t) = \begin{cases} ON & \text{if } H(t) < H_{min} \\ OFF & \text{if } H(t) \geq H_{target} \end{cases}$$

Implementation: A simple threshold check. In our tests, $H_{\{min\}} = 30\%$. The system employs a hysteresis cycle with $H_{\{target\}} = 35\%$ to prevents actuator flickering.

5.2 Scenario B: Smart Pest Control (Conflict Resolution)

Goal: Treat pests (P) only if safe.

Variables: $P_{\{count\}}$ (Trap count), $W_{\{speed\}}$ (Wind speed), $T_{\{delay\}}$(Time elapsed since first attempt).

Logic Formulation:

The activation function $f_{nebulizer}$ depends on a composite condition:

$$C_{infestation} = P_{count} > Threshold_{pest}$$

$$C_{safety} = W_{speed} < Threshold_{wind\_safe}$$

$$C_{override} = T_{delay} > 30 \text{ min}$$

$$Action_{nebulizer} = C_{infestation} \wedge (C_{safety} \vee C_{override})$$

Behavior: If $C_{infestation}$ is true but C_{safety} is false (high wind), the system queries InfluxDB. If the high wind persists longer than T_{delay}, $C_{override}$ becomes true, forcing the activation.

5.3 Scenario C: Frost Protection (Predictive)

Goal: Prevent freezing using proactive trend analysis.

Logic Formulation:

We define the temperature drop rate ΔT_{rate} over a time window Δt:

$$\Delta T_{rate} = \frac{T(t - \Delta t) - T(t)}{\Delta t}$$

The protection triggers if the temperature is critical OR if it is dropping too fast:

$$Action_{antifrost} = (T(t) \leq 0) \vee (\Delta T_{rate} > 2.0°C/h)$$

This predictive capability allows the system to act *before* the critical 0°C threshold is reached.

6. Visualization and Validation

The system's status is monitored via a **Grafana Dashboard**, connected to the InfluxDB Knowledge Base. The following sections demonstrate the validation of the three key scenarios, comparing the theoretical logic with the actual runtime behavior captured from the dashboard.

6.1 Validation of Scenario A: Hydration Maintenance

Goal: Reactive maintenance of soil humidity.

Evidence:

The graph below shows the system reaction when soil humidity drops below the threshold.

Fig 2: Hydration Loop response. The system detects humidity < 30% and activates the valve.

Logic Applied:

The activation follows the simple reactive rule defined in the configuration:

$$Action_{irrigate}(t) = \begin{cases} ON & \text{if } H(t) < 30\% \\ OFF & \text{if } H(t) \geq 35\% \end{cases}$$

6.2 Validation of Scenario B: Conflict Resolution & Temporal Override

Goal: Manage conflicting objectives (Crop Health vs. Environmental Safety) using a stateful priority system.

Logic Applied:

The Planner resolves the conflict using a composite condition that includes a time-based override:

$$Action_{nebulizer} = (P_{count} > 50) \wedge ((W_{speed} < 15) \vee (T_{delay} > 30\text{min}))$$

With $T_{\{max\}} = 30min$. This logic ensures that safety is prioritized initially, but critical crop preservation takes precedence if the adverse condition persists too long.

Evidence 1: Safety Constraint Enforcement (Wait State)

In the initial phase, a critical pest infestation ($P_{\{count\}} > 50$) coincides with unsafe wind conditions ($W_{\{speed\}} > 15\ km/h$).

As shown in **Fig 3a**, the **Nebulizer** remains **OFF** (Gray/Empty state). The system detects the conflict and enters a "Waiting State" to prevent chemical drift.

Fig 3a: Safety Lock. The system postpones the treatment due to high wind speeds, prioritizing safety over immediate pest control.

Evidence 2: Critical Override Activation (Stateful Behavior)

Fig 3b captures the system behavior when the unsafe wind condition persists beyond the defined threshold ($T_{\{delay\}} > T_{\{max\}}$) with $T_{\{max\}} = 30\, min$.

Although the wind speed (Blue Line) remains in the critical zone ($> 15\, km/h$), the **Nebulizer** switches to **ON** (Green State). The Analyzer queried the **InfluxDB Knowledge Base**, calculated the delay duration, and triggered the override to prevent irreversible crop damage.

Fig 3b: Temporal Override. The system forces the activation despite the unsafe wind constraint (visible particularly in the initial segment of the graph, where Wind > 15 km/h and Nebulizer is ON), as the infestation condition persisted longer than the allowed safety window (`EXT_INFLUX_DELAY`).

6.3 Validation of Scenario C: Frost Protection (Predictive)

Goal: Proactive infrastructure protection.

Evidence:

The dashboard captures a rapid drop in temperature. The system calculates the drop rate (dT/dt) and activates the **Anti-frost Emitter** *before* the freezing point is reached, demonstrating proactive behavior.

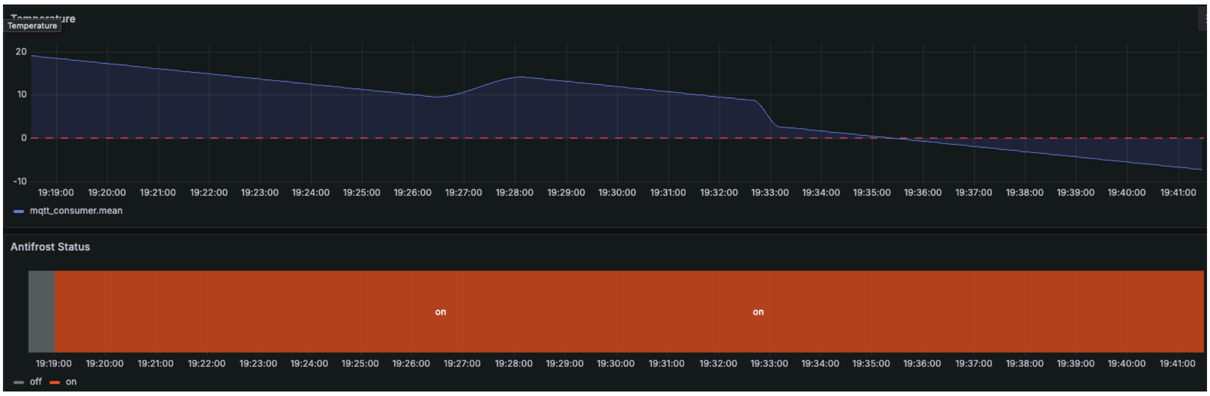

Fig 4: Predictive Frost Protection. Activation occurs based on the negative temperature trend.

Logic Applied:

The Analyzer evaluates the trend using a sliding window on the time-series database:

$$Action_{antifrost} = (T(t) \leq 0) \vee \left(\frac{\Delta T}{\Delta t} > 2.0°C/h \right)$$

7. Future Expansions

To further enhance SOGM, the following expansions are proposed:

1. **AI-Based Pest Detection:** Replace the simulated "Trap Count" with a Computer Vision module that analyzes images from real traps to count insects automatically.
2. **Energy Management:** Introduce a new control loop to manage the energy consumption of the pumps, prioritizing activation during hours of high solar production (if photovoltaic panels are available).
3. **Decentralized Executor:** Move the Executor logic directly to Edge devices (e.g., ESP32) to reduce latency and dependence on the central server.
4. **Automated Unit Testing Strategy:** Developing robust autonomic systems requires verifying logic without relying on full runtime simulations. A future expansion involves introducing a **Unit Testing Framework** (e.g., Jest or Mocha) to test the *Analyzer* logic in isolation.
    - **Mocking Methodology:** We can inject synthetic JSON messages (simulating Monitor data) directly into the Analyzer's evaluation functions.

- **Assertion:** The test asserts that for a specific input (e.g., *Wind=20, Pests=60, Time=0*), the Analyzer produces exactly the expected plan (e.g., *Action*=*NO_OP*), ensuring logical correctness and preventing regressions in the adaptation rules.