A Self-Adaptive Digital Twin Architecture to Automate Greenhouse Management

Riccardo Sieve*, Prasad Talasila†, Peter Gorm Larsen†, Einar Broch Johnsen*
* University of Oslo, Norway † Aarhus University, Denmark
riccasi@ifi.uio.no, prasad.talasila@ece.au.dk, pgl@ece.au.dk, einarj@ifi.uio.no

***Abstract*—Digital Twins are an emerging technology that is increasingly adopted for industrial applications across different domains. Digital Twins can leverage self-adaptive techniques to provide interoperability for cross-domain applications. This interoperability can come with challenges related to the different digital models applied, and with different scenarios in which distributed architectures are in place. In this setting, we propose a self-adaptive digital twin architecture to support reliable automated management for a category of physical twins. We apply declarative dynamic reclassification to capture the appropriate state, and adaptive control through strategies to improve the resilience of the system. The proposed approach is validated in both a simulated environment and a physical environment to compare the efficacy and reliability of the architecture.**

***Index Terms*—Digital Twins, Self-Adaptive Architectures, Automated Management.**

I. Introduction

Digital Twins (DTs) are virtual representations mirroring their physical counterparts; these so-called Physical Twins (PTs) range from the components of physical entities to social systems [1], [2]. DTs have been widely adopted in industrial applications at a mature stage of technological advancement [3], making these industrial applications suitable for the transition from Industry 4.0 to Industry 5.0 [4]. Through simulations, DTs can further offer predictive [5] and prescriptive [6] capabilities, expanding the different scenarios through which they can operate.

DTs have proven effective in empowering smart farming [7] and can help automate the management of greenhouses, for example by watering plants autonomously [8] or for greenhouse horticulture [9]. Furthermore, the health of plants may change frequently depending on several factors, both internal (e.g., the soil moisture and light exposure), and environmental (e.g., the overall humidity and temperature of the greenhouse). To handle these changes, the DT needs to capture the behaviour of the plants, and identify their different states [10].

This paper presents a self-adaptive DT architecture for automated greenhouse management. We show how the system can collect and process data to identify different states and adapt the digital model, using a declarative approach for both structural and behavioural adaptation. Together, these techniques are used to identify an automatic adaptation strategy to achieve overall system reliability, by ensuring that the DT system stays coherent with its physical counterpart.

The rest of the paper is structured as follows: in Sect. II, we describe related work on DTs, with a focus on their applications. In Sect. III we describe the use case and its physical setup, in Sect. IV we present the self-adaptive architecture proposed to automate the greenhouse management, and in Sect. VI we describe the experiment and their results. Finally, in Sect. VII we discuss future work and conclude.

II. Related Work

Several studies have focused on systems to improve automation control [11], [12], navigation [13] and inspection [14] for greenhouses. DTs have proved successful for fault diagnosis with different real-time conditions [15] and for predictive maintenance [16]. The integration of DTs in other areas, such as industrial water management, has also been explored [17], demonstrating the versatility of DTs in various industrial domains. However, the increase in the development of DTs also highlights an increase in the complexity of lifecycle management, requiring a more systematic and modular approach to the design, development, and deployment of DTs throughout the system's lifecycle [18]. DevOps practices have been proposed to address these challenges, by promoting composable and reusable components [19], and by coupling the architecture of the DT with semantic lifting [10], [20], [21].

To map the physical system to a runtime knowledge base (KB), we use SMOL [22]–[24], a small object-oriented programming language that, through semantic reflection into KBs [25], can adapt the runtime state [26] of entities defined in the underlying ontology of the KB. We can, therefore, map objects and properties of the ontology to classes and attributes of SMOL, respectively. Through SMOL, we can further adapt the runtime KB by adding new classes and attributes to represent new entities and properties that were not initially considered in the ontology, as well as by modifying existing ones, or removing them entirely [8]. This allows us to keep the digital model up-to-date with the physical system, as it evolves over time. Moreover, SMOL supports the definition of declarations to capture the behaviour of the entities in the ontology, allowing us to model not only the static structure of the twinned physical system in the KB but also the dynamic behaviour of the physical system in the SMOL program [27]. Using Functional Mock-up Interface (FMI), we can then simulate the physical system to identify the correct strategy to apply to the system to ensure reliability [28].

Work on DTs and ontologies have also been explored within data-driven simulation models. In particular, Bjørnskov et al. [29] proposed a development framework for DTs in which

the semantic model is used to capture the structure of the physical system, translated to be used within the simulator, and used to align the data processing layer. This allows for a seamless integration within the user interface layer. The work further relies on Long Short-Term Memory (LSTM) architectures to predict indoor temperature. In contrast, our work aims to explore the use of semantic reflection and declarative approach to identify the states of the PT, and to use them to adapt the components of the DT accordingly.

III. Use Case

This paper proposes a use case based on two physical greenhouses, namely *greenhouse-1* and *greenhouse-2* (shown in Fig. 1). The greenhouses are both used for research and experimentation with different plants, layouts and configurations. The greenhouses are equipped with sensors and actuators to monitor and control environmental conditions such as temperature, humidity, light intensity, and soil moisture.

The use cases have two levels of abstraction. The greenhouses provide a first layer of abstraction from low-level hardware to the DTs, i.e., sensor data and actuation commands. The second level of abstraction connects the runtime KB and the PTs themselves, mapping the information of the PT to the logical greenhouse components described in the physical configuration of Fig. 1. Since the two greenhouses differ in terms of physical configuration, we constructed a general ontology model that can be adapted to both physical greenhouses. The ontology model captures the essential components and relationships of a greenhouse, including plants, sensors, actuators, and environmental conditions. The DT model is implemented using OWL[1] (Web Ontology Language) and is designed to be flexible and extensible to accommodate different categories of logical greenhouse components. While it might not capture every specific detail of each greenhouse, it provides a common framework for representing and reasoning about the greenhouse environment. This modular approach is then used to create the runtime knowledge base by populating the ontology with data from a specific greenhouse configuration, leaving the missing components as *optional* in the ontology. The overall physical configuration introduced is shown in Figure 1. Compared to the configuration proposed in Greenhouse DT [8], the shelves have been renamed to sections, as the greenhouse can be scaled up to bigger configurations with sections instead of shelves, and (optional) nutrient sensors have been added to capture the current condition in *greenhouse-2*. To validate our approach, we define the following research questions:

- **RQ1** (*Correctness of strategy*): can we ensure that the selected strategy, for each plant, matches the expected one from its condition?
- **RQ2** (*Adaptation of strategy*): can we modify and adapt the selected strategy at runtime?
- **RQ3** (*Generic greenhouse construction*): how can we ensure that the system maintains consistency during construction to accommodate possibly missing elements?

[1]https://www.w3.org/OWL/

IV. Self-Adaptive Architecture

This section describes the methodology used to develop a self-adaptive architecture to optimise plant watering, shown in Figure 2. The architecture consists of several key components that work together to monitor plant health, adapt their states at runtime, and determine the optimal watering strategy.

a) Data Collector: The Data Collector is responsible for gathering real-time data from various sensors placed in the greenhouse. These sensors monitor, e.g., soil moisture, temperature, humidity, and light intensity. The collected data is stored in a centralised time-series database for further analysis.

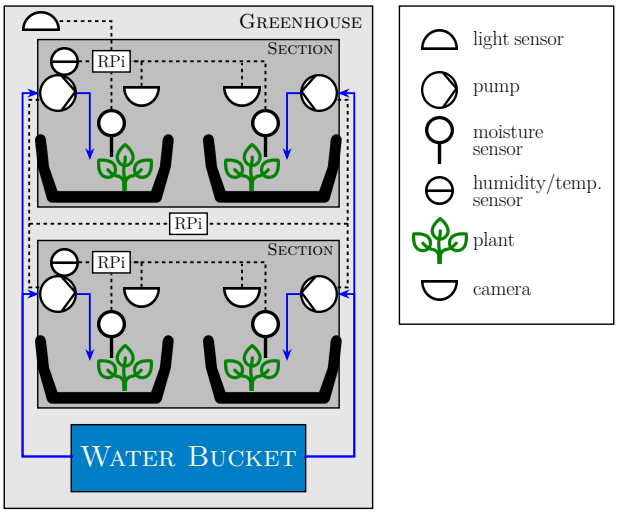
b) DT Orchestrator: The DT Orchestrator manages the DTs of the greenhouse. With SMOL, it creates and maintains a digital representation of the system's physical components and their interactions. Using data from the Data Collector, the DT Orchestrator ensures that each component accurately reflects the current state of its physical counterpart. It triggers dynamic adaptation of plants based on the collected sensor data and the adaptation conditions specified in the ontology.

The adaptation process involves reclassifying plants into different states (e.g., thirsty, moist, overwatered) based on the moisture sensor readings. This reclassification process allows the DT Orchestrator to make informed decisions about the appropriate watering strategy for each plant. Plant states are discretised based on the adaptation conditions defined in the ontology. While the current implementation focuses on the moisture sensor, the system can be extended with other sensors, such as temperature and light intensity, to refine the state classification and improve the accuracy of the watering strategies. The current choice of constraints stems from the limitation of the current physical setup, which only includes moisture sensors. Compared to predictive models, the declarative approach allows for better explainability of the system's decisions based on ontology reasoners.

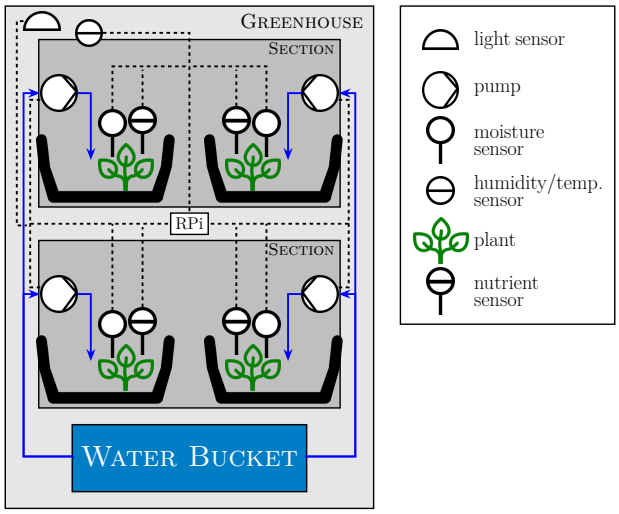
The orchestrator further implements the watering strategies by applying the appropriate watering strategy to each plant. The strategy is defined in a configuration file for better flexibility and customisation at runtime. The configuration is initially defined with three different strategies: conservative, aggressive, and default, with the possibility of extending it at runtime. Each strategy determines the amount of water to be delivered to plants according to their classified states.

c) Pump Actuator: The Pump Actuator is responsible for manipulating the water pumps, to deliver the specified amount of water to each plant. It executes watering commands, based on the time specified in the watering strategy provided by the orchestrator. The Pump Actuator ensures that the watering process is efficient and minimises water wastage.

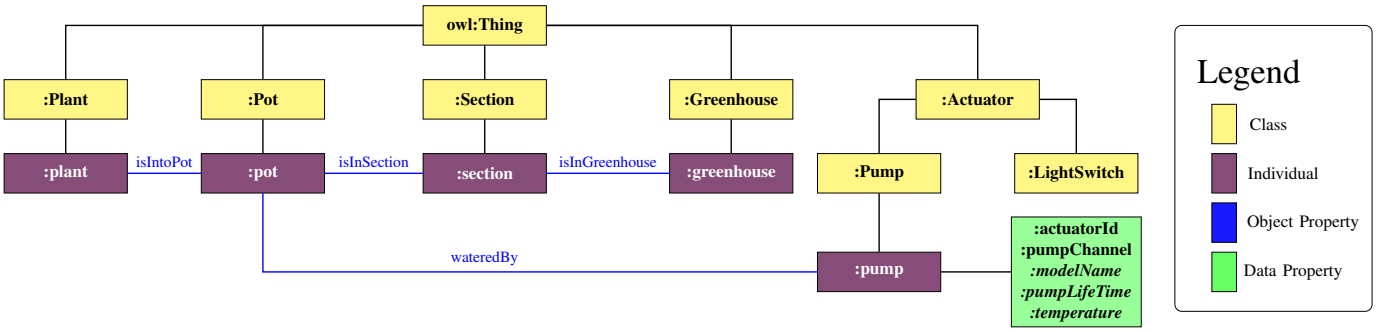
d) Message Broker: The Message Broker facilitates communication between the components of the Self-Adaptive Architecture, using a publish-subscribe model to transmit messages between the *Data Collector*, *DT Orchestrator*, and *Pump Actuator*. This decoupled communication not only allows for scalability and flexibility in the system, but also distribution as the orchestrator that contains the program logic need not reside on the same machine as the collector and actuator.



(a) Physical configuration of *greenhouse-1*.



(b) Physical configuration of *greenhouse-2*.



(c) Representation of a portion the Knowledge Base used to structure both physical configurations.

Fig. 1: Representation of the physical configurations of *greenhouse-1* (a), *greenhouse-2* (b), and the core Knowledge Base that integrates the different components (c). The elements in italic are optional, and can be missing in one of the two greenhouses.

V. Physical and Digital Twin Life Cycles

The operational control of a greenhouse PT by the DT can be seen as a state machine. Figure 3 shows how both external and internal events can trigger state changes in DT behaviour.

The DT starts with the *Load KB* state in which the KB, modelled using an ontology, is loaded into the DT which is not yet aware of the PT. The DT then enters the *Unbound* composite state in which the DT prepares for the possibility of linking with the PT. In this composite state, the DT goes through validation and update of the KB upon user input. There are two possible ways to enter this composite state. The first entry is the unconditional transition from Load KB to *Check Validity* in which the validity of the KB is checked. The DT cannot work with an invalid KB and therefore exits if the KB is found to be invalid. A valid KB transitions the DT into *Standby* state in which it can receive two user inputs, namely updates to the KB and a watering strategy. Requests to update the KB go through a sequence of states, namely *Check Consistency*, *Reject Update* and *Update KB*. The conditional check on the maintenance of KB consistency leads to an update of the KB; otherwise, the update is discarded. The *Standby* state can also respond to availability of the PT. The *PT active* event is sent by the PT when it is ready to be linked with this DT. This event comes with information about the PT configuration which typically consists of its network identity (IP address / hostname), physical setup details. The details shown in Figure 1 must be sent through this event.

The DT exits the *Unbound* composite state when it receives the *PT active* event and enters the *Bind* state, which checks the information received in the PT active event and verifies against the current KB. If the PT configuration is found to be compatible with the KB, the DT binds the appropriate model to the PT and sets up the duration of the control cycles. If the PT configuration is incompatible with the KB, the DT rejects the binding request and returns to *Standby* state. Otherwise, it proceeds to the *Entangled* composite state, and starts the monitor cycles. Each monitor cycle consists of sensing the PT, analysing the sensed data, and planning the next actions. If sensor data can be accessed, the DT proceeds to the watering action, following the defined strategy. If not, the DT falls back to the *Standby* state without performing any action. After performing the watering action, the DT returns

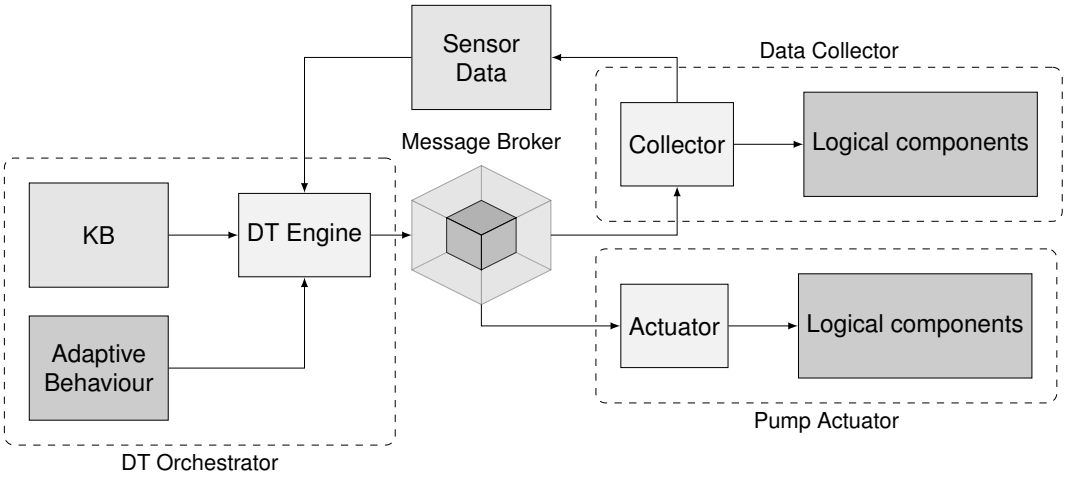


Fig. 2: Software architecture for the greenhouse implementation.

to *Monitor* state to wait for the next monitor cycle, or ends the entanglement by going to the final state with *stop DT* event.

VI. Validation

We present experiments to validate our approach and detail the different aspects of the self-adaptive architecture used.

A. Experiment Setup

To properly perform tests safely and address **RQ1**, we constructed a sandbox test environment. This setup incorporated an ontology reflecting the current status of *greenhouse-2*, along with additional components currently utilised in *greenhouse-2*. The aim was to capture a snapshot that could reflect as closely as possible the situation from both greenhouses. The requirement for a self-contained testing environment stems from the necessity to modify parameters and data in the time-series database without affecting the live system. This approach ensures the correctness of the results while maintaining isolation from production. We utilised values from previously recorded data and replayed them within the testing environment. This was done to maintain data consistency.

We use a Python script to automate the data writing, capture the resulting information and compare it with the expected ones from Table I. This approach allows us to automate the testing process and verify the correctness of the results.

Additionally, we have physical greenhouses that can be used for analysis and inspection of the collected data.

TABLE I: Watering strategies for the plants in the greenhouse.

	Durations			
name	**thirsty**	**moist**	**overwatered**	**unknown**
aggressive	8	3	0	3
conservative	3	1	0	1
default	5	2	0	2

B. Results

To answer **RQ1**, we checked the duration applied by the watering strategy with the values for the **default** strategy shown in Table I. To verify the correctness of the system with all the possible combinations of values, we set up a time-series database with values that capture different states for the plants. As shown in Table II, the system was able to classify the plants into the correct state based on the input moisture value and assigned the appropriate watering strategy.

To answer **RQ2** we use historical data collected from the *greenhouse-2*. We focused on the data collected for one plant and traced the watering strategy corresponding to the classified state. As shown in Figure 5, the state of the plant and, accordingly, the strategy applied, varies significantly depending on the water supplied. In the later part of the measured moisture trajectory, the plant was watered more often manually to keep the plant moist. In this sense, the state remained more stable in the moist area, leaving the plant state to be *moist*.

To answer **RQ3**, we followed a methodology of "generic by construction", where we apply consistency checks only on required elements. This is enforced in SMOL by allowing non-required attributes to be nullable. Thus, we can set the missing components to `null` and focusing only on the one in use. Since SMOL uses ontologies and semantic lifting, we can leverage the `OPTIONAL` keyword inside the SPARQL queries, as shown in Fig. 4. While the proposed approach supports

TABLE II: Test of the different plants with the corresponding duration given by the default strategy. In the table, expected and actual values represent the number of seconds for which the pump will be turned on to water the plant.

Plant ID	**State**	**Expected Value (s)**	**Actual Value (s)**
1	*overwatered*	0	0
2	*thirsty*	5	5
3	*moist*	2	2
4	*thirsty*	5	5

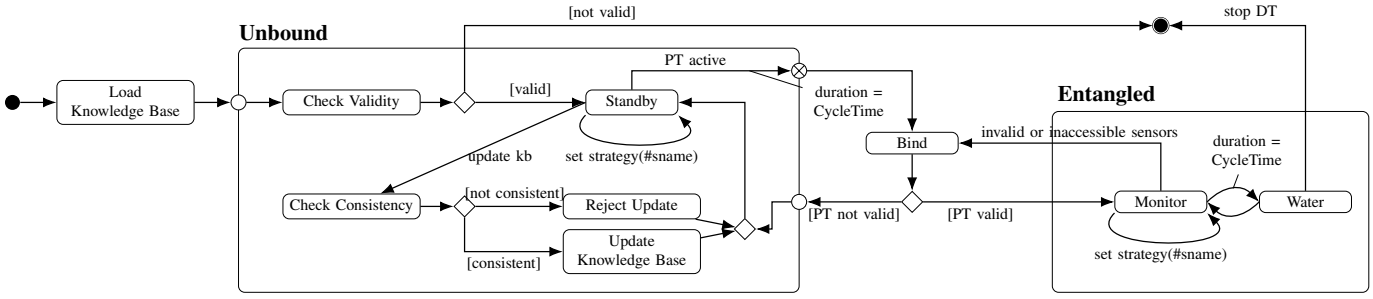


Fig. 3: State diagram illustrating the operations of a greenhouse digital twin.

SPARQL

```
SELECT ?pumpChannel ?actuatorId ?modelName
  ?pumpLifeTime ?temperature {
    ?pump a :Pump ;
          :pumpChannel ?pumpChannel ;
          :actuatorId ?actuatorId .
    OPTIONAL {
          ?pump :modelName ?modelName .
    }
    OPTIONAL {
        ?pump :pumpLifeTime ?pumpLifeTime .
    }
}
```

Fig. 4: Example of SPARQL query that leverages OPTIONAL elements when constructing Pump objects in SMOL.

flexibility and adaptability by accommodating different configurations in our experiments, it may not be able to capture all possible greenhouses; in fact, the structure of the underlying physical system must be expressible in the ontology. Otherwise, errors may potentially occur in the construction of the runtime KB for greenhouses with a deviating structure, and, in the worst case, in the execution of the system.

C. Discussion

The self-adaptive process improves the reliability and correctness of the model by providing formal guarantees using semantic reflection in SMOL. However, these guarantees come at a cost in performance: the construction and adaptation of the runtime KB impose an overhead on overall execution time. Overall, the overhead of the semantic lifting process is acceptable in the context of greenhouse management, where changes in plant states and environmental conditions typically occur over longer periods. However, it is important to consider the implications of scalability when using semantic lifting [23], as the overhead can increase with the size and complexity of the KB. As more entities and relationships are added to the model, the processing time may increase, potentially limiting the scalability of the system in larger and more complex greenhouses with a large number of plants and sensors. In this regard, we trigger the dynamic reclassification process only when the watering control is being processed. This limits the lifting process to the specific moment when it is actually required, improving the overall performance of the system at runtime. This poses, nevertheless, an interesting aspect in dealing with performance and correctness to obtain a system that can balance both aspects in the resulting system.

Furthermore, our approach is developed to be as generic as possible, leveraging SPARQL queries to ensure the correct execution of the system under different structural configurations. This further allows us to keep our system extensible and adaptable to different greenhouses. In this regard, the use of ontologies and semantic lifting provides a solid foundation to build upon, allowing for future extensions and adaptations as needed. Further work can be done in this direction to ensure the system can handle a wider variety of configurations and structures, extending its applicability and robustness.

VII. Conclusion and Future Work

In this paper, we proposed a self-adaptive architecture for DTs to automate greenhouse management. We applied a generic approach to make our system extensible to different categories of PTs, and tested our architecture with different environments to show the reliability of the methodology.

Our approach shows promising results, both in adapting to different greenhouses and in adapting states correctly. The separation of concerns between the generic digital twin architecture and the knowledge base, which captures the necessary domain knowledge about the plants and the structure of the greenhouse, simplifies the construction of the self-adaptive capabilities of the greenhouse, and enables the adaptability and extensibility of the proposed digital twin solution. So far, we have only tested automated management for the watering process. In future work, we plan to use nutrient sensors in conjunction with an additional cannister to manage nutrient delivery to plants. While creating optional components in the KB allows us to capture both greenhouses within the same structure, we want to explore a more granular approach that could allow us to ensure a proper generic structure that can be adjusted with completely different categories of PT.

Finally, we already mitigated the overhead introduced by the semantic lifting process for architectural and behavioural self-adaptation. Further improving the overall efficiency of the underlying implementation could expand the potential applications of our system towards real-time systems. We plan to address these aspects to further improve the scalability and range of components available for the DT.

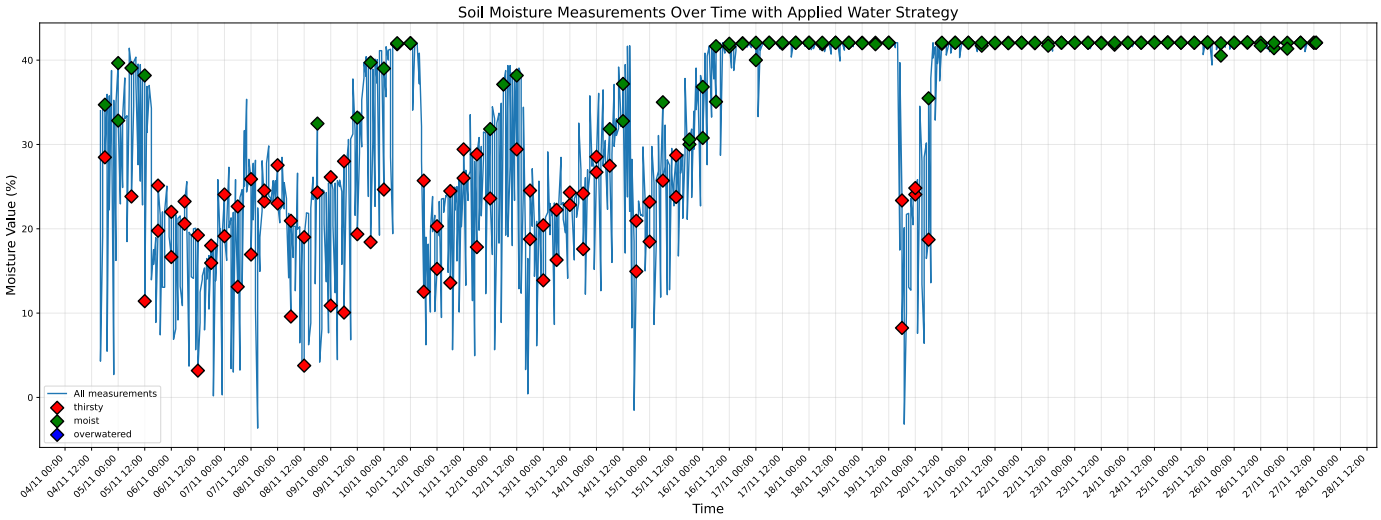


Fig. 5: Soil measurements for a plant in *greenhouse-2* with the respective water strategies.

REFERENCES

[1] National Academies of Sciences, Engineering, and Medicine (NASEM), *Foundational Research Gaps and Future Directions for Digital Twins*. The National Academies Press, 2024.

[2] J. Fitzgerald, C. Gomes, and P. G. Larsen, Eds., *The Engineering of Digital Twins*. Springer, 2024.

[3] J. Ren, R. Ahmad, D. Li, Y. Ma, and J. Hui, "Industrial applications of digital twins: A systematic investigation based on bibliometric analysis," *Adv. Eng. Informatics*, vol. 65, p. 103264, 2025.

[4] H. Asmat, I. U. Din, A. Almogren, and M. Y. Khan, "Digital twin with soft actor-critic reinforcement learning for transitioning from industry 4.0 to 5.0," *IEEE Access*, vol. 13, pp. 40 577–40 593, 2025.

[5] R. Sieve, P. Kobialka, L. Slaughter, R. Schlatte, E. B. Johnsen, and S. L. Tapia Tarifa, "BedreFlyt: Improving patient flows through hospital wards with digital twins," in *ASQAP*, ser. EPTCS, vol. 418. Open Publishing Association, 2025, pp. 1–15.

[6] Å. A. A. Kløvstad, P. Kobialka, R. Sieve, A. Pferscher, L. Slaughter, S. L. Tapia Tarifa, and E. B. Johnsen, "What-if scenarios for the BedreFlyt digital twin," in *Principles of Formal Quantitative Analysis*, ser. LNCS, vol. 15760. Springer, 2026, p. 360–381.

[7] C. Verdouw, B. Tekinerdogan, A. Beulens, and S. Wolfert, "Digital twins in smart farming," *Agricultural Systems*, vol. 189, p. 103046, 2021.

[8] E. Kamburjan, R. Sieve, C. P. Baramashetru, M. Amato, G. Barmina, E. Occhipinti, and E. B. Johnsen, "GreenhouseDT: An exemplar for digital twins," in *SEAMS*. ACM, 2024, pp. 175–181.

[9] N. Ariesen-Verschuur, C. Verdouw, and B. Tekinerdogan, "Digital twins in greenhouse horticulture: A review," *Comput. Electron. Agric.*, vol. 199, p. 107183, 2022.

[10] E. Kamburjan, N. Bencomo, S. L. Tapia Tarifa, and E. B. Johnsen, "Declarative lifecycle management in digital twins," in *EDTconf*, ser. MODELS Companion'24. ACM, 2024, pp. 353–363.

[11] X. Cao, Y. Yao, L. Li, W. Zhang, Z. An, Z. Zhang, L. Xiao, S. Guo, X. Cao, M. Wu, and D. Luo, "iGrow: A smart agriculture solution to autonomous greenhouse control," in *AAAI, IAAI, EAAI*. AAAI Press, 2022, pp. 11 837–11 845.

[12] H. M. Abbood, S. H. S. Alagheband, A. M. Imran, A. S. Mahdi, and M. A. Hassan, "Design adaptive non-linear PID control using reinforcement learning for optimal autonomous greenhouse microclimate regulation," *J. Adv. Comput. Intell. Intell. Informatics*, vol. 29, no. 6, pp. 1464–1483, 2025.

[13] Z. Huang, N. Yang, R. Cao, Z. Li, Y. He, and X. Feng, "Autonomous navigation system in various greenhouse scenarios based on improved FAST-LIO2," *Comput. Electron. Agric.*, vol. 234, p. 110279, 2025.

[14] R. Yan, X. He, Z. Wang, Y. Wang, Z. Li, and X. Li, "A compact autonomous inspection system for greenhouse environmental information sensing and three-dimensional visualization," *Comput. Electron. Agric.*, vol. 231, p. 109976, 2025.

[15] C. Hu, Z. Zhang, C. Li, M. Leng, Z. Wang, X. Wan, and C. Chen, "A state of the art in digital twin for intelligent fault diagnosis," *Adv. Eng. Informatics*, vol. 63, p. 102963, 2025.

[16] Q. Xu, G. Wen, Z. Lei, S. Gu, Y. Su, Z. Zhang, and X. Chen, "Deep digital twin-powered large vision-language model for multi-scenario industrial fault diagnosis," *Adv. Eng. Inform.*, vol. 69, p. 103997, 2026.

[17] S. Plitsos, E. Kostaki, C. Bardaki, and P. Eirinakis, "Enabling digital twins for industrial water management," *Intell. Decis. Technol.*, vol. 19, no. 5, pp. 3199–3225, 2025.

[18] M. Picone, P. Talasila, N. Bicocchi, and P. G. Larsen, "Exploring devops practices for lifecycle management of physical and digital twins in cyber-physical systems," in *ICPS*. IEEE, 2025, pp. 1–6.

[19] P. Talasila, C. Gomes, L. B. Vosteen, H. Iven, M. Leucker, S. Gil, P. H. Mikkelsen, E. Kamburjan, and P. G. Larsen, "Composable digital twins on digital twin as a service platform," *Simul.*, vol. 101, no. 3, pp. 287–311, 2025.

[20] E. Kamburjan, N. Bencomo, E. B. Johnsen, and S. L. Tapia Tarifa, "Declarative lifecycle management in digital twins," *Software and Systems Modeling*, 2026, to appear.

[21] S. Gil, E. Kamburjan, P. Talasila, and P. G. Larsen, "An architecture for coupled digital twins with semantic lifting," *Softw. Syst. Model.*, vol. 24, no. 5, pp. 1379–1404, 2025.

[22] E. Kamburjan, V. N. Klungre, R. Schlatte, S. L. Tapia Tarifa, D. B. Cameron, and E. B. Johnsen, "Digital twin reconfiguration using asset models," in *ISoLA*, ser. LNCS, vol. 13704. Springer, 2022, pp. 71–88.

[23] E. Kamburjan, V. N. Klungre, R. Schlatte, E. B. Johnsen, and M. Giese, "Programming and debugging with semantically lifted states," in *ESWC*, ser. LNCS, vol. 12731. Springer, 2021, pp. 126–142.

[24] E. Kamburjan, V. N. Klungre, Y. Qu, R. Schlatte, E. V. Kostylev, M. Giese, and E. B. Johnsen, "Semantically reflected programs," *Trans. Graph Data and Knowledge (TGDK)*, vol. 4, no. 1, pp. 3:1–3:52, 2026.

[25] E. Kamburjan, A. Pferscher, R. Schlatte, R. Sieve, S. L. Tapia Tarifa, and E. B. Johnsen, "Semantic reflection and digital twins: A comprehensive overview," in *The Combined Power of Research, Education, and Dissemination*, ser. LNCS, vol. 15240. Springer, 2025, pp. 129–145.

[26] E. Kamburjan, C. C. Din, R. Schlatte, S. L. Tapia Tarifa, and E. B. Johnsen, "Twinning-by-construction: Ensuring correctness for self-adaptive digital twins," in *ISoLA*, ser. LNCS, vol. 13701. Springer, 2022, pp. 188–204.

[27] R. Sieve, E. Kamburjan, F. Damiani, and E. B. Johnsen, "Declarative dynamic object reclassification," in *ECOOP*, ser. LIPIcs, vol. 333. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, pp. 29:1–29:31.

[28] E. Kamburjan and E. B. Johnsen, "Knowledge structures over simulation units," in *ANNSIM*. IEEE, 2022, pp. 78–89.

[29] J. Bjørnskov, A. Badhwar, D. Shikhar Singh, M. Sehgal, R. Åkesson, and M. Jradi, "Development and demonstration of a digital twin platform leveraging ontologies and data-driven simulation models," *J. Build. Perform. Simul.*, pp. 1–13, May 2025.