GreenhouseDT: An Exemplar for Digital Twins

Eduard Kamburjan
Riccardo Sieve
Chinmayi Prabhu Baramashetru
University of Oslo
Oslo, Norway
{eduard,riccasi,cpbarama}@ifi.uio.no

Marco Amato
Gianluca Barmina
Eduard Occhipinti
University of Turin
Turin, Italy

Einar Broch Johnsen
University of Oslo
Oslo, Norway
einarj@ifi.uio.no

ABSTRACT

Digital twins, which are increasingly adopted in industry, are model-centric systems used to improve the behavior of a twinned physical system. Seen as a whole, this system has several layers of self-adaptation: first, the digital twin manages its physical counterpart and maintains its models through a feedback loop to, e.g., fine-tune model parameters. Second, the digital twin needs to deal with unforeseen changes in the composition of the physical system, which require models to be partly replaced or recomposed. To facilitate research on self-adaptive digital twins, without requiring access to industrial production systems, this paper presents GreenhouseDT, an exemplar that explicitly separates these layers of self-adaptation. GreenhouseDT provides an extensible software architecture for a digital twin of a simple, low-cost greenhouse, in which plants, sensors and water pumps constitute the physical system. GreenhouseDT includes an asset model in the digital twin's knowledge base and uses reflection to lift twinned configurations into the knowledge base. We discuss how GreenhouseDT can be extended with different digital twin capabilities, demonstrated by the addition of plant health monitoring and model-based control.

KEYWORDS

self-adaptive systems, digital twins, knowledge representation

ACM Reference Format:
Eduard Kamburjan, Riccardo Sieve, Chinmayi Prabhu Baramashetru, Marco Amato, Gianluca Barmina, Eduard Occhipinti, and Einar Broch Johnsen. 2024. GreenhouseDT: An Exemplar for Digital Twins. In *SEAMS'24: ACM International Conference on Software Engineering for Adaptive and Self-Managing Systems, April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Digital twins are an excellent case for applying self-adaptation techniques. A digital twin [20] is a model-centric system; it maintains a digital replica of a physical system and interacts with the twinned physical system in two ways: the digital twin adapts its models of the physical system in near real-time using information events from the physical system and it can influence the behavior of the physical system using actuators (e.g., controllers) [11]. Digital twins can be used for descriptive ("what happened"), predictive ("what will happen") and prescriptive ("what if") analyses [21] of the twinned physical system, by exploring its models. As such, many twinned systems are self-adaptive, and can be implemented following an external approach for self-adaptive systems [24], in which a *managing subsystem* (part of the digital twin) controls a *managed subsystem* (the physical twin).

Digital twins also need to adapt to changes in the structure of the physical assets, as they evolve over time; e.g., the twinned physical assets may pass through different stages of an engineering life cycle: commissioning, via operations, to decommissioning. Consequently, the digital twin itself (i.e., the managing subsystem of the self-adaptive system) is also necessarily self-adaptive; the structure and capabilities of the physical assets evolve over time. This *architectural self-adaptation* (or *structural self-adaptation* [9]) differs from *behavior self-adaptation*, and it is recommended to separate these concerns into different self-adaptive layers [3]. For digital twins, behavioral self-adaptation amounts to adjusting the behavior of the physical twin, whereas architectural self-adaptation corresponds to reconfiguring the digital twin itself to reflect changes in the structure and requirements of the twinned assets.

We present GreenhouseDT[1], an exemplar to investigate self-adaptive digital twins. GreenhouseDT contributes a software architecture that enables exploring solutions not only for behavioral adaptation of a managed subsystem (the physical twin) but also for the architectural self-adaptation of the managing subsystem (the digital twin). Our architecture is based on knowledge graphs [6] that capture asset models. Asset models [22] (or digital threads [15]) are commonly used in engineering to record the current structure of the physical system and its changes. Knowledge graphs are a key technology to digitally represent asset models [18] and play an important role in digital twin architectures [12, 14, 23, 25], in particular to connect with standards [1, 13].

Digital twins today are heavily focused on industrial applications. These are often tightly coupled to complex and expensive production plants and laboratories, with proprietary solutions and data. The GreenhouseDT exemplar significantly lowers the barrier to having an operational digital twin for research purposes and enables solutions for self-adaptation in digital twins to be explored and compared. In this paper, we demonstrate such exploration at the level of architectural self-adaptation.

[1]Available under https://github.com/smolang/GreenhouseDT

Approach. The GreenhouseDT exemplar consists of a *greenhouse* as its physical system and an *extensible software architecture* that realizes the digital twin. The physical twin is a low-cost alternative to industrial plants, yet it provides exactly the features needed to explore digital twin technology: an easy-to-install physical system with off-the-shelf sensors and actuators, and a modular digital twin software layer. To facilitate the sharing of research results, the physical twin can be virtualized by replaying recorded data streams. The main focus of GreenhouseDT is the modular software architecture of the digital twin, its interconnection with the asset model of the physical twin, and the use of simulation models for prediction. The core of the software architecture of GreenhouseDT consists of an external self-adaptive system which includes a knowledge base that formalizes the asset model of the physical twin, and support for model-based predictions; we here use simulation models for simplicity. We assume that we are given simulation models that correspond to the different entities of the asset model (these decide the granularity of the architectural self-adaptation, discussed below). Sensor data (e.g., from humidity sensors) connect the digital twin to the greenhouse; these are stored in a time-series database which can be queried from the digital twin. The managing subsystem of the digital twin uses model-based predictions to decide how to control the actuators of the managed subsystem; we use a watering pump as an example of an actuator in the greenhouse.

Self-adaptation of the architectural configuration within the digital twin is realized by an additional self-adaptation layer that manages the managing system discussed above. This architectural self-adaptation layer ensures that changes in the asset model, reflecting changes in the physical system, are detected and that corresponding repair actions on the simulation model of the managing subsystem are performed. This way, we can ensure that the simulation model used to make decisions in the managing subsystem, is correctly configured with respect to the asset model, and thereby that it is kept in sync with the physical system. Observe that it is assumed that the asset model is updated to reflect changes in the physical twin; for simplicity in the exemplar, we let a human operator—generally an engineer but, for GreenhouseDT, a gardener—perform these update operations on the asset model.

The main contributions of this paper are

- a *self-adaptation digital twin exemplar* that can be equipped with different strategies for self-adaptation, enables the comparison of different self-adaptation strategies, and can form a basis for other digital twins; and
- an *adaptation logic for architectural self-adaptation*, combining asset models formalized in knowledge graphs with simple predictive decision making, which can easily be extended with other capabilities and adopted to other models for decision making.

Related Work. Several systems for self-adaptation in digital twins have been proposed by the literature, but most target engineering systems that are not suitable as research exemplars, such as the digital twin architecture proposed by Spaney et al. [19] which adapts the digital twin's runtie model to reflect machine degradations in a manufacturing system. As low-cost solutions, Feng et al propose an incubator digital twin [4], which targets *behavioral* self-adaptation. For architectural self-adaptation, Govindasamy et al. [5] propose an exemplar for air quality management, based on a self-adaptive

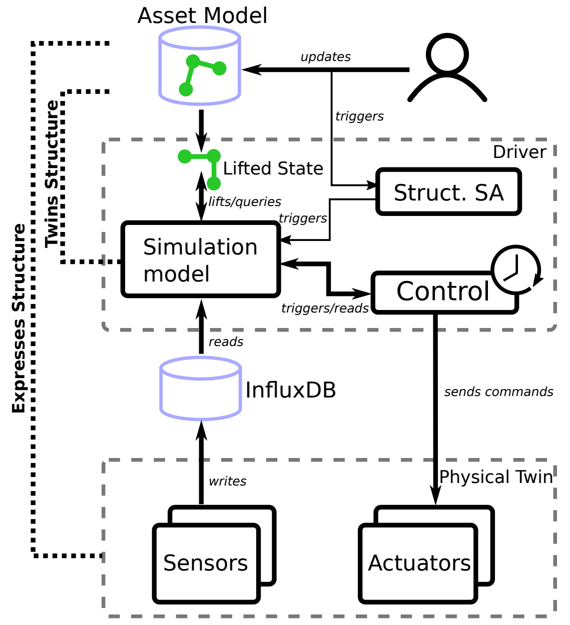

Figure 1: System Architecture. Components in the physical twin can be replaced with mock-ups that replay data.

software architecture [2]. Their system targets the evolution of the digital capabilities by means of domain-specific case-based reasoning, not structural changes in the physical asset, and does not contain a knowledge graph for asset management.

2 SYSTEM DESCRIPTION

GreenhouseDT consists of a physical twin (PT) and a digital twin (DT). The physical twin is the actual greenhouse, together with sensors and actuators, as well as software operating them. The digital twin is a simulation model of the greenhouse, together with an asset model and a time-series database.

2.1 The Need for Self-Adaptation

Whereas behavioral adaptation allows a greenhouse digital twin to calibrate the frequency or duration for running watering pumps in a greenhouse to how quickly soil in the pots gets dry, the greenhouse digital twin also needs to support architectural configuration. Consider a greenhouse with two shelves, in which plants are moved from the lower to the upper shelf after a certain amount of time. The digital twin must keep track of where a plant is positioned to map it to the correct sensor stream. Similarly, new knowledge may affect what is considered the optimal moisture level for a given plant. This knowledge may stem from new regulations, new objectives such as minimizing energy, or from contextual knowledge such as seasonal variation. The digital twin needs to adapt to such knowledge.

GreenhouseDT uses an asset model to capture knowledge about the domain and structure of the physical twin. We assume that this asset model is updated when the structure changes or new knowledge is added. The simulation model must twin this structure, using the asset model as a proxy. First, observe that these changes are not detected by the physical system itself, but are reported using

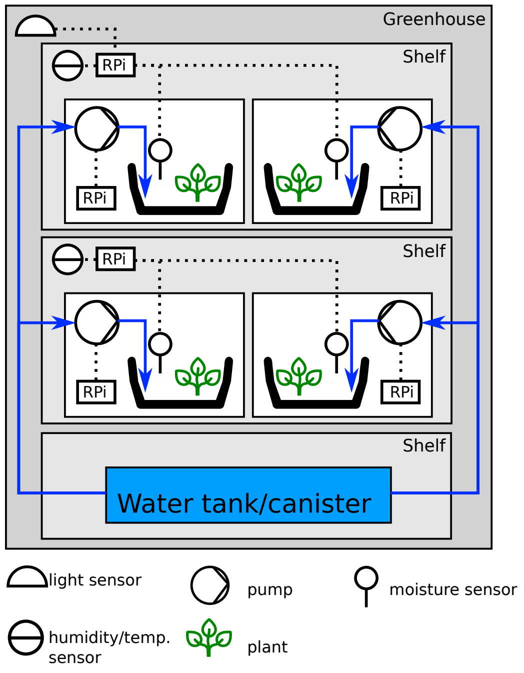

Figure 2: Structure of the physical twin.

the asset model, because they are not related to unexpected *behavior* of a single plant. Second, they are targeting a fundamental property of the system as a whole: the digtital twin must mirror the physical twin in behavior and structure. Otherwise, any control decision or analysis result may cause failure or mistakes when applied to the physical twin, even if no immediate error is raised.

2.2 System Overview

We first give an overview of the components in both DT and PT, shown in Figure 1.

Physical Twin. The physical twin of GreenhouseDT consists of a greenhouse with three shelves, pictured in Figure 2. On each of the two top shelves there are two plants, each equipped with a moisture sensor. Each shelf is equipped with a combined humidity/temperature sensor, and the whole greenhouse has a light sensor. As for actuators, each plant is connected to one water pump, which pumps water from a basin at the bottom shelf of the greenhouse.

Each shelf has a Raspberry Pi, which collects and relays data from the sensors on this shelf. to a time-series database that acts as an interface to the digital twin. Similarly, each pump is connected to a Raspberry Pi that receives commands to water its associated plant. The connections of the Raspberry Pi can be configured locally, but must adhere to the information of the asset model. The light sensor is handled by the minicomputer of the top shelf.

Digital Twin. The digital twin of GreenhouseDT uses a message-oriented middleware to connect the following components:

- **Simulation Model:** At its core, the digital twin is a program twinning the structure of the physical twin and providing operations for self-adaptation and control. In this program, each relevant part of the asset (e.g., actuators and plants) is represented by one object, connected according to the asset model.
- **Driver:** The driver triggers the digital twin's self-adaptation and control mechanism. It relays decisions to the actuators and gets input from a listener of the knowledge graph to trigger self-adaptation upon any change in the asset model. For simplicity, decision making is triggered in a fixed, configurable time interval.
- **Knowledge Graph:** The knowledge graph contains an asset model, a formal description of the architecture of the physical twin, in an RDF store.
- **Time-Series Database:** The time-series database serves as an interface to the sensors.

The last two components are conceptually straightforward and we next focus on the first two components.

2.3 Architectural Self-Adaptation

Architectural self-adaptation ensures that the *configuration* of the simulation model and its capabilities reacts to changes in the asset. The simulation model is a semantically lifted program [8], i.e., a program whose current runtime structure is interpreted as a knowledge graph. This interpretation, the *lifted state* of the program, contains each object currently present in the simulator, each value stored in their fields and their connections.

The simulation model is *reactive*; i.e., it does not perform its self-adaptation and decision making without an external trigger. Instead, it is embedded into a driver-component, which, upon every change of the asset model, triggers architectural self-adaptation. It is directly connected to both the time-series database (to access sensor data) and the knowledge graph (to access the asset model).

The architectural self-adaptation of the configuration of the simulation model executes so-called *defect queries* on the combined knowledge graph, which contains both the asset model and the lifted state of the simulation model. Each defect query corresponds to one property that connects model and asset; for example, there must be an asset in the asset model for each object in the simulation, and there must be an object in the simulation model that contains the right parameters for each asset for its control. Each defect query returns *witnesses* for its defect. The simulation model is implementing adaptation routines to repair these defects, i.e., to adapt itself to the changed asset model.

As an example, let us assume that the optimal moisture level for some asset has changed in the asset model. One of the defect queries returns as a witness the simulation object that corresponds to this asset and has the wrong (i.e., old) moisture level and the new optimal moisture level. The adaptation then updates the optimal moisture level in this object to the new optimal level.

Control. The driver issues a command to the simulation objects at regular, configurable intervals, to control the system to read the recent sensor data and make a decision, which is then transmitted to the actuators. This may implement behavioral self-adaptation.

3 PHYSICAL TWIN OF GreenhouseDT

The physical twin of GreenhouseDT is a greenhouse, depicted in Figure 2, including four plants on two shelves, sensors and actuators to realize control, and minicomputers to operate sensors and actuators. We used Raspberry Pi 4 Model B for the minicomputers.

Prebuilt indoor greenhouses are widely available, they are otherwise easy to assemble. We used a prebuilt greenhouse with three levels for the exemplar, as sensors and actuators need to be calibrated for the individual asset. On the lowest level is a canister with water for the watering system. The upper two levels have the same setup with two plants each. Our system uses basil plants, but any green plant is suitable.

Sensors. The following sensors monitor the physical twin:

- **Light:** For the overall greenhouse, one light-sensor monitors light conditions. Any sensor returning a value from 0-100 after calibration can be used, for example a webcam.
- **Temperature and Humidity:** For each shelf, one DHT22 sensor monitors temperature and humidity. These conditions are similar for both plants on a shelf, so we did not need one sensor per pot.
- **Moisture:** Each pot has one capacitive soil moisture sensor.

For each shelf, one minicomputer is used to collect data from all sensors on the shelf (the light sensor is considered to be part of the upper shelf) and to send this data to the time series database, as configured by the asset model. Each plant and sensor is identified by an alpha-numerical identifier as described in Figure 2.

Actuators. For each plant, there is one R385 water pump, controlled by a minicomputer, that connects the water canister with the corresponding plant. The actuator listens for messages using the message broker The URL and topic to which it subscribes and the basic local setup (e.g., the used GPIO to drive the pump and the id of the pump) are configured locally. The actuator activates the pump for the given amount of seconds that it receives in a message.

Distributing Experiments and Extensions. To share data and reproduce *results of software artifacts* for digital twins, a fully physical setup is not always suitable. For this reason, it is possible in GreenhouseDT to have a virtual greenhouse and feed prerecorded data to the InfluxDB to replay a situation. Similarly, it is possible in GreenhouseDT to issue a sequence of queries to the asset model, described below, to replay changes in the virtual greenhouse.

GreenhouseDT is modular with respect to the structure of the physical twin: Other sensors and actuators can be added, reusing most of the software on the minicomputers and without modifying the architecture. This modularity is discussed in detail in Section 5.

4 DIGITAL TWIN OF GreenhouseDT

GreenhouseDT realizes the simulation model in form of a program that mirrors the structure of the physical twin and connects to the infrastructure needed to communicate with the physical twin. To adapt to the changes in the configuration of the physical twin, this configuration is captured in an asset model.

4.1 Asset Model

The asset model is a knowledge graph that contains an OWL ontology describing the concepts needed to specify the greenhouse, and data that expresses its current configuration. The main classes and properties of the ontology are given in Figure 3. The ontology expresses the existence and properties of greenhouses, pots, and shelves. Additionally, it refers to water tanks/canisters, pumps and

```
Class: ast:Greenhouse
  SubClassOf:
    ast:hasWaterTank exactly 1 ast:WaterTank,
    ast:hasLightStrength some ast:percentage,
    ast:hasShelf min 1 ast:Shelf
Class: ast:Pot
  SubClassOf:
    ast:hasPlant max 1 ast:Plant,
    ast:hasMoisture some ast:percentage,
    ast:isWateredBy exactly 1 ast:Pump
Class: ast:Shelf
  SubClassOf:
    ast:hasHumidity some ast:percentage,
    ast:hasTemperature some xsd:double,
    ast:hasPot min 1 ast:Pot
DataProperty: ast:hasHumidityGpioPin
  Domain: ast:Shelf Range: xsd:int
DataProperty: ast:hasIdealMoisture
  Domain: ast:Plant Range: xsd:double
DataProperty: ast:hasIdealTemperature
  Domain: ast:Plant Range: xsd:double
```

Figure 3: Excerpt of the GreenhouseDT ontology.

plants. For simplicity in GreenhouseDT, we consider a specific ontology (and not a general description of greenhouses from an IoT or Smart Agriculture perspective): a greenhouse has two shelves, exactly one canister and a current light strength. A pot has at most one plant, exactly one pump and a current moisture level, while a shelf has a current humidity and temperature level, as well as exactly two pots. To model the ideal moisture and temperature for plant, two different data properties are used: `ast:hasIdealMoisture` and `ast:hasIdealTemperature`. To connect the configuration of the physical twin, the GPIO pin for the humidity sensor is also part of the asset model, using the data property `ast:hasHumidityGpioPin`. Figure 4 illustrates the representation of information about the greenhouse, its top shelf and the left pot and plant there. This ontology can easily be generalised or adopted to greenhouses with different layouts (see Section 6).

4.2 Simulation Model

The simulation model is written in SMOL [8], an object-oriented language that can be semantically lifted; i.e., every runtime state can be interpreted as a knowledge graph. Every kind of asset is modeled by one class (e.g., plant, pump, and shelf). Figure 5 shows the class for plants, a data class with three fields that correspond to values from an asset model. A SMOL object is a twin of an asset if they have the same `plantId`. The method `getPotMoisture` realizes the connection to the time series database. Note that the control and self-adaptation logic is external to these classes.

The simulation model is started once the digital twin is initialized and is afterwards used in two ways: for *adaptive control* and for *architectural self-adaptation*. For adaptive control, where a decision procedure is executed for each plant to determine whether, and for how long, the pump should be activated. This is done by a procedure executed by the simulation driver, detailed below. For architectural self-adaptation, the simulation driver executes another procedure

```
Individual: ast:greenhouse
  Types: ast:Greenhouse
  Facts: ast:hasShelf  ast:shelf1,
         ast:hasShelf  ast:shelf2,
         ast:hasWaterTank  ast:waterTank
Individual: ast:shelf1
Types: ast:Shelf
Facts: ast:hasHumidityGpioPin  4,
       ast:hasShelfFloor  "1",
       ast:hasTemperatureGpioPin  4
       ast:composedOf  ast:pot1,
       ast:composedOf  ast:pot2
Individual: ast:pot1
  Types: ast:Pot
  Facts: ast:hasPlant  ast:basil1,
         ast:isWateredBy  ast:pump1,
         ast:hasPotPosition  "left"
```

Figure 4: Excerpt of the GreenhouseDT asset model.

that semantically lifts the program state of the simulation model and then runs defects queries on the combined knowledge graph. For example, the query in the upper part of Figure 6 detects the defect that the optimal moisture level of a plant has changed, while the lower part shows an except of the repair function that sets the moisture level to the correct, new value.

4.3 Digital Twin Infrastructure

The simulation model is connected to *(a)* an InfluxDB database, where each `Plant` object retrieves the recent values from the sensors of its plant, *(b)* a Fuseki SPARQL endpoint that stores the asset model, and *(c)* the simulation driver. The simulation driver, implemented in Java, has the task to regularly trigger adaptive control and, upon changes in the asset model, architectural self-adaptation.

The communication between actuators, asset model, and the simulation driver is implemented using an ActiveMQ message broker middleware. Each query that changes the knowledge graph (i.e., every `Update` query), also triggers a message that causes the simulation driver to start architectural reconfiguration. Other messages are sent by the simulation driver after triggering control to issue commands to the actuators.

Lastly, the digital twin infrastructure includes a graphical web interface that visualizes the data in the time series database, the logs of the simulation model (i.e., control decisions and reconfigurations) and the updating queries for the asset model.

5 USING GreenhouseDT

We now discuss how to install and extend the GreenhouseDT exemplar, with a focus on the software components. Detailed instructions on hardware and software setup, including instructions for calibration, are available as part of the exemplar. GreenhouseDT is distributed in two forms: (1) a VM and two images for the Raspberry Pi's as an easy to install and use solution, and (2) an open source repository. The VM and images are generated from the open source repository. To redistribute data, the VM can be used with prerecorded data without the Raspberry Pi OS images.

Installing the Minicomputers. To install the minicomputers for the sensors, one has to configure (a) their connection to the other components and (b) their local parameters.

For the former, we need a URL, organization and access token of the InfluxDB time series database, enter the minicomputer's ID in the asset model (used to write into the correct bucket), and the IDs of the shelf and plants it manages. Furthermore, the URL and topic of the message broker must be provided.

For the latter, we need the GPIO pins, and maximal and minimal values for all connected sensors, e.g., minimal and maximal lux values from the light sensor. These values are used to normalize the output—all data in the time series database range between 0 and 100. The data is automatically sent to the configured database. The message broker loads a new configuration when a new configuration file is sent, e.g., to use a different GPIO pin.

Installing the microcomputers for the actuators similarly requires to configure the connection to the message broker, the GPIO of the actual pump and the ID according to the asset model.

Setting up the Digital Twin. The digital twin VM runs the simulation driver and the two databases, contains the asset model for the knowledge graph and a predefined schema design for the influxDB time series database. To run driver and databases on different machines, one can either set them up manually and reconfigure the connection, or follow the instruction in the open source repository.

Configuration is split in several files. The driver's connection to the databases is configured by the main file `config_local.yml`. Each shelf *i* is configured in a file `config_shelf_`*i*`.ini`, where *i* is the ID of the shelf in the asset model. These files mirror the configuration of the sensor minicomputers—whenever they are modified locally in the digital twin, they are also sent via the message broker to the corresponding minicomputer where they are loaded.

6 EVALUATION

For convenience, GreenhouseDT provides a dashboard monitoring the state and communication of the system. In particular, it shows the moisture level of the plants, queries run on the knowledge graph and output from the simulation driver. To demonstrate simple reconfiguration, a preconfigured scenario adds a new plant to the asset model and shows how the digital twin reconfigures.

We evaluate the extensibility of GreenhouseDT by adding two new capabilities; both fully implemented and part of the exemplar. For a detailed description of which parts of the code have been modified, we refer to the online documentation.

6.1 Plant Health Monitoring

We first extend the twin with the capability to detect the health status of a plant. This extension is representative for new capabilities obtained by extending the physical twin or its digital requirements, e.g., adding new sensors, actuators, or assets to the physical system,

An infrared camera is added to the physical twin, and the asset model and databases are extended to keep track of limits for recorded NDVI[2] values. Depending on this value, the digital twin will report on the current health status. The health statuses provide a classification over the NDVI measurements, specified in the asset

[2]Normalized difference vegetation index, a measure that can be used for the health analysis of vegetation [10].

```
class Plant extends Twin (String plantId, Double idealMoisture, String plantType)
    Double getPotMoisture()
        List<Double> influxReturn = access(/* InfluxQL query*/, INFLUXDB("config_local.yml"), /* parameter: */ this.plantId);
        //default: aggregated value, depending on InfluxQL query
        return influxReturn.get(0);
    end
end
```

Figure 5: A class representing the simulation model of a plant.

```
SELECT ?obj ?idealMoistureNew {
  ?obj a prog:Plant;
       prog:Plant_plantId ?plantId ;
       prog:Plant_idealMoisture ?idealMoisture .
  ?y rdf:type ast:Plant;
     ast:hasPlantId ?plantId ;
     ast:hasIdealMoisture ?idealMoistureNew .
  FILTER(?idealMoisture != ?idealMoistureNew ) }
```

```
List<MoiRepair> repairs = construct(Q); //query above
for r in repairs do
  r.obj.idealMoisture = r.idealMoistureNew; end
```

Figure 6: Architectural self-adaptation.

model. These statuses may change over time; GreenhouseDT automatically detects the current classification from the asset model—the statuses are treated as a kind of digital asset, and can can be used, e.g., to adjust watering or for reporting. To add health monitoring, we need to *(a)* add an infrared sensor, *(b)* extend the asset model and the influxDB schema, and *(c)* extend the simulation model.

- **Physical Twin:** The physical twin is extended with a Pi Cam v2 NoIR without an infrared filter. The software of the corresponding minicomputer is extended in two ways: (1) the configuration is extended with a new bucket and normalization range, and (2) the code for reading sensor data is extended to capture images, calculate the average NDVI per pixel and write to the time series database. Auxiliary functions for such tasks are available.
- **Digital Twin Infrastructure:** The time series database schema is extended with the corresponding buckets for NVDI value, and the asset model with a class `ast:HealthState` for a possible health status, with properties `ast:minNVDI` and `ast:maxNVDI` for the threshold values, between which a plant is considered in a given health state.[3] The driver does not need to be extended.
- **Simulation Model:** The simulation model needs two extensions. The architectural self-adaptation is implemented by adding defect queries that detect changing NDVI limits, similar to the optimal moisture levels, and the addition or removal of NDVI sensors for a given plant. Additionally, health statuses and their defect queries need to be added. For this purpose, the simulation model provides an interface (`KindModel`) that needs to be implemented and initialized on startup. Finally, the `KindModel` instantiation for plants needs to be extended to detect which status a plant has and output it to the user.

[3]For simplicity, we here assume that only adult plants of a rather similar size are used, as the average NDVI per pixel will be lower if there is less leaf area.

6.2 Model-Based Control

Let us now extend the twin with the capability to perform *model-based* control. This extension is representative for new capabilities based on extending the simulation model.

The decision to water is no longer made based on the measurements from the moisture sensor, but rather on a *digital shadow* [20], i.e., a model that predicts the moisture level. We monitor the model drift by comparison to the sensor stream, automatically reconfiguring the model if it drifts too far for reliable model-based control.

Neither physical twin, asset model nor time series data base need to be extended. The simulation driver and model is extended as follows. First, the behavior of the moisture level is approximated by a simple linear model in OpenModelica [17] and exported as a simulation unit using the FMI [16] interface. Second, each plant in the simulation model is extended with an instance of this simulation unit, using the available SMOL-FMI interface [7]. Upon creation of a new object, the unit is initialized using the last read moisture level from InfluxDB. Third, the decision procedure is changed to base the decision to water on the *simulated* moisture level, not the read one. Finally, a new *drift* procedure that is added. This procedure is called each time the sensor stream is read and advances the simulation accordingly. The procedure compares the last values in the time series database with the simulated values at this time. If the difference is too big (i.e., the model drifted too far), the model is automatically reset with the last, correct sensor value.

7 CONCLUSION

GreenhouseDT is a digital twin exemplar that emphasizes the software architecture and self-adaptation aspects of digital twins, based on knowledge graphs and asset models, two technologies widely used in Industry 4.0 and related initiatives. The exemplar is an easy-to-install low-cost alternative to expensive or highly complex industrial digital twin that often come with proprietary solutions and data. GreenhouseDT is extensible, and easy to distribute for reproducibility and replication of research results. The former is facilitated by an extendable architecture, while the latter is enabled through the ability to share data and results through replays, which does not require the physical system to be recreated.

ACKNOWLEDGMENT

This work was partly funded by the RCN project *PeTWIN* (grant no. 294600) and the EU project *Sm4rtenance* (grant no. 101123423).

REFERENCES

[1] Sebastian R. Bader and Maria Maleshkova. 2019. The Semantic Asset Administration Shell. In *SEMANTiCS (Lecture Notes in Computer Science, Vol. 11702)*. Springer, 159–174.

[2] Tim Bolender, Gereon Bürvenich, Manuela Dalibor, Bernhard Rumpe, and Andreas Wortmann. 2021. Self-Adaptive Manufacturing with Digital Twins. In *SEAMS@ICSE*. IEEE, 156–166.

[3] Víctor A. Braberman, Nicolás D'Ippolito, Jeff Kramer, Daniel Sykes, and Sebastián Uchitel. 2015. MORPH: a reference architecture for configuration and behaviour self-adaptation. In *Proc. 1st International Workshop on Control Theory for Software Engineering (CTSE@FSE 2015)*, Antonio Filieri and Martina Maggio (Eds.). ACM, 9–16. https://doi.org/10.1145/2804337.2804339

[4] Hao Feng, Cláudio Gomes, Casper Thule, Kenneth Lausdahl, Michael Sandberg, and Peter Gorm Larsen. 2021. The Incubator Case Study for Digital Twin Engineering. *CoRR* abs/2102.10390 (2021).

[5] Hari Shankar Govindasamy, Ramya Jayaraman, Burcu Taspinar, Daniel Lehner, and Manuel Wimmer. 2021. Air Quality Management: An Exemplar for Model-Driven Digital Twin Engineering. In *MoDELS (Companion)*. IEEE, 229–232.

[6] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. 2022. Knowledge Graphs. *ACM Comput. Surv.* 54, 4 (2022), 71:1–71:37. https://doi.org/10.1145/3447772

[7] Eduard Kamburjan and Einar Broch Johnsen. 2022. Knowledge Structures Over Simulation Units. In *Annual Modeling and Simulation Conference, ANNSIM 2022, San Diego, CA, USA, July 18-20, 2022*, Cristina Ruiz Martin, Niloufar Emami, María Julia Blas, and Roya Rezaee (Eds.). IEEE, 78–89. https://doi.org/10.23919/ANNSIM55834.2022.9859490

[8] Eduard Kamburjan, Vidar Norstein Klungre, Rudolf Schlatte, Einar Broch Johnsen, and Martin Giese. 2021. Programming and Debugging with Semantically Lifted States. In *Proc. 18th International Conference on the Semantic Web (ESWC 2021) (Lecture Notes in Computer Science, Vol. 12731)*, Ruben Verborgh, Katja Hose, Heiko Paulheim, Pierre-Antoine Champin, Maria Maleshkova, Óscar Corcho, Petar Ristoski, and Mehwish Alam (Eds.). Springer, 126–142. https://doi.org/10.1007/978-3-030-77385-4_8

[9] Eduard Kamburjan, Vidar Norstein Klungre, Rudolf Schlatte, Silvia Lizeth Tapia Tarifa, David Cameron, and Einar Broch Johnsen. 2022. Digital Twin Reconfiguration Using Asset Models. In *Leveraging Applications of Formal Methods, Verification and Validation. Practice - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 13704)*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer, 71–88. https://doi.org/10.1007/978-3-031-19762-8_6

[10] F. J. Kriegler, W. A. Malila, R. F. Nalepka, and W. Richardson. 1969. Preprocessing Transformations and Their Effects on Multispectral Recognition. In *Remote Sensing of Environment, VI*. University of Michigan, 97.

[11] Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. 2018. Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine* 51, 11 (2018), 1016–1022. https://doi.org/10.1016/j.ifacol.2018.08.474 INCOM 2018.

[12] Han Li, Guoxin Wang, Jinzhi Lu, and Dimitris Kiritsis. 2022. Cognitive twin construction for system of systems operation based on semantic integration and high-level architecture. *Integr. Comput. Aided Eng.* 29, 3 (2022), 277–295.

[13] Yuanfu Li, Jinwei Chen, Zhenchao Hu, Huisheng Zhang, Jinzhi Lu, and Dimitris Kiritsis. 2022. Co-simulation of complex engineered systems enabled by a cognitive twin architecture. *Int. J. Prod. Res.* 60, 24 (2022), 7588–7609.

[14] Franz Georg Listl, Jan Fischer, and Michael Weyrich. 2023. An Architecture for Knowledge Graph based Simulation Support. In *ETFA*. IEEE, 1–8.

[15] Tiziana Margaria and Alexander Schieweck. 2019. The Digital Thread in Industry 4.0. In *IFM (Lecture Notes in Computer Science, Vol. 11918)*. Springer, 3–24.

[16] Modelica Association. [n. d.]. FMI Standard 2.0.3. https://github.com/modelica/fmi-standard/releases/download/v2.0.3/FMI-Specification-2.0.3.pdf.

[17] Open Source Modelica Consortium. [n. d.]. OpenModelica. https://openmodelica.org/.

[18] Nada Sahlab, Simon Kamm, Timo Müller, Nasser Jazdi, and Michael Weyrich. 2021. Knowledge Graphs as Enhancers of Intelligent Digital Twins. In *ICPS*. IEEE, 19–24.

[19] Patrick Spaney, Steffen Becker, Robin Ströbel, Jürgen Fleischer, Soraya Zenhari, Hans-Christian Möhring, Ann-Kathrin Splettstößer, and Andreas Wortmann. 2023. A Model-Driven Digital Twin for Manufacturing Process Adaptation. In *ModDiT@Models*.

[20] Fei Tao, He Zhang, Ang Liu, and A. Y. C. Nee. 2019. Digital Twin in Industry: State-of-the-Art. *IEEE Transactions on Industrial Informatics* 15, 4 (2019), 2405–2415. https://doi.org/10.1109/TII.2018.2873186

[21] Raymon van Dinter, Bedir Tekinerdogan, and Cagatay Catal. 2022. Predictive maintenance using digital twins: A systematic literature review. *Information and Software Technology* 151 (2022), 107008. https://doi.org/10.1016/j.infsof.2022.107008

[22] Constantin Wagner, Julian Grothoff, Ulrich Epple, Rainer Drath, Somayeh Malakuti, Sten Grüner, Michael Hoffmeister, and Patrick Zimmermann. 2017. The role of the Industry 4.0 asset administration shell and the digital twin during the life cycle of a plant. In *ETFA*. IEEE, 1–8.

[23] Maryna Waszak, An Ngoc Lam, Volker Hoffmann, Brian Elvesæter, Maria Flavia Mogos, and Dumitru Roman. 2022. Let the Asset Decide: Digital Twins with Knowledge Graphs. In *ICSA Companion*. IEEE, 35–39.

[24] Danny Weyns. 2020. *An Introduction to Self-adaptive Systems: A Contemporary Software Engineering Perspective*. John Wiley & Sons.

[25] Xiaochen Zheng, Jinzhi Lu, and Dimitris Kiritsis. 2022. The emergence of cognitive digital twin: vision, challenges and opportunities. *Int. J. Prod. Res.* 60, 24 (2022), 7610–7632.