MODELING AND SYNCHRONIZING DIGITAL TWIN ENVIRONMENTS

Juan Alberto Llopis
Javier Criado
Luis Iribarne

Applied Computing Group (TIC-211)
University of Almería, SPAIN
{jalbertollopis,javi.criado,
luis.iribarne}@ual.es

Paula Muñoz
Javier Troya
Antonio Vallecillo

ITIS Software
University of Málaga, SPAIN
{paulam,jtroya,av}@uma.es

ABSTRACT

A digital twin is a virtual replica of a system defined at a certain level of fidelity and synchronized to a specific frequency. Digital twins are often used to replicate physical systems whose simulations are computationally expensive. If modeling the environment of any system is generally difficult, the problem is even harder in the case of digital twins, since the model of their environment must be synchronized with that of the physical system too. In this paper, we show how the environment of a digital twin can be effectively modeled and kept in sync with the real one, by adding digital proxies of the relevant elements of the environment to the models of the digital twin elements, and connecting them using the same synchronization mechanisms used by the twins. We demonstrate our approach with a case study of a smart room with sensors, using high-level software models.

Keywords: digital twin, environment, model-based software engineering, simulation.

1 INTRODUCTION

A *Digital Twin* (DT) is a comprehensive digital representation of an actual system, service or product (the *Physical Twin*, PT), synchronized at a specified frequency and fidelity (Digital Twin Consortium 2021). The DT includes the properties, condition and behavior of the physical entity through models and data, and is continuously updated with real-time system data about the PT performance, maintenance, and health status throughout its entire lifetime (Grieves 2014).

Designing and engineering a digital twin system is a difficult task because it entails several complex problems (Bordeleau et al. 2020). The first one is the development of the simulation models of the physical system at the appropriate level of abstraction and resolution that can provide faithful results in a reasonable computational time. Then we need to verify and validate the models, checking that they are in fact faithful enough to the system they emulate. Implementing the connections between the two twins is also a challenge due to the potentially stringent scalability and response time requirements. Putting in place the continuous synchronization mechanisms between the twins may represent yet another challenge, while developing and especially testing the prediction and visualization services of any digital twin system are not easy tasks either. Finally, the co-evolution of the two twins must also be considered.

One thing that is usually involved in all these design and development tasks is the system environment. By environment, we mean the surroundings (natural or human-made) in which the system of interest is utilized and supported (INCOSE 2015). Elements of the environment are not part of the system, and interact with the system elements through well-defined *interfaces*, which define the shared boundary across which information is passed (ISO 24765 2017). Normally, information is transmitted in both directions between the system and its environment. Environment elements, as their system counterparts, have properties. For example, a person in a room switches on the light and the brightness of the room increases. The system consists of a light bulb and a switch. The environment comprises the room and the person. One of the properties of the room is its brightness, which is directly related to one of the properties of the light, its luminosity. Interactions can be either actions (e.g., turning on the switch) or effects (e.g., an increase in the luminosity of the room), and can work in both ways.

So far, most of the software engineering efforts have focused on modeling the *context* of a system, i.e., the representation of its immediate environment (ISO 24765 2017), which is normally achieved with variables that represent the environment properties. However, in a digital twin system, we do not only need to have a digital replica of the system environment in order to analyze and predict its behavior but it also must be synchronized with the real system environment.

In this paper, we show how the environment of a digital twin can be effectively modeled and kept in sync with the real environment, by adding digital proxies of the relevant elements of the environment to the models of the digital twin elements, and by connecting them using the same synchronization mechanisms used by the twins. We demonstrate our approach with a case study of a smart room with sensors, using high-level software models.

Paper structure. After this introduction, Sect. 2 describes the background of our proposal and presents the running example used in the paper to illustrate it. Then, Sect. 3 describes our approach and demonstrates it with the running system. Some of the advantages and limitations of our work are discussed in Sect. 4, while Sect. 5 relates our work to other similar proposals. Finally, Sect. 6 presents the conclusions and outlines some future lines of work.

2 BACKGROUND

2.1 Digital Twin Systems and their Architectures

Although the term Digital Twin is commonly used nowadays, different authors assign different meanings to it. In this work, we will adopt the definition by the Digital Twin Consortium (2021): A *Digital Twin* (DT) is a comprehensive digital representation of an actual system, service or product (the *Physical Twin*, PT), synchronized at a specified frequency and fidelity.

Note that the term Digital Twin (DT) refers just to the virtual replica of the Physical Twin (PT). Thus, we will use the term *Digital Twin System* (DTS) to refer to the system composed of the DT, the PT, and the connections between them, i.e., those that implement the synchronization mechanisms and the continuous update of information between the two twins. A DTS may also contain a set of additional services to provide, among others, visualization dashboards, prediction capabilities, or monitoring components (Tao et al. 2019, Grieves and Vickers 2017).

In the literature, several architectures have been proposed to realize Digital Twin Systems. These architectures differ in the number of dimensions included as part of the DTS. For example, the most basic architectures include the three mentioned dimensions (DT, PT, and the connections) (Grieves and Vickers 2017), while other further proposals include five (Tao et al. 2018, Tao et al. 2019) or more dimensions (Sharma et al. 2020) to include elements such as prediction services or dashboards.

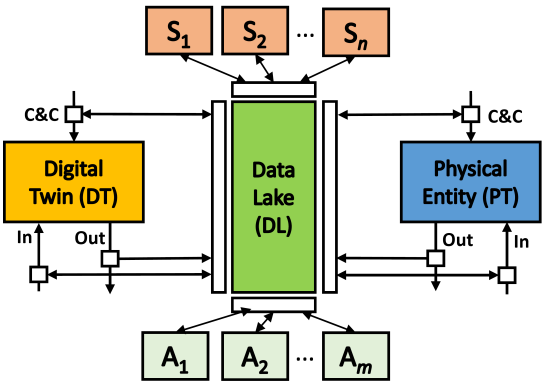


Figure 1: A Digital Twin System Framework.

In this paper, we follow a reference architecture of five dimensions that uses a data lake for connecting all the DTS constituent elements, including the DT, the PT, a set of services, and some analysis components, as described by Muñoz, Troya, and Vallecillo (2021). The service and analysis components are optional since a system does not have to include them to be considered a DTS. The existence of a replica and a bidirectional connection is enough. That reference architecture is depicted in Fig. 1. The PT is shown to the right and the DT, its specular image, to the left. The PT interacts with the environment through *inputs* (In) and *outputs* (Out). The values of these inputs are generated by sensors that detect changes in the natural environment. Meanwhile, the outputs are the reactions of the PT to environmental inputs, commands issued to the PT (represented by the *Command and Control* (C&C) incoming arrow), or to internal changes. These latter are usually captured by other sensors, which inform about the state of the PT. The behavior of the DT is the same, except the environmental inputs may come from the real environment or a simulated environment.

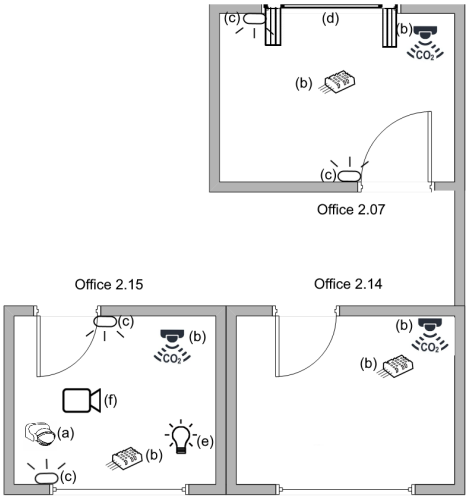
In this architecture, the communication between the twins and the different components is achieved through a *Data Lake* (DL) (Hai, Quix, and Jarke 2021), which enables loosely-coupled, asynchronous communication between the different components of the architecture. It implements a Blackboard architectural pattern (Buschmann et al. 1996), and it is accessed using drivers that transform the data into the formats accepted by each component. The drivers are represented in Fig. 1 by rectangles surrounding the DL.

The *Service* components (S_1,S_2,..., S_n) implement additional functionality, such as dashboards or predictive algorithms. Note that in the Blackboard architectural pattern, any (permitted) component can write in the DL. This means that services can issue commands to the physical entity as if they were its external users, enabling, e.g., the implementation of self-adaptive behaviors. Finally, the *Analysis* components (A_1,A_2,..., A_n) are in charge of implementing different types of tests. For example, a monitoring component can check that the traces produced by both the DT and the PT represent equivalent behaviors (Muñoz et al. 2022).

2.2 Running example: A smart room with sensors

2.2.1 The physical system

This section describes the example that we have selected to demonstrate our proposal. It comprises three rooms in one of our buildings, equipped with sensors and actuators. The physical devices of the system are deployed in these three offices. Furthermore, a weather station is located on the rooftop, which is used for analyzing the external weather conditions.



(a) Floor plan of the three offices.

(b) View of Office 2.15.

Figure 2: The physical system.

The type of devices deployed are (a) motion sensors; (b) temperature, humidity, and CO_2 sensors; (c) contact sensors in the doors and windows; (d) smart blinds; (e) smart lights, and (f) video cameras. Fig. 2 shows the floor plan of the offices with the devices deployed in each one and a photo of Office 2.15.

In this example let us focus on three specific devices. First, the motion sensor in Office 2.15 is used to detect moving objects in the room, e.g., people. The sensor has an internal value that is modified when an object is present in its vision range. Second, the actuator in Office 2.07 can open or close the room blinds, and stores how open it is by means of a percentage (0=closed, 1=completely open). Finally, the light bulbs of the smart lights in Office 2.15 have a brightness level, a saturation level, and several properties to define the light color. These types of smart lights allow operations to turn their bulb on or off, change their brightness and make them blink. They also include a sensor that can detect the illumination level of the room.

2.2.2 The Digital Twin

The Digital Twins of the rooms, including all the sensors and actuators, were developed as high-level models using the Unified Modeling Language (UML) (Object Management Group 2015). We employed the USE (Gogolla, Büttner, and Richters 2007) as a modeling tool because it enables the definition of the system behavior and the execution of the model instances to perform simulations. Additionally, it allows the definition of constraints over the model to check invariants during execution using Object Constraint Language (OCL) (Object Management Group 2014). We decided to use executable models to realize the Digital Twin to abstract ourselves from a concrete implementation, centering the work on demonstrating our approach and testing it with models which enable runtime verification of system properties and a lighter development process. Afterward, applying the proposed techniques, these models could be transformed into any concrete implementation.

To represent IoT devices of the Web of Things (WoT) and their information we will use the metamodel proposed by the W3C, called *Thing Description* (TD) (Charpenay, Käbisch, and Kosch 2016), an initiative proposed by Guinard, Trifa, and Wilde (2010) with the idea of improving the interoperability of heterogeneous IoT devices. The behavior of IoT devices is described by *InteractionAffordance*, which allows describing the operations and interactions of *things* by defining their properties, actions, and events (Charp-

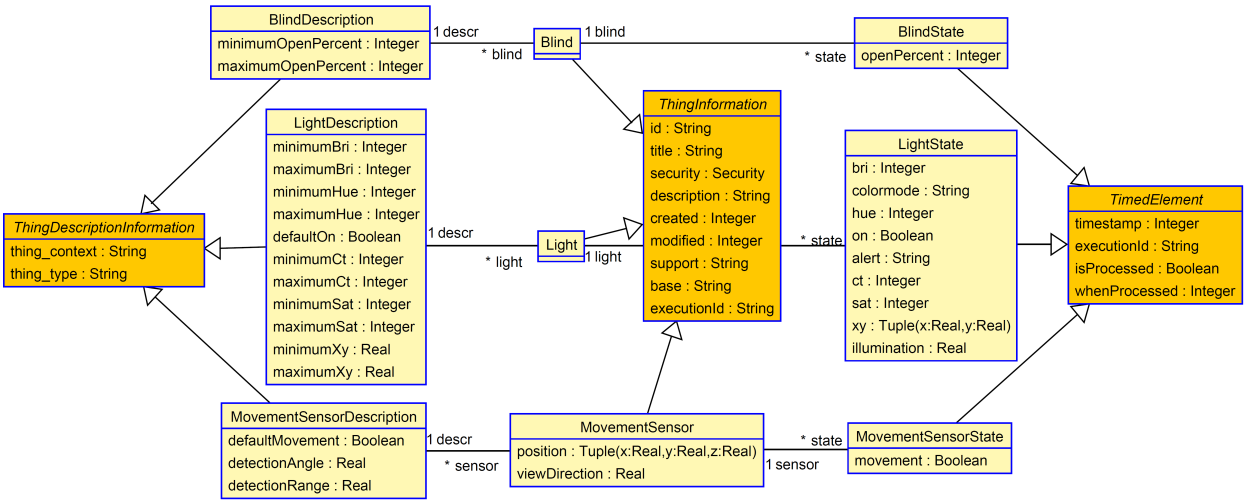


Figure 3: Class diagram with the conceptual model of the system.

enay and Käbisch 2020). Properties are captured by the attributes of the device. For instance, the properties of a light bulb are its brightness and its state (on/off). Actions represent the operations that the device can perform, e.g., turning its light on/off, changing its brightness, and blinking. Finally, events are operations that are executed as a result of the actions of other devices. For example, the light may turn on when the motion sensor detects presence in the office.

Figure 3 shows a UML class diagram with the conceptual model of the physical system. Each device is represented by three classes, following the TD specification. First, the devices themselves (*Things*), represented in this case by classes Light, Blind and MovementSensor. Second, the values of the device parameters (*Thing Descriptions*), namely LightDescription, BlindDescription and MovementSensorDescription. Finally, the values of the properties of each device at each moment in time (*State*): LightState, BlindState and MovementSensorState. Abstract classes ThingDescriptionInformation, ThingInformation and TimedElement capture the common features of the descriptions, things and states, respectively.

Based on this conceptual model, a snapshot of the system with its state at a given moment in time is specified by an object model with the current instances and their states. For example, the object model shown in Fig. 4 describes the state of our system at time 0, where we suppose the system execution starts. It contains one motion sensor and one light in Office 2.15; one light in Office 2.14; and one light and one blind in Office 2.7. A Clock object stores the time in seconds, using the POSIX format (IEEE Std 1003.1-2008 2016).

2.2.3 The Digital Twin system

After the PT is defined and the DT is modeled, we need to define the connections between them to implement the synchronization mechanisms, thus creating the *Digital Twin System* (DTS). As explained in section 2.1, the connections between the PT and the DT are established using a Data Lake (DL), which stores the data generated by the sensors and the operations performed on the actuators, as well as the states of all DT and PT objects along time. The DL is developed in our system using Neo4J (Robinson, Webber, and Eifrem 2015), a graph database that allows the system to establish relationships between the time and states of the DTs, thus facilitating the search and comparison of the states, taking into account (or not) their timestamps.

The DT interacts with the DL through a driver developed as a USE plugin. The driver scans the object model, checking for changes. When an object representing a device creates new data states or issues a new

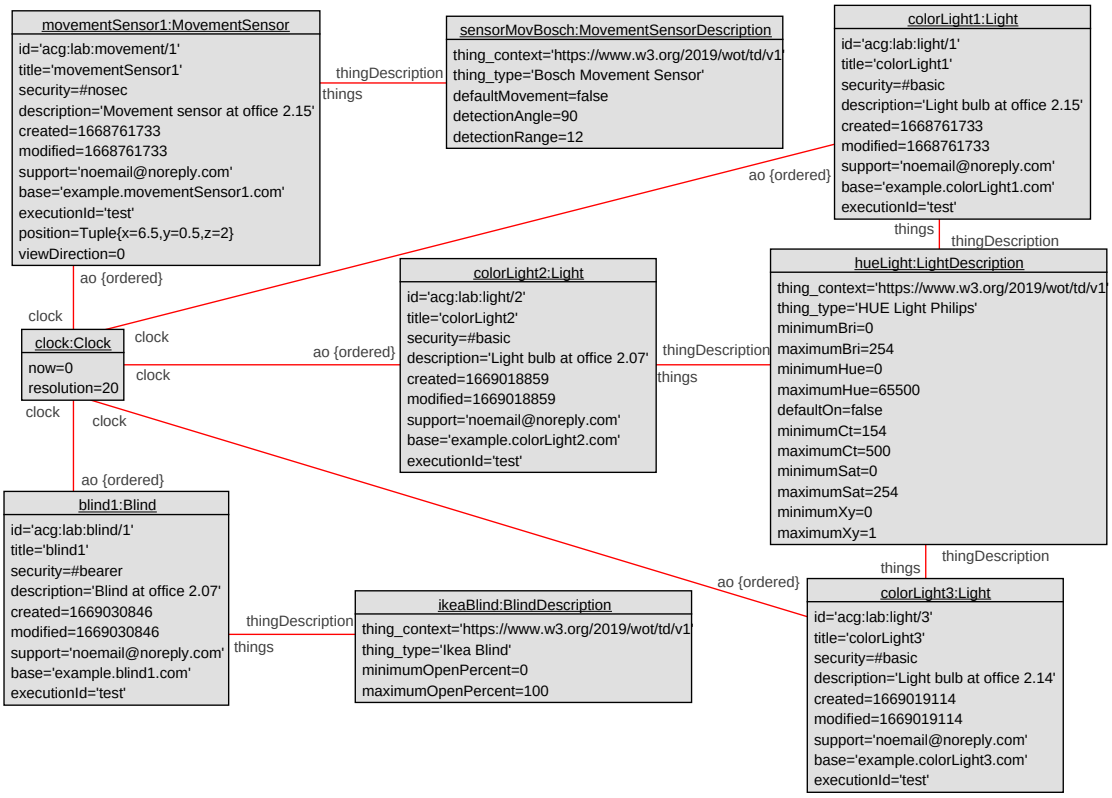


Figure 4: Initial state of the running example.

command, the plugin stores them in the DL and deletes them from the model to reduce memory overload. Similarly, when the driver recognizes a record in the database that is addressed to the DT, it creates the corresponding object in the USE object model. The synchronization works analogously with the physical twin: a driver captures the values of the sensors and writes them in the DL database. When a record describes an action to be performed on the system, the driver is in charge of executing it (e.g., by acting on the actuators).

2.2.4 The environment of the system

The environment includes all external sources that interact with the physical system. These include people that can enter the office, move around it, turn the lights on and off, open or close the blinds, or leave the room. External conditions such as weather events that affect the brightness of the rooms such as sunshine, darkness, or heavy clouds, can also be part of the environment of the system.

The interface between the system and its environment is realized through *sensors* that detect the state of the environment (e.g., illumination, presence of people) and the *commands* performed on the system's actuators (e.g., switching on/off lights or opening/closing blinds). Next section describes how these elements of the environment are modeled in the DT and synchronized with those of the physical system.

3 MODELING THE ENVIRONMENT OF A DIGITAL TWIN

In this paper, we propose modeling the environment of the system by means of proxies of its interface elements, both *sensors* and *commands*. Fig. 5 shows a UML class diagram with the model of some of

these elements, namely those that represent the values of the brightness and motion sensors (Illumination and MovingObject, resp.) and the commands used to control the blind's movements (BlindOpen, BlindClose, BlindStop), switch a light on or off (LightToggle), change its brightness (LightBri), and make it blink (LightAlert).

3.1 Modeling the environment elements

Sensor values inherit from abstract class *TimedElement* which captures some properties of the state objects, such as their timestamp, the identifier of the simulation (to differentiate between separate executions), and information about when they were processed. Illumination is modeled as a real value with the level of brightness. This value depends on the brightness level of the bulbs and the outside light brightness from windows and doors. Moving objects are modeled using squares to simplify calculations, in particular, to check whether the moving object is within the vision range of a motion sensor.

Class Command models operations on physical devices. These operations can be executed manually by the persons present in the room—e.g., a person who decides to open the blind; or triggered automatically by another device—e.g., the motion sensor asks the light to start blinking because, in theory, the office should be empty, but it has detected a moving object in the room.

3.2 Synchronizing the physical and the digital environments

Digital Twin Systems are built with different goals in mind, and the way in which the synchronization between the DT and the PT needs to be achieved strongly depends on the particular *purpose* of the DTS. For example, in one scenario the goal of the DT is to act as an oracle that defines the expected behavior of the PT, checking for deviations in the behavior of the physical system. To achieve that goal, the DT emulates the behavior of the PT under the same environmental elements (sensed values and received commands), and produces a set of outputs, namely states and commands. Such outputs can be checked against the corresponding outputs of the PT for possible deviations by a monitoring service component (Muñoz et al. 2022). In this case, synchronization is achieved by continously updating the state of the sensor values and commands of the DT environment with those obtained from the environment of the physical system, and simulating its defined behavior accordingly. This scenario can also be used to predict future behaviors of the PT, allowing the realization of what-if analyses.

In a second scenario, the DTS aims to detect deviations in the physical system behavior and correct them if needed. In this case, the DT contains a model of the expected changes in the system environment under the execution of some commands. For example, the room's brightness is expected to increase during the day

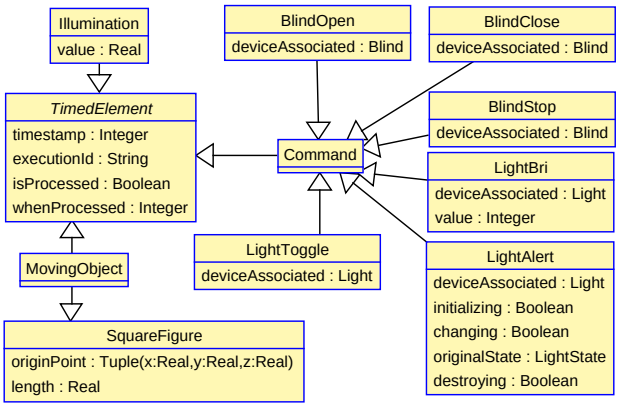


Figure 5: Class diagram with the model of the environment.

when the blinds are opened. Thus, after issuing a BlindOpen command, the DT can compare the brightness level in the room, as perceived by the system environment, with the one it expects. If significant differences are detected, the DT can assume several situations: (a) the device that controls the blind is not working properly; (b) the sensor that reads the level of brightness in the room is defective; or (c) it is completely dark outside (something very odd during the day time). The DT can try different actions to identify the causes of the mismatches. For example, by switching the light on and off and observing the effects on the room brightness, the DT can check whether the brightness sensor is working. Trying to close the blinds can help check if they were not fully open already. In this second scenario, the DT does not update the state of its proxies from the system environment to perform the simulation but compares its expected states with those received from the PT. Here, synchronization is achieved by acting on the PT's actuators to influence the PT's environment.

The communication mechanisms used between the proxies of the digital environment and the elements of the physical environment are the same as the communication mechanisms used between the two twins because we have just reified the elements of the environment (values and commands) as other objects of the UML model of the DT.

Storing the data from the physical environment into the DL. To capture the information from the physical environment and use it in the DTS, we store it in the DL. The way of capturing the environment data depends on whether it corresponds to *sensors* or *commands*. First, data coming from sensors (room brightness, weather conditions, presence of moving objects) is collected by the driver that sits between the PT and the DL. The driver uses the values captured by the corresponding sensors. In our case, the driver periodically polls the sensors, obtains the values of their readings, and stores these values in the database that implements the DL. The polling frequency is determined by a parameter that is configurable by the DTS user. Environmental conditions cannot be captured, but moving objects can. For this, we use mobile phones that report the position of the persons in the room. The driver uses Bluetooth to discover the phones, and ask them for their coordinates. A small App running on the phones is in charge of implementing this process. This way, in the DL, we do not only have the data produced by the brightness and motion sensors in the room, but also information about the moving objects. This is key for allowing the DT to check that the motion sensor is properly working. Similarly, information from the commands performed on the actuators—which is the second type of information produced by the environment—is also collected by the driver and stored in the database.

Updating the information of the digital twin environment. Another driver, connecting the DL and the DT, is in charge of transforming the relevant records in the database into the corresponding UML model objects. For this, the driver periodically polls the database at the specified frequency for new records of the required types. Once the records are transformed into the corresponding UML objects, the behavior of the driver depends on the type of goal of the DT, as discussed above.

Storing the data from the digital twin environment into the DL. There are situations in which the DT needs to control the PT and its environment. For example, a change in the digital system can trigger an operation in one of the physical devices; this is called an *event*. Thus, the DT may need to issue commands that represent operations to be performed on the physical actuators. For this, the DT creates a Command object targeted towards the PT. When the driver between the DT and the DL discovers the existence of such objects, it stores them in the DL. Then, the corresponding driver between the DL and the PT will be able to trigger the corresponding commands in the physical actuators.

In this way, we can manage each operation executed by either physical or digital devices and synchronize their corresponding counterparts accordingly. Timestamps in all records allow time synchronization, too.

In summary, with this approach, we can model and simulate the bi-directional interactions between the physical and the digital systems, including their environments. The simulation in the DTS is affected by

the state of the real environment, which is synchronized using the DL and the corresponding drivers. In addition, events triggered by the DT can result in operations being performed on the physical system. These operations usually have an impact on both the state of the physical system and its environment, which in turn update the state of the digital twin's environment.

4 EVALUATION AND DISCUSSION

To evaluate our proposal, this section describes the implementation we have developed for one particular DTS, and then discusses the main advantages and limitations that we have identified during its design, development and operation.

4.1 Evaluation using the running example system

First, we have demonstrated our approach with the running case study of the smart offices with sensors described in Sect. 2.2.1. We have developed simulations for several execution scenarios of the system.

For example, one of these scenarios starts at 8:30 in the morning before anyone is in the building. During the first 15 minutes, nobody enters any of the three offices. The only states that change during this period are the brightness of offices 2.15 and 2.07, which progressively increase. At 8:45, the first person (worker1) enters Office 2.15 and is detected by the room's motion sensor. The person wears a smartphone with our App, so both the positions of that person and the state of the motion sensor are recorded and stored in the DL, and then propagated to the DT by the DT driver. After entering the office, worker1 turns on the three lights in the room and sits at his desk. The DTS simulates these actions by creating several instances of MovingObject in the DL (and thus in the UML model of the DT) that represent the different positions of that person as he moves through the office, each one with a different timestamp. Likewise, the three performed commands on the lights are recorded and propagated to the DT. For instance, an object of class MovingObject will be created at 8:45:10 with position (7,4.2,0), representing the time worker1 entered the office. The position of the person at 8:45:18 is (2.3,1, 0), i.e., the location of the desk, which is out of the range of vision of the motion sensor and therefore this causes the sensor to stop detecting it. In addition, three commands are created to record the switching on of the three lights.

Then, worker1 opens up the blind of Office 2.15 at 8:46:03, which increases the brightness value of the room sensor from 5 to 106. In turn, this change of brightness triggers an event on the lights that creates, for each one, a command to turn them off. This event happens in the PT, and is propagated to the DT through the DL, thus synchronizing the physical and digital systems and their environments.

Finally, worker2 enters Office 2.07 at 8:48:22 and opens the room's blind at 8:48:31. This creates the corresponding BlindOpen command, which causes the illumination of office to increase. An excerpt of the object model that represents the final state of the DT at the end of the system execution (at 8:48:50) is shown in Fig. 6. All artifacts and models required for the simulation, including the SOIL file used to execute it, the USE model, and the plugin for connecting the simulation environment with the DL are available from a GitHub repository: https://anonymous.4open.science/r/environment-simulation-06EB/.

4.2 Discussion

During the design, development and operation of the Digital Twin System of the running example, we have identified several advantages and limitations of the proposal. They are discussed below.

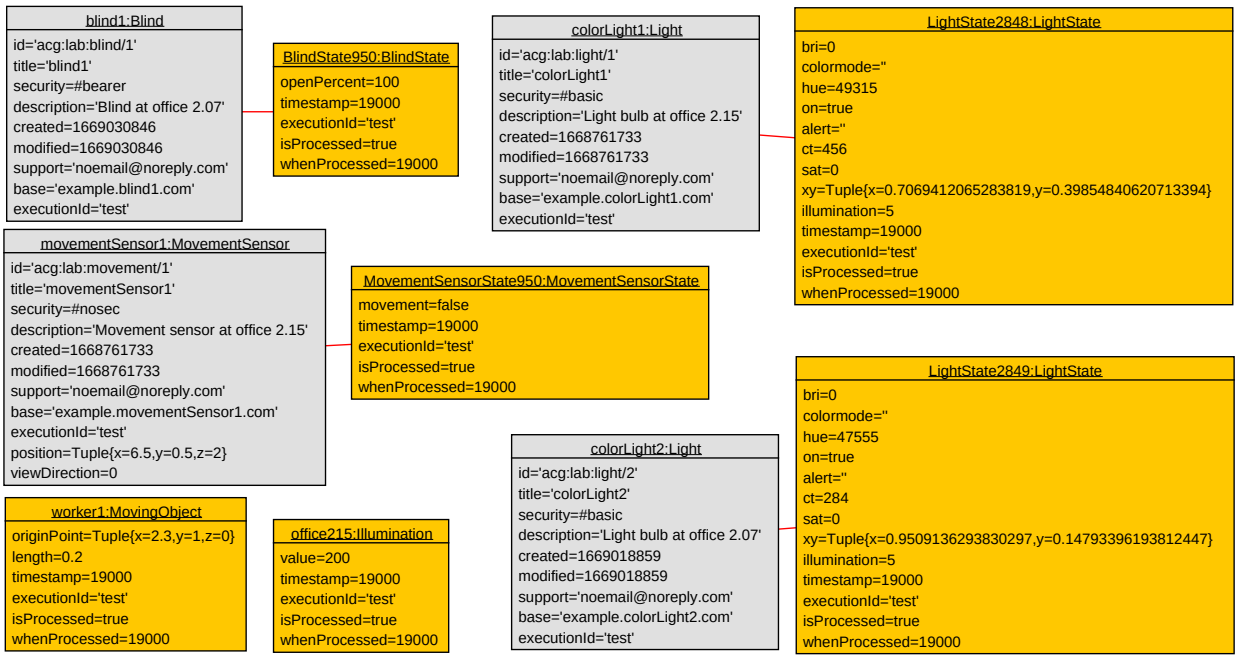


Figure 6: Excerpt of the final state of the system with one person, one blind, one motion sensor and two of the lights, plus the state of the luminosity of one of the offices.

First, we have been able to represent and establish a bi-directional connection between the physical system and its digital replica, able to connect and synchronize not only the states of the system elements but also those of their environments' elements. Using our connection, commands performed on the real system can be propagated to the DT and vice-versa: it is possible that the DT issues commands that are performed on the physical twin.

The use of high-level models has proved to be very effective because we were able to model the complete system and have a running digital twin of the running example in less than one week (including several iterations). Execution times are acceptable, although not for real-time simulation of high-demanding systems. Optimizations need to be implemented to achieve better simulation performance. For example, when running the system on a portable computer, the execution of the running example should be performed in batches of 15 seconds to avoid memory overload. Although the execution must be carried out in a controlled manner, the representation mode chosen for the simulations allows fast executions thanks to the USE modeling environment and the SOIL action language. In addition, the level of abstraction in the model descriptions facilitates their interpretation, while the use of models allows validating not only the simulations but also the real system executions.

The proposal has some limitations, too. For example, modeling certain environment elements can be really complex. Some weather conditions, for example, can be extremely difficult to model realistically due to their inherent complexity and manifold aspects. We have been able to capture their relevant properties through sensors, but simulating their behavior is a challenge. Similarly, modeling human behavior is complex because it is often unpredictable and even erratic. Moreover, acting on humans is difficult because there is no way to force them to perform actions. At best, we could send them recommendations or kindly ask them to perform tasks (Bilberg and Malik 2019). The dependency on the USE modeling environment and the SOIL action language could represent another limitation, although our proposal could be implemented using any UML tool with execution support for an action language. Additionally, the fact that USE is a non-commercial tool implies a performance far from what might be necessary for high-demand systems or

environments that would require real-time management of operations. For instance, USE has a limitation in the number of classes it can model, thus increasing the system's latency when adding more classes. In our approach, it is solved by only modeling the last snapshot of the information of each device. However, the evaluation lacks analysis and comparison of the proposal with different numbers of devices.

5 RELATED WORK

Although there is a vast literature on the simulation of the environment of systems, the list of works that deal with their synchronization with the real environment is rather reduced. First, some works such as (Bilberg and Malik 2019, Poursoltan et al. 2022) focus on systems where Cyber-Physical Systems interact with humans. They propose frameworks able to model the systems and convert these conceptual models into simulations, the environment being the combination of people, information and energy that interacts with the CPS. In our case, modeling the behavior of humans is a complex task, and interactions are defined at the interface level—e.g., displaying texts that the humans can read with the instructions about the tasks they have to perform, or turning lights on and off.

Feng et al. (2021) develop the Digital Twin of an incubator system. They propose a schema of bi-directional communication between the PT and the DT. Later, Feng et al. (2022) extend their work by implementing a self-adaptation MAPE-K loop. The authors discuss interesting issues such as self-calibration at runtime, being able to detect and correct errors on the fly. They use a model of the environment to calibrate their DT models, although only in one direction—i.e., they develop a *shadow model* of the environment (Liebenberg and Jarke 2020), and not a digital twin.

Paredis and Vangheluwe (2021) propose a Formalism and a Process Model (FTG+PM) to develop a DT for a line-following robot, with the goal of simulating the robot's performance. Its environment consists only of the line to be followed by the robot, but again it is used in only one direction, i.e., the flow of information is from the physical robot to the DT, not from the DT to the physical robot. Furthermore, our approach models more than one element of the environment to simulate the behavior of the CPSs. Their environment consists of the line that the robot has to follow, while our environment consists of MovingObjects, the Illumination, and the commands processed by the DTs.

Finally, Sensyml (Haris et al. 2019) is a simulation environment for CPS to test large-scale IoT scenarios. Sensyml allows users to add sensors using properties with random values, history values, or real values from data sources. These elements are important for systems deployed outdoors, e.g., in Smart Cities scenarios. However, they lack an explicit model of the environment which is synchronized with that of the DT, as in our case. We think our proposal could be beneficial to their solution since we allow tighter and more powerful integration with the elements of the environment within the DT System.

6 CONCLUSIONS

In this paper, we have shown how the environment of a digital twin can be modeled and kept in sync with the real one, by adding digital proxies of the relevant elements of the environment and connecting them using the same synchronization mechanisms used by the DTS. We have demonstrated our approach with a realistic case study of a smart room with sensors, using high-level software models.

The ideas presented in this paper can be continued in different directions. First, we would like to explore the automatic generation of the UML models of the IoT devices directly from their Thing Descriptions. We also need to work on the optimization of the performance of the simulations of the UML models, using more compact and efficient model representations and comparing the performance of simulations using a different amount of devices. Further and larger case studies should provide us with more information on the features and expressiveness of our approach, as well as on the limits of our proposal by comparing different

scenarios. In this sense, empirical validation and exercises with modellers and industrial users would help us to validate the applicability, usability and effectiveness of our proposal.

REFERENCES

Bilberg, A., and A. A. Malik. 2019. "Digital twin driven human-robot collaborative assembly". *CIRP Annals–Manufacturing Technology* vol. 68, pp. 499–502.

Bordeleau, F., B. Combemale, R. Eramo, M. v.d. Brand, and M. Wimmer. 2020. "Towards Model-Driven Digital Twin Engineering: Current Opportunities and Future Challenges". In *ICSMM'20*, pp. 43–54.

Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. 1996. *Pattern–Oriented Software Architecture: A System of Patterns*. John Wiley & Sons.

Charpenay, V., and S. Käbisch. 2020, 5. "On Modeling the Physical World as a Collection of Things: The W3C Thing Description Ontology". In *The Semantic Web*, Volume 12123, pp. 599–615, Springer.

Charpenay, V., S. Käbisch, and H. Kosch. 2016. "Introducing Thing Descriptions and Interactions: An Ontology for the Web of Things". In *Proc. of Semantic Web Technologies for the Internet of Things*, Volume 1783 of *CEUR Proceedings*, pp. 55–66. http://ceur-ws.org/Vol-1783/#paper-06.

Digital Twin Consortium 2021. "Glossary of Digital Twins". https://www.digitaltwinconsortium.org/glossary/index.htm.

Feng, H., C. Gomes, S. Gil, P. H. Mikkelsen, D. Tola, P. G. Larsen, and M. Sandberg. 2022. "Integration Of The Mape-K Loop In Digital Twins". In *Proc. of ANNSIM'22*, pp. 102–113.

Feng, H., C. Gomes, C. Thule, K. Lausdahl, A. Iosifidis, and P. G. Larsen. 2021. "Introduction to Digital Twin Engineering". In *Proc. of ANNSIM'21*, pp. 1–12.

Gogolla, M., F. Büttner, and M. Richters. 2007. "USE: A UML-Based Specification Environment for Validating UML and OCL". *Sci. Comput. Program.* vol. 69, pp. 27–34.

Grieves, M. 2014. "Digital Twin: Manufacturing Excellence through Virtual Factory Replication". White paper, Florida Institute of Technology, Florida,US.

Grieves, M., and J. Vickers. 2017. *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*, pp. 85–113. Springer.

Guinard, D., V. Trifa, and E. Wilde. 2010. "A resource oriented architecture for the web of things". pp. 1–8.

Hai, R., C. Quix, and M. Jarke. 2021. "Data lake concept and systems: a survey". *CoRR* vol. abs/2106.09592.

Haris, I., V. Bisanovic, B. Wally, T. Rausch, D. Ratasich, S. Dustdar, G. Kappel, and R. Grosu. 2019. "Sensyml: Simulation Environment for large-scale IoT Applications". In *IECON'19*, pp. 3024–3030.

IEEE Std 1003.1-2008 2016. *The Open Group Base Specifications. Issue 7, Sect. 4.16, Seconds Since the Epoch*.

INCOSE 2015. *Systems Engineering Handbook*. 4 ed. Wiley.

ISO 24765 2017. *Systems and software engineering – Vocabulary*. ISO/IEC/IEEE.

Liebenberg, M., and M. Jarke. 2020. "Information Systems Engineering with Digital Shadows: Concept and Case Studies". In *Advanced Information Systems Engineering*, pp. 70–84, Springer.

Muñoz, P., J. Troya, and A. Vallecillo. 2021. "Using UML and OCL Models to Realize High-Level Digital Twins". In *Proc. of ModDiT2021@MODELS'21*, pp. 212–220, Institute of Electrical and Electronics Engineers, Inc.

Muñoz, P., M. Wimmer, J. Troya, and A. Vallecillo. 2022. "Using trace alignments for measuring the similarity between a physical and its digital twin". In *Proc. of MODDIT@MODELS'22*, pp. 503–510, ACM.

Object Management Group 2014, February. *Object Constraint Language (OCL) Specification. Version 2.4*. OMG Document formal/2014-02-03.

Object Management Group 2015, March. *Unified Modeling Language (UML) Specification. Version 2.5*. OMG document formal/2015-03-01.

Paredis, R., and H. Vangheluwe. 2021. "Exploring a Digital Shadow Design Workflow by Means of a Line Following Robot Use-Case". In *Proc. of ANNSIM'21*, pp. 1–12.

Poursoltan, M., N. Pinède, B. Vallespir, and M. K. Traore. 2022. "A New Modeling Framework For Cyber-Physical And Human Systems". In *Proc. of ANNSIM'22*, pp. 90–101.

Robinson, I., J. Webber, and E. Eifrem. 2015. *Graph Databases: New Opportunities for Connected Data*. 2 ed. O'Reilly Media, Inc.

Sharma, A., E. Kosasih, J. Zhang, A. Brintrup, and A. Calinescu. 2020. "Digital Twins: State of the Art Theory and Practice, Challenges, and Open Research Questions". *CoRR* vol. abs/2011.02833.

Tao, F., H. Zhang, A. Liu, and A. Y. C. Nee. 2019. "Digital Twin in Industry: State-of-the-Art". *IEEE Trans. Ind. Informatics* vol. 15 (4), pp. 2405–2415.

Tao, F., M. Zhang, Y. Liu, and A. Nee. 2018. "Digital twin driven prognostics and health management for complex equipment". *CIRP Annals* vol. 67 (1), pp. 169–172.

AUTHOR BIOGRAPHIES

JUAN ALBERTO LLOPIS: is a Phd Student researching, under the national project TIN2017-83964-R, about Federation service discovery in the Web of Things after receiving a FPU grant (ref. FPU19/00727). His research focuses on: Internet of Things, the Web of Things and Smart Environments. His email address is jalbertollopis@ual.es

PAULA MUÑOZ is a PhD candidate at the University of Málaga. She graduated in Software Engineering from the University of Málaga in June 2019. Her research focuses on the precise specification and testing of software systems using models. Contact her at paulam@uma.es.

JAVIER CRIADO is an Associate Professor at the Department of Informatics, University of Almería, (UAL), Spain. He has participated in four national research projects and two regional research projects. His research interests include: UML design, Model-Driven Engineering, Component-Based Software Engineering, User Interfaces, Interoperability, Service-Oriented Architectures, Microservices and Web of Things. His email address is javi.criado@ual.es.

JAVIER TROYA is Associate Professor at the Universidad de Málaga, Spain. Before, he was Assistant Professor at the Universidad de Sevilla, Spain (2016-2020), and a post-doctoral researcher in the TU Wien, Austria (2013-2015). He obtained his International PhD with honors at the Universidad de Málaga, Spain (2013). His current research interests include MDE, Software Testing and Digital Twins. Contact him at jtroya@uma.es.

LUIS IRIBARNE is a Professor at the Department of Informatics, University of Almeria (Spain), and Head of Applied Computing Group. He has served as the Principal Investigator for seven R&D projects founded by the Spanish Ministry of Science and Technology. His main research interests include simulation, component-based software development, model-driven engineering, and software engineering. His email address is luis.iribarne@ual.es.

ANTONIO VALLECILLO is full Professor at the University of Málaga, where he leads the Atenea Research Group on Software and Systems Modeling. His main research interests include Open Distributed Processing, Model-based Engineering and Software Quality.