ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

WoTwins: Automatic Digital Twin Generator for the Web of Things

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Sciullo L., Trotta A., Montori F., Bononi L., Di Felice M. (2022). WoTwins: Automatic Digital Twin Generator for the Web of Things [10.1109/WoWMoM54355.2022.00095].

Availability:
This version is available at: https://hdl.handle.net/11585/894472 since: 2022-09-22

Published:
DOI: http://doi.org/10.1109/WoWMoM54355.2022.00095

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

13 November 2026

This is the final peer-reviewed accepted manuscript of:

L. Sciullo, A. Trotta, F. Montori, L. Bononi and M. Di Felice, "WoTwins: Automatic Digital Twin Generator for the Web of Things," 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Belfast, United Kingdom, 2022, pp. 607-612

The final published version is available online at ***https://dx.doi.org/10.1109/WoWMoM54355.2022.00095***

Rights / License:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/)

When citing, please refer to the published version.

WoTwins: Automatic Digital Twin Generator for the Web of Things

Luca Sciullo*, Angelo Trotta*, Federico Montori*†, Luciano Bononi*, Marco Di Felice*†,
* Department of Computer Science and Engineering, University of Bologna, Italy
† Advanced Research Center on Electronic Systems "Ercole De Castro", University of Bologna, Italy
Emails: {luca.sciullo, angelo.trotta5, federico.montori2, luciano.bononi, marco.difelice3}@unibo.it

***Abstract*—Digital Twins are crucial in Industry 4.0 IoT scenarios, as they replicate physical assets and enable important tasks such as predictive analytics, what-if scenarios and real time monitoring. The heterogeneity of IoT use cases usually makes the development of digital twins extremely application-specific as well as prone to interoperability issues. To overcome these two challenges, we propose WoTwins, a framework that, on one side, leverages the W3C Web of Things (WoT) standard to model data and entities, and, on the other side, generates automatically Digital Twins of existing Web Things by modeling their state space through a Markov Decision Process (MDP) graph and by predicting its behavior though Machine Learning techniques. We conduct experiments on a simulated use cases related to IoT robotics to evaluate our proposal.**

***Index Terms*—Internet of Things, W3C Web of Things, Digital Twin, Markov Decision Process, Machine Learning, Simulation**

I. Introduction

Digital Twins (DTs) are computational models that are coupled with a physical asset and evolve over time in order to replicate its structure, its behavior and its reaction to events as closely as possible [1]. DTs are extremely valuable in various contexts, however they find their natural applicability in predictive scenarios such as the Industry 4.0, where putting the physical asset at risk is an infeasible option. Even though the concept of a digital model of a physical component has long been used over time, DTs are often extremely specialized for the appliance that they are abstracting, creating a significant barrier for reusability [2]. Few research efforts are dedicated to offer a homogeneous approach. The problem is further exacerbated by the fragmentation of IoT devices which expose their data and services through different access mechanisms and, in most cases, in a non machine-understandable way. To this purpose, among the various standardization efforts proposed over the past decade, one of the most proficient is the Web of Things (WoT), which has been standardized by the W3C in 2020 [3]. The WoT seeks to abstract physical things (or entire networks) into single entities called Web Things (WTs), which expose their functionalities (also called *affordances* in [3]) by leveraging standard Web protocols and semantics. In addition, a number of IoT frameworks is now dedicated to embrace the concept of DT, but only as a way to abstract the physical layer through a set of standard interfaces. An example is Eclipse Ditto[1], which recently announced an integration with the WoT [4] and the Eclipse Arrowhead[2] which implies a set of core systems to support their deployment [5]. While these solutions offer tools for an efficient integration of heterogeneous DTs within a single application ecosystem, they still do not take into account the actual representation of the physical asset in terms of behavioral modeling.

To overcome the above mentioned issues, in this paper we propose WoTwins, a framework that enables the automatic generation of DTs by leveraging Markov Decision Processes (MDP) to model the general state space of a physical entity which has been mapped to a WT. WoTwins leverages the Thing Description (TD) to retrieve the capabilities of the monitored entity by abstracting from the application domain, and it is able to learn its behaviour under the form of state-action transitions. Then, it supports the automatic creation of DTs in the form of standardized WTs, so that the resulting component is ready to undergo a seamless on-boarding procedure and to enable predictive and what-if analysis. More in detail, this paper makes the following three contributions: (i) it proposes a framework that is able to autonomously generate a DT of a physical WT, by inferring its affordances through the TD and generating a clone, a Digital Web Twin (DWT), that embeds the behavioral model of the WT in input; (ii) it proposes multiple techniques to estimate the transition matrix of the MDP by monitoring the actual behaviour of the WT, including the application of Machine Learning (ML) techniques for those cases in which the state space has been only partially explored; (iii) it evaluates the proposed solution through a set of extensive experiments over a discrete scenario related to IoT robotics.

The rest of the paper is organized as follows: Section II presents the related works from literature, Section III details the mathematical modeling of DTs on which the proposed framework is based, while its architecture is outlined in Section IV, Section V provides implementation details, Section VI is about the evaluation of the framework and, finally, Section VII concludes the paper.

II. Related Works

Several application scenarios that employ DTs often cannot rely on supervised or heuristic approaches due to the nearly intractable multi-objective optimization problems that describe

[1] https://www.eclipse.org/ditto/index.html

[2] https://projects.eclipse.org/projects/iot.arrowhead

them [6]. In such cases, the major task is rather modeled as a Markov Decision Process (MDP) and Reinforcement Learning (RL) techniques are exploited in order to autonomously achieve the global KPIs. However, most of these solutions, like [7] and [8], are extremely tailored to a single use case, without a general approach that could be of use outside such field of application. As a matter of fact, the need for a DT generalizable model is envisioned as a core part of the path ahead for the very near future [9]. The work in [10] is a recent proposal that, similarly to ours, aims to move from one-off implementation of DTs to a generative paradigm, where formal mathematical models represent the physical asset and evolve through time according to a number of states and the sensor observations in the real world. More in detail, they propose a general framework based onto a graphical model inspired by partially observable Markov Chains, where digital states are estimated probabilistically, then they evaluate it over a practical use case, *i.e.* a UAV. Our work uses similar models, however it focuses on the aspect of data representation, as it leverages the WoT paradigm to share data and interact, making the issue of interoperability much less of a burden. In a similar research trend we can find other solutions, such as the work in [11], which explicitly refers to the concept of Probabilistic Digital Twins (PDTs), that are built on Bayesian probabilistic frameworks. The base concept is that DTs must rely on both *observations*, coming from sensing experience, as well as *assumptions*, as in fact it is never possible to observe every single property that affects the physical asset, therefore, some of the observations must be replaced by mathematical models in order to capture uncertainties.

A. W3C Web of Things

The concept of DT is often paired with the world of the IoT, in which the past decade has witnessed an ongoing fragmentation due to the emergence of heterogeneous technologies at all layers of the stack. The WoT proposes to overcome such a differentiation by extending legacy Web technologies to the IoT, bringing in solid standards and uniform interfaces [3].

More in particular, WoT abstracts physical entities into Web Things (WT), that are software components that act as a bridge between the physical asset and the outer world. They expose the entity's capabilities by means of a Thing Description (TD), a standard JSON-LD document that describes the WT metadata, its semantics and a number of *affordances*, i.e. the interactions that is is possible to have with such WT. More in particular, these include: (i) *properties*, which describe state variables that can be naturally fetched and/or modified, (ii) *actions*, used to call functions implemented by the WT, and (iii) *events*, that can trigger effects (such as generating data) onto the WT on top of defined conditions.

The concepts of WoT and DT have seldom been paired in literature, such as in [12], where a "Digital Twin WT" is envisioned as a virtual entity derived from the aggregation of multiple physical ones, or in [13], where the WoT is seen as the scaffolding to better grasp the intertwining of single components within structurally complex Systems-of-Systems, or again in [14], where DTs are WTs that are automatically generated and managed within a container middleware, however authors do not specify whether the behavior is also reproduced, as they focuses on semantic interactions.

As a conclusion, the context of DT is extremely novel and few research efforts have been directed towards automatic generation of DTs in a generalized context, at the same time none of them imply a standard, such as the W3C WoT, to describe how to interact with them.

III. Digital Twin Modeling

The WoTwins framework enables the automatic generation of a DT of a running WT in a domain-agnostic way, i.e. no prior knowledge about the semantic of the WT is required. The DT of the WT, referred as DWT in the following, is modeled through a Markov Decision Process (MDP). The latter includes the following components:

- A discrete list of states (S);
- A discrete list of actions (A);
- A set of probability distributions $T : S \times A \times S \to \mathbb{R}$, where $T(s, a, s')$ is the probability of moving from state s to state s' after executing action a, with $\sum_{s'} T(s, a, s') = 1$, $\forall s \in S, a \in A$.
- A reward function $R : S \times A \to \mathbb{R}$, where $R(s, a)$ is the value of the reward when executing action a in state s.

Due to the well structured definition of a WT through its TD, the mapping between the WT Affordances and the MDP components is straightforward. Indeed, let $P = \{p_1, p_2, ..., p_n\}$ be the properties of the WT exposed in its TD. We assume that each property is discrete and assumes only a limited set of values; let V_{p_i} be the set of valid values for property p_i. Similarly, let AW be the list of actions of the WT exposed in its TD. For ease of modeling, we assume that actions can take only discrete parameters in input and we assume also to have no information about the effect of this action on the behaviour of the WT, i.e. zero or multiple properties can change their value after the execution of an action. The MDP of the DWT is defined as follows:

- $S = V_{p_1} \times V_{p_2} \times \cdots \times V_{p_n}$, i.e. the set of states coincides with the set of possible configurations of the n properties. Let $id : S \to \mathbb{N}$ be a function numbering the states.
- $A = AW$, i.e., same actions of the WT.

The WoTwins framework builds the S and A sets by parsing the TD of the WT. The transition probability matrix (T) is estimated by monitoring the running behaviour of the WT, i.e., the property change as a consequence of an action invocation. More specifically, we consider three methodologies for the estimation of the T matrix:

- *Frequency* estimator. The probability $T(s, a, s')$ is estimated from the relative occurrences of the transitions. More formally:

$$T(s, a, s') = \frac{N(s, a, s')}{N(s, a)} \tag{1}$$

where $N(s, a)$ the number of times action a is executed from state s, and $N(s, a, s')$ the number of cases where

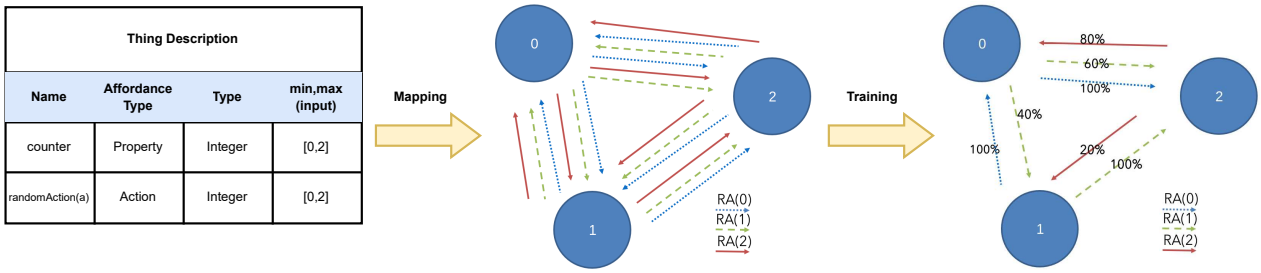


Fig. 1. The Figure shows the mapping between a WT and its DWT. The WoTwins framework takes in input a WT exposing one property (`counter`) and one action (`randomAction`). The latter takes as input one parameter (a), and changes the value of the property according to a probability distribution which depends on the current property value and on the a value. A fully connected MDP is generated by parsing the TD. After having monitored the WT for a while, the MDP is updated with the values of the estimated state-action transition probabilities.

the WT switches to state s'. We set $T(s,a,s') = \frac{1}{|S|}$ in case the state-action is unknown ($N(s,a) = 0$). In other words, we assume that the MDP graph is fully connected at startup, and that the WT may transit to any possible state after each action.

- *ML* estimator. The previous solution may fail when the MDP includes a large set of states, or when the WT visits only a limited set of it. For this reason, we investigate the application of Machine Learning (ML) techniques to predict the values of the $T(s,a,s')$ function. The current implementation leverages an Artificial Neural Network (ANN) as further detailed in Section V.
- *Hybrid* estimator. It combines the previous two methods. It applies the *Frequency* estimator in case action a in state s has been executed a minimum number of times, i.e., $N(s,a) > \phi$ where $\phi \in \mathbb{N}$ where ϕ is a user-defined threshold. The *ML* estimator is used otherwise.

A performance comparison between the two classes of estimators is reported in Section VI. Figure 1 shows an example of MDP generation from a WT that exposes one property *Counter* (with possible values 0, 1 and 2) and one action *randomAction* (with a parameter in range $\{0,1,2\}$). At startup, the WoTwins framework creates a fully connected MDP, where all states are reachable after an action execution. After a monitoring phase, the WoTwins framework estimates the values of T matrix: the subfigure on the left shows the resulting MDP where the transitions with zero probability have been pruned.
It is worth highlighting that the choice of the MDP introduces some implicit assumptions about the WT being modeled, like the fact that the property variables are discrete and that the current states contain all the information to predict the system evolution over time. However, we believe that such assumptions may fit well the characteristics of many IoT devices, specially edge ones, that expose limited functionalities of actuation and configuration. Finally, we remark that the reward function is currently not exploited by the WoTwins framework. However, the possibility to self-optimize the behaviour of a WT based on predicted reward values of the DWT will be considered as future study.

IV. Framework Architecture

The WoTwins architecture is micro-services oriented and each macro-functionality has been designed to be contained in a single module, as shown in Figure 2. We distinguish between front-end and back-end functionalities. In the first case, all the front-end functionalities are included in the *Dashboard module*, that offers an intuitive Graphical User Interface (GUI) to use the WoTwins framework. The latter includes the possibility to interact with the physical WTs and the relative DWTs and of controlling the training phase that is used to estimate the T function of the MDP. As better detailed in Figure 3, user starts the training phase during which the behaviour of the WT is observed, in terms of property value changes after the execution of each action. The user can decide when to stop the monitoring phase by spawning a new DWT with the current level of training. Finally, user can interact with the DWT enabling what-if analysis through a dedicated interface, for instance invoking actions on the DWT and observing in real-time the system evolution as sequence of next states. We highlight that the interaction with the DWT occurs through a WoT interface which is a clone of the TD of the physical WT; in other words, the fact that the user or a software client is interacting with a simulated entity rather than with its physical counterpart is equivalent in terms of API.

The back-end functionalities are split into four different modules: (i) the *Core Module*, (ii) *Twin Module*, (iii) *Thing Module*, (iv) *Learning Module*. The *Core module* is in charge of enabling the communication between the *Dashboard Module* and the other components, by translating the requests from the dashboard into proper commands to invoke on the other services. In addition, the *Core module* implements the WT monitoring phase by including mechanisms for collecting WT data which will be used during the training phase (mainly events of property changes and actions invocations). Finally, it acts as a proxy for interacting with the DWT once it has been generated. The *Thing Module* is responsible of communicating with the physical WT, while the *Twin module* includes all the functionalities for spawning a new DWT as well as for interacting with it. Finally, the *Learning Module* includes the algorithms for the estimation of the behaviour of the DWT,

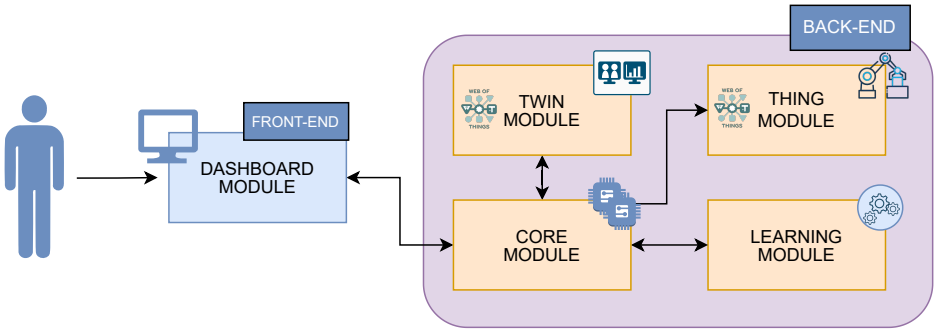


Fig. 2. The WoTwins architecture

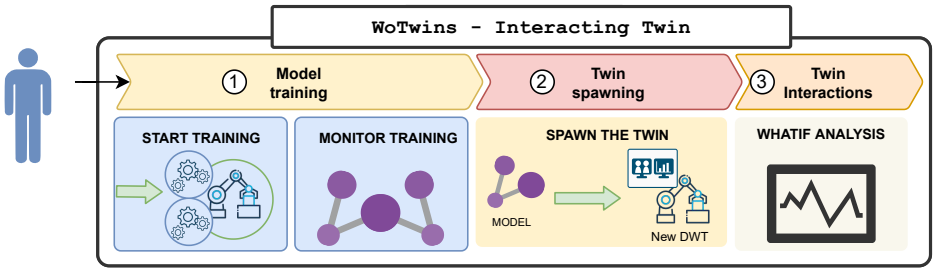


Fig. 3. The WoTwins interaction flow

modeled by its MDP as described before.

V. Implementation

The *Dashboard module* includes front-end functionalities and is constituted by a Web interface written using the Angular framework, and in particular the Apache Echarts[3] library for plotting the MDP graphs of each TW/DTW. The back-end functionalities consist of four services, as explained in Section IV; each of them has been implemented in Typescript using NodeJS as a runtime system. Both the *Twin module* and the *Thing module* are W3C WoT-enabled services, meaning that they implement a WT following the W3C WoT Scripting API indications[4]. More in detail, they run a *WoT Servient* that has been written using the Eclipse Node-wot[5], the official NodeJS WoT framework provided by the W3C. The *Core module* is based on the NestJS[6] framework and it covers a dual role: on the one side, it offers REST APIs to interact with the framework, on the other side it communicates with the W3C WoT services of the Twin and the Thing Modules through Node-wot as a client library. Finally, the *Learning Module* includes the MDP estimator algorithms mentioned in Section III. All the services have been containerized through docker and orchestrated through docker-swarm. The ML estimator has been implemented in Python through the Keras[7] framework. More in detail, the service keeps a history of the $\langle s, a, s' \rangle$ state transitions performed by the monitored WT. To be able to estimate the probability $T(s, a, s')$ also for a state s that has been never visited by the WT, the estimator attempts to learn the patterns of state transitions rather than the destination state. The pattern is defined as the state difference $id(s') - id(s)$, where $id(\cdot)$ is the state numbering function previously introduced in Section III. The ML model is a feed-forward ANN with a dense hidden layer and an output layer with softmax activation.

[3]https://echarts.apache.org/en/index.html
[4]https://www.w3.org/TR/wot-scripting-api/
[5]https://github.com/eclipse/thingweb.node-wot
[6]https://nestjs.com/
[7]https://keras.io

VI. Performance Evaluation

In this Section, we test and evaluate the WoTwins framework by demonstrating its ability of autonomous DWTs generation from a generic WT.

To this aim, we set up a simulation environment where a small ground rover (GR) moves on a fixed-size $N \times N$ grid map. The GR is characterized by its position $\langle x, y \rangle$ inside the grid map, i.e. $0 \leq x, y < N$ and movements are constrained to be inside the grid map. The GR can execute a single step movement in four directions: *up, down, left, right* or it can stay still. We assume a slotted time model $T = \{t_0, t_1, \dots\}$, with a fixed time slot duration. The simulation environment works as follows: at each time slot $t_k \in T$, the action `move` is executed and the GR advances to time slot t_{k+1} accordingly to the function parameter: *up, down, left, right, stop*. The overall rationale is the simulation of a ground rover that starts from its home charging station and moves around the map by executing its task and returning back home to recharge its batteries.

To make this simulation environment not trivial, we introduced a non-deterministic behaviour in the movement; more specifically $S_{x,y}$ is a binary variable for each cell $\langle x, y \rangle$ which models the presence of slippery material on the ground. Here,

$S_{x,y}$ = True if the cell is slippery and False otherwise. Moreover, we define p_{slip} as the probability of slip on a slippery cell. The slippery effect will change the destination cell after one movement, i.e. it will move the GR left or right after a `move(up)` or after a `move(down)` or it will move the GR up or down after a `move(left)` or after a `move(right)`, as depicted in Figure 4.

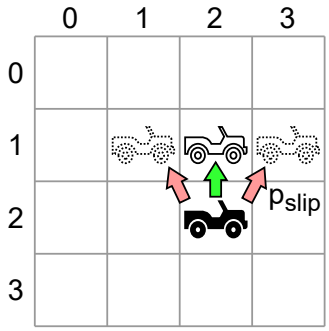


Fig. 4. The scenario environment used for the experiments with $N = 4$. The GR is placed at position $\langle 2, 2\rangle$ and the action `move(up)` is executed. The normal behaviour would move the GR to $\langle 2, 1\rangle$. If $S_{2,1}$ = True then with probability p_{slip} the rover will go to $\langle 3, 1\rangle$ or $\langle 1, 1\rangle$.

Then, we mapped the GR states and actions through a WT with the affordances listed in Table I.

TABLE I
THE AFFORDANCES OF THE GR.

Affordance Type	Value	Brief Description
Property	$x \in [0..N-1]$	The x-coordinate of the GR's position
Property	$y \in [0..N-1]$	The y-coordinate of the GR's position
Action	`move(d)`	Move the GR. Here d is the possible direction with admissible values: $d \in \{\text{up}, \text{down}, \text{left}, \text{right}, \text{stop}\}$

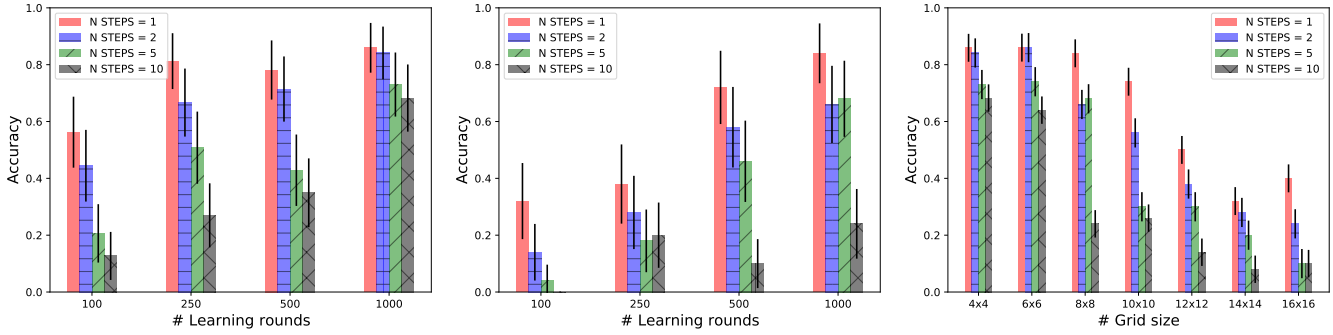
The evaluation process consists of three phases: (i) model training, (ii) DWT spawning, and (iii) DWT evaluation. In the first phase, we executed a variable number of *learning rounds* to train the matrix T (see Section III). To this aim, the GR moves randomly in the grid map starting from cell $\langle 0, 0\rangle$; all the movements and the change in the GR's states are observed by the *Monitor Training* module. The slippery cells, i.e. the cells $\langle x, y\rangle$ having $S_{x,y}$ = True, are placed randomly in the scenario with a percentage of 20% and with $p_{\text{slip}} = 0.1$. During the third phase, we evaluated the performance of the generated DWT in terms of average *accuracy* metric. The latter is the ratio of correct estimations of the DWT's final position compared with the real (simulated in our case) WT's position after N_STEPS movements.

We used the *Frequency* estimator methodology to update the transition probability matrix T. In Figures 5(a) and 5(b), we evaluated the evolution of the DWT after different *learning rounds*, for different values of N_STEPS, and for two different grid size: 4×4 in Figure 5(a) and 8×8 in Figure 5(b). It is easy to notice that the *learning rounds* and the N_STEPS parameters have a significant impact on the *accuracy* index. In fact, the longer the WT behaviour is observed, the more the MDP model is able to correctly emulate the behaviour of the original WT. The accuracy decreases with N_STEPS due to the longest path to emulate. However, in Figure 5(a) the *accuracy* tends to reach easily the stable point above 0.8, while in Figure 5(b) the increase is much slower due to the larger MDP size. The impact of the MDP size is shown in Figure 5(c) which shows the accuracy as a function of the grid size on the x-axis. The results confirm that the *Frequency* estimator method can be used only on small scenarios and, more in general, for WTs having a small number of state-action transitions. Figures 6(a) and 6(b) show the heatmaps of the explored grid cells during the learning phase after 1000 *learning rounds* for grid maps of 8×8 and 16×16, respectively. It's easy to notice that in the 16×16 case (Figure 6(b)) the GR explored only a small part of the whole scenario; as a consequence, the *Frequency* method is not able to estimate correctly the GR behaviour for the unexplored states.

Finally, Figure 6(c) compares the three different estimators of the transition matrix T described in Section III. On the x-axis we report the grid size while on the y-axis the average *accuracy* for N_STEPS=1. We considered different settings of the environment and of the testing methodology compared to the previous experiments: (i) the environment is still non deterministic however the slipping cells are placed on the diagonal and not randomly; (ii) during the training phase, the rover performs random actions of fixed length (10 actions) starting from an initial position with x, y both randomly chosen in the interval $\{0, \frac{N}{4}\}$. As a result of this setting, the GR may have not explored all possible state-action combinations in large-scale grid environments. Vice versa, during the testing phase, the initial position of the rover is randomly chosen among all the cells of the scenario. It is easy to see that the *Frequency* estimator achieves its highest accuracy for $N = 4$, however the performance drops when increasing the scenario size due to the consequential explosion of the number of states. At the same time, the Figure confirms the ability of the *ML* estimator to learn the state transition patterns of the rover in an effective way. The *Hybrid* estimator ($\phi = 3$) combines the benefits of both approaches: it uses the frequency measure for $N = 4$ and starts using the ANN as primary solution for $N > 10$.

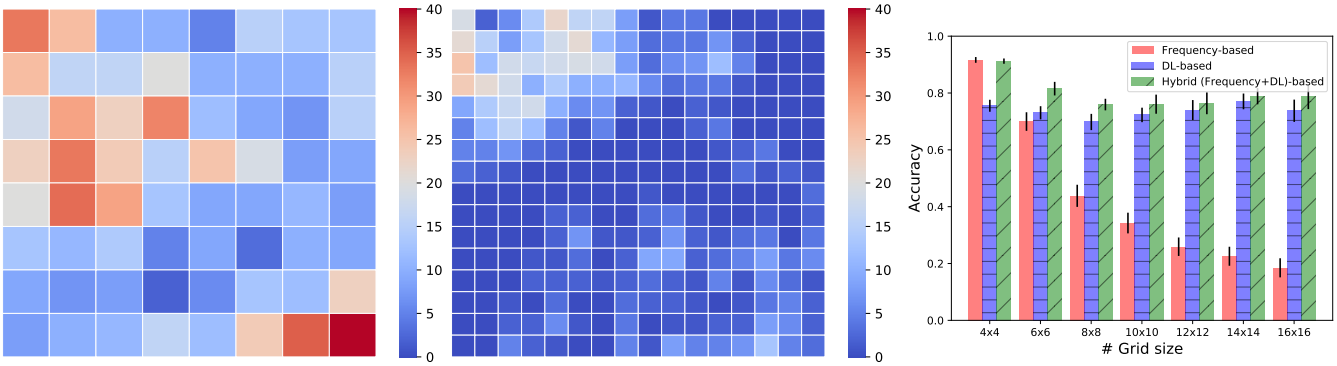
VII. CONCLUSION AND FUTURE WORKS

In this paper we proposed WoTwins, a framework that is able to generate DTs on demand in WoT scenarios. The W3C WoT standard guarantees the standardization of our approach, since if the physical asset is consistently described by a TD, then we are able to gain access to all its interfaces to the outer world. WoTwins maps the affordances of a WT into a Markov Decision Process (MDP), and it learns the behaviour of the WT by estimating the state-action transition probabilities through multiple techniques. In particular, we integrated a ML approach to learn the state transition function over unseen scenarios and validated the effectiveness of our proposal over a test use case. Future works include: the support for reward properties to enable WT self-optimization and the



(a) (b) (c)

Fig. 5. The *accuracy* index when varying the *learning round* and N_STEPS is shown in Figures 5(a) and 5(b) with a grid size of 4×4 and 8×8, respectively. The *accuracy* when varying the grid size and N_STEPS is shown in Figure 5(c).



(a) (b) (c)

Fig. 6. Figures 6(a) and 6(b) show the heatmaps of the visited cells during the learning phase in a grid map of 8×8 and 16×16, respectively. The *accuracy* index when varying the grid size and for three different estimators is shown in Figure 6(c).

extension to other modeling approaches to support continuous properties.

Acknowledgements

This work is supported by the FSE REACT EU - PON R&I 2014-2020 under the contract RTDA_GREEN (CUP J41B21012140007) and by the EU ECSEL Joint Undertaking under grant agreement No 826452 (Arrowhead Tools), within the EU Horizon 2020 research and innovation programme.

References

[1] AIAA Digital Engineering Integration Committee, "Digital twin: Definition & value. An AIAA and AIA position paper," *American Institute of Aeronautics and Astronautics (AIAA) and Aerospace Industries Association (AIA)*, 2020.

[2] S. A. Niederer, M. S. Sacks, M. Girolami, and K. Willcox, "Scaling digital twins from the artisanal to the industrial," *Nature Computational Science*, vol. 1, no. 5, pp. 313–320, 2021.

[3] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, and K. Kajimoto, "Web of Things (WoT) Architecture," W3C Recommendation, Apr. 2020, https://www.w3.org/TR/wot-architecture/.

[4] T. Jäckle, "W3C WoT (Web of Things) integration," https://www.eclipse.org/ditto/2022-03-03-wot-integration.html, accessed: 2022-03-04.

[5] G. Kulcsár, P. Varga, M. S. Tatara, F. Montori, M. A. Inigo, G. Urgese, and P. Azzoni, "Modeling an industrial revolution: How to manage large-scale, complex iot ecosystems?" in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021, pp. 896–901.

[6] Z. Huang, Y. Shen, J. Li, M. Fey, and C. Brecher, "A survey on ai-driven digital twins in industry 4.0: Smart manufacturing and advanced robotics," *Sensors*, vol. 21, no. 19, p. 6340, 2021.

[7] T. Wang, J. Cheng, Y. Yang, C. Esposito, H. Snoussi, and F. Tao, "Adaptive optimization method in digital twin conveyor systems via range-inspection control," *IEEE Transactions on Automation Science and Engineering*, 2020.

[8] F. Jaensch, A. Csiszar, A. Kienzlen, and A. Verl, "Reinforcement learning of material flow control logic using hardware-in-the-loop simulation," in *2018 First International Conference on Artificial Intelligence for Industries (AI4I)*. IEEE, 2018, pp. 77–80.

[9] R. Minerva, G. M. Lee, and N. Crespi, "Digital twin in the iot context: a survey on technical features, scenarios, and architectural models," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1785–1824, 2020.

[10] M. G. Kapteyn, J. V. Pretorius, and K. E. Willcox, "A probabilistic graphical model foundation for enabling predictive digital twins at scale," *Nature Computational Science*, vol. 1, no. 5, pp. 337–347, 2021.

[11] C. Agrell, K. R. Dahl, and A. Hafver, "Optimal sequential decision making with probabilistic digital twins," *arXiv preprint arXiv:2103.07405*, 2021.

[12] C. Aguzzi, L. Gigli, L. Sciullo, A. Trotta, F. Zonzini, L. De Marchi, M. Di Felice, A. Marzani, and T. S. Cinotti, "Modron: A scalable and interoperable web of things platform for structural health monitoring," in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2021, pp. 1–7.

[13] G. Privat, T. Coupaye, S. Bolle, and P. Raipin-Parvedy, "Wot graph as multiscale digital-twin for cyber-physical systems-of-systems," in *Proc., 2nd W3C Web of Things Workshop. Grenoble, France: Gilles Private Orange Labs Services*, 2019.

[14] S. Muralidharan, B. Yoo, and H. Ko, "Designing a semantic digital twin model for iot," in *2020 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2020, pp. 1–2.