Towards a Lightweight Model-driven Smart-city Digital Twin

Jean-Sébastien Sottet, Pierre Brimont, Christophe Feltus, Benjamin Gateau and Jean-Francois Merche
Luxembourg Institute of Science and Technology (LIST), 5 Avenue des Hauts Fourneaux, Esch-Alzette, Luxembourg

Keywords: Digital Twin, System Model Prototype, Executable Models, Flexible Modelling.

Abstract: In this article we consider the use of digital twin for representing open physical system, notably smart-cities. In this context everything is not necessarily defined or sensed; as a result a pure data-driven approach is not possible and should be completed by expert knowledge, hence through models. We propose an approach to design a model-driven digital twins MDDT that supports this context. It allows to 1/ quickly prototype digital twins, 2/ integrate various information from different kind of sources (diverse expertise, data, etc.) 3/ support evolution and enhancement all along the system's life-cycle.

1 INTRODUCTION

Amongst the many existing definitions of Digital twin (Barricelli et al., 2019) (DT) we propose to adopt this one: digital twin is "a digital replica of a physical system". The DT should behave similarly to the physical system if a similar event occurs to it. Conversely, action on the DT could have an impact on the physical system if it is designed as such [1].

Existing commercial approaches are proposing data-based DT and are focusing on data collection, for instance the generic platform *Microsoft Azur DT*[2]. Mainly dedicated to smart-city cases, it focuses on heterogeneous data collection and harmonization through a standardized single data model. Some initiatives try to add semantic (Singh et al., 2020) and additional information to the initial pure data-based DT. The current approaches mainly focuses on building DT from data obtained from the physical system only. As a result they are heavy weight and relying on complex infrastructure for data collection (e.g., *Fiware*[3]) and advances Machine Learning (ML) (Chakraborty and Adhikari, 2021) plus Data Analytics to build a digital twin; additional technologies are to be used to complete the view, like large 3D representations (see Unreal Engine for 3D digital twins[4]). However this view may remain only partial as all the knowledge about the physical system is not necessarily available through data/sensors.

From our perspective, we think that digital twins are rather hybrid entities (alike ideas expressed in (Lektauers et al., 2021)) composed by either human knowledge (technical blue prints, equations, etc.) or data information and machine inferred models, all being part of the definition of Model-driven Digital Twin (MDDT). The need of having more than only data-based approaches for digital twins construction is emphasized when twining beyond the classical industrial use-case (Raj and Surianarayanan, 2020) systems :

- which are more open like Smart Cities (Ketzler et al., 2020),
- which include a lot of assumptions of their borders (and behavior) with other systems,
- with part of the information which cannot be given by data collection (e.g., no sensors for this kind of data).

This is where models, as human construction for representing the reality, come into play.

Moreover, under this open system assumption, an approach would be to build incrementally a DT relying on model-driven engineering. In this approach we take an excerpt or a specific aspect of the physical system and first build a partial MDDT [5]. Then we extend it all along the discovery of the system facets, whilst improving the twin with the physical

[1] In smart-city cases the automated loop back from digital twin to the physical system is not always possible

[2] https://azure.microsoft.com/en-us/services/digital-twins/

[3] https://www.fiware.org/

[4] https://www.unrealengine.com/en-US/digital-twins

[5] Other definition of partial digital twin can be given like digital shadow or digital models as partial construction of the reality or abstraction of the reality (Ladj et al., 2021)

Sottet, J., Brimont, P., Feltus, C., Gateau, B. and Merche, J.
Towards a Lightweight Model-driven Smart-city Digital Twin.
DOI: 10.5220/0010906100003119
In *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, pages 320-327
ISBN: 978-989-758-550-0; ISSN: 2184-4348

system feedback (e.g., data). The excerpt can be a complex sub-system, like the electricity consumption of a quarter of a given city. It acts as a lens, helping to start with a concrete aspect of the physical system; it can be for instance later extended with the local electrical production of the quarter allowing to gradually build a more complete MDDT. This extension can be realised by combining different MDDT coming from other perspectives, thus calling for a federation of DT, close to the initial vision followed by Cambridge University's Center for the Digital Built Britain [6]. In this article, we promote the use of lightweight modeling support that would ease the development of such digital twins. Under such modelling support, parts of the model should also be easily replaced or extended when a more precise element is available: e.g., a refinement of the system behavior by analyzing actual data.

This paper is organised as follow: we first confront the MDE approach used with traditional Data Driven solutions to identify specific requirements for a lightweight Model processing platform, calibrated for smart-city open worlds. Then we expose those requirements through our approach to design a flexible MDDT, applied to a proof of concept in the context of smart city, and through snapshots of the currently developing prototype. We notably highlight the static and dynamic (action language) modelling aspects and, more importantly, the life-cycle of a MDDT, following the refinement on its relations to physical system: reducing the modeling gap.

2 THE REQUIREMENTS FOR OPEN MODEL-DRIVEN DIGITAL TWIN

As exposed in the introduction, we focus on open-world digital twins, with potentially few sensors and/or with a lot of human interventions. This is typically the context of a smart-city; where some of the information have been first designed (e.g., building information models) and where the knowledge evolves during its life-cycle. We set the requirements of a MDDT with the following capabilities:

- **Simulation Capabilities:** evaluate and store resilience of the physical system when extreme conditions occur.
- **Monitoring Capabilities:** direct connection to the physical twin: MDDT should be a mirror of what happens in the reality.
- **Alignment Capabilities:** learn from the physical system in order to be updated and to better fit with reality: coping with twin-physical system gap.

2.1 Need for Rapid Prototyping Digital Twins

Digital twin supporting systems can be complex like the one described in (Kaur et al., 2020): it encompasses a lot of domains and technologies like internet of things, edge and cloud computing architectures, artificial intelligence, machine learning, big data analysis, etc. However, for some specific situations, such a complex architecture is not always desirable, notably under the following circumstances:

- When considering a new domain, or at the early phases of designing a digital twin, it may be necessary to cope with very few data and complete it with models. In addition, even with a partial MDDT we should be able to validate the hypothesis we made in it.
- Within an existing digital twin, some aspects are not connected to sensors: They are done by human or not yet taken in account because of their novelty. Thus, no pre-existing and complete model of this physical part can be defined. Consequently, it requires to rapidly design an approximate but realistic model (according to human knowledge) in order to avoid blind spots when twining the system (for e.g., simulation or prediction purpose).
- the digital twin is also used as a simulation environment where it is important to be able to quickly test specific up to extreme scenario and prototype innovative ideas for the physical system (e.g., "what-if" scenario).
- Finally, we may have to deal with different digital twins, each completing the other and acting in a federated way; So we need to take into account that some are still missing in the twins constellation.

As a result, it is necessary to be able to quickly sketch models and model manipulations (e.g., model transformation, model processing) to build those digital twins from a human knowledge. It is also essential to have flexibility to discover the data and information coming from the physical system. Finally, it is also requisite to ensure interoperability with other twins at the border of the considered one like e.g., (Grace et al., 2016).

[6] https://www.cdbb.cam.ac.uk/

2.2 Need for Supporting Evolution and Flexibility

As stated in (Bordeleau et al., 2020), there is a strong need to interconnect models when dealing with MDDT. It is also necessary to interconnect those models with the physical system sensors and even sometimes to discover the sensors themselves (value, semantic). Thus, we should rely on a flexible modeling framework, enough permissive to relax modelling language constraints (Salay and Chechik, 2013) to allow those interconnections. Moreover, as their physical counter parts evolve, the underlying models of the MDTT should also be able to evolve easily. The models remains open to new event, sensors, modification, alteration of the physical system.

We should also take into account, as any designed model, that the MDDT may be at a certain point of time of its lifecycle, partially defined with potentially uncertainty. This uncertainty is part of the process of building a twin that mimics more effectively the reality.

Finally, we consider that, beyond data, the MDDT content is provided by different human profiles: different kind of users, different domain experts, etc. This is where flexibility also comes into play when capturing, as models, the knowledge provided by the experts. Notably, this flexibility is part of natural modelling approach (Zarwin et al., 2014) and is important to ease the capture of expert knowledge through models from a wide range of expertise domains.

2.3 Need for Executable Model and Action Languages

Some of the models composing a digital twin are by essence executable (e.g., machine learning models, scientific models), others are more static representation of the system, like UML models, or includes static representation of the dynamic, like statecharts. Defining behavior of the models part is crucial notably for supporting our prototyping approach of MDDT. Typically model execution is ensured through scripting (Peltonen and Selonen, 2001), generic action languages (e.g., AlF, FUML (Guermazi et al., 2015), xUMl,Scrall [7] etc.), business rules or through domain-specific action language (Mayerhofer et al., 2013).

In order to exploit the MDDT we should be able to execute it for:

- replaying past scenario and thus test and validate MDDT prototype,
- make simulation to see the physical system resilience,
- align itself with actual physical system state (as for monitoring goal).

Finally, to provide a coherent context and allow for continuity of execution (notably supporting the evolution), we should relate the executable part (i.e., dynamic) with their static counter-parts modeling element and we also need to provide backup solutions to the execution. For instance, an approximation function of required value for the execution is one of the possible backup solutions when some of the model elements are suppressed.

2.4 Need for Considering Data as Model

When we try to make sense of the data, i.e., not to consider the data as simple binary values, we already consider the data as an element of the model. To support this claim, we get inspired by (Bézivin, 2004): the execution of a given program is also considered as a model (in the MDE definition) if it is conform to a metamodel. By extension, we can say that the data handled by this program is also a model (since we can express a metamodel to which it conforms to). This idea finds also an echo in the field of the semantic web with the concept of sensor linked data (Patni et al., 2010). In addition, when dealing with data and sensors, it is important to consider the evolution through time (beyond discrete modelling) as the approach described in (Moawad et al., 2015).

Some current infrastructures or platforms for digital twining (e.g., Microsoft Azur DT) are already providing means to structure and gather IoT data from an interoperable way (Conde et al., 2021). Furthermore, like in Fiware, raw data is enhanced with meta-data to be exchanged by applications, modules, and services through a central ”context broker”. It is reasonable to think, by interfacing with such infrastructure, that a sensor producing data should have a metamodel, which allows for a seamless integration of the data into a MDDT. Under a simulation perspective, the data produced by the MDDT will turn to be directly expressed in the right metamodel (or data schema) allowing the expert a better understanding of what happened (in the simulation) as it is also expressed in the same format as real data. It can then help experts to compare and assess design alternative in the reality by simulation on the twin.

[7] https://github.com/modelint/scrall

Figure 1: Current experimental reloading bus station.

3 MODEL DRIVEN DIGITAL TWIN IN ACTION

As we have shown in the requirements section (see Section 2), it is necessary to have a modelling infrastructure that has the following functionalities: 1/ sketch easily a first MDDT either on its static parts (the models) or dynamic parts (actions) involving expert knowledge;// 2/ easily test and validate (e.g., using a scripting-like approach) the different models of knowledge available on a digital twin;// 3/ align this first MDDT with the physical system coping with reality gap and potential evolution of the physical system properties;// 4/ make this MDDT flexible enough to connect to the physical system and other potential twins.// In the following of this section we will address the functionalities 1, and partially 2 and 3. The fourth one is out of the scope of this position paper.

3.1 Case Study

Our case study addresses to electric public transportation. This is a first step into a smart-city-wise problem that can be extended to all the electricity production, traffic management, etc. Thus, we focus on an electric bus line which aims at optimising its electricity consumption and still offering a good quality of service. The electricity can be reloaded during the bus tour at local reloading stations see Figure 1. The buses should try to keep-up with their schedule with regards to e.g., potential work, traffic jam, meteorological conditions, while avoiding emptying their batteries and getting stuck.

In this case-study, a DT should be used to assess the resilience of such full electric buses on a given pathway simulating potential traffic jam, works and drastic change in weather conditions. It can also answer about the best stop to reload using only green energy, thus depending on weather condition and local energy consumption. Finally it can help in assessing if new bus stops or alternative tracks still remain compliant with the service quality requirements.

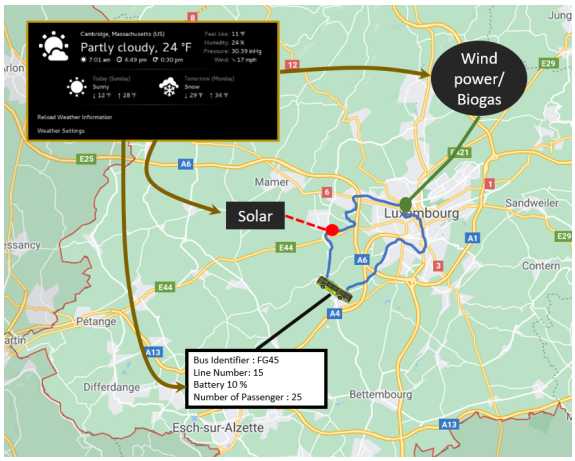

Figure 2: Electro-mobility with hybrid bus scenario including weather forecast, green electricity production.

3.2 Designing a First MDDT

During our prototyping phase of MDDT, we consider a first abstraction of the physical system, i.e., the bus electro-mobility case. So, we first design a digital model (among others) that serves as a basis of the MDDT and that as the ability to evolve as data collection and knowledge about the system progress. In this first step, several sources of information are to be taken into account:

1. we know (from the bus company experts) the bus stops, reloading stations and routes.
2. we know from past data and/or the expert knowledge, the potential impact on electricity consumption on each route portion (e.g., high climbing percentage, dedicated bus portion) between two stops.
3. we know, from the company designing the bus, an average consumption of the bus according to its load (passengers), and the weather (temperatures).
4. we can extrapolate based on expert feedback the impact of works, traffic jams, etc. on each route portion.

Within this context, we propose to use JSMF (Sottet and Biri, 2016), a JavaScript-based modeling framework, that aims at being flexible and make model-based application prototyping easy. In its current implementation JSMF partially supports the identified requirements on Section 2. We aims to rely on this modeling language to describe the static part of the model and we illustrate the dynamic part with JavaScript functions that manipulates JSMF Models. Those JSMF models are processed by elaborated specific languages on top of JavaScript.

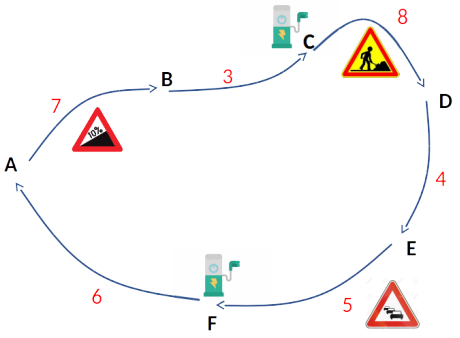

Figure 3: Visual of bus route modelling, including event like work or traffic jam, reloading stations are special kind of bus stop.

3.2.1 Static Information

JSMF can be directly used to represent elements 1 and 2 of the previous enumeration: they can be represented as a graph (see Figure 3) of routes (the edges) connecting the stops (the nodes). On each route the impact on the consumption is an attribute factor. Reloading stations are subclass of bus stops. We first establish a JSMF metamodel to support the definition of the buses and the pathway graph. The JSMF allows to define a metamodel in multiple (sub)modules and combines them together in a seamless way.

The bus is defined in Listing 1 as follow: a bus has an identifier, a line number, a battery level (from 0 to 100) and a number of passengers.

```
var Bus = Class.newInstance('Bus')
Bus.setAttribute('identifier', String
    )
Bus.setAttribute('lineNumber', Number
    )
Bus.setAttribute('batteryLevel', JSMF
    .Range(0,100)) //Batterie from 0
    to 100
Bus.setAttribute('passengers', Number
    )
```

Listing 1: Bus metamodel definition.

The metamodel for the stops and routes that forms a graph is defined in Listing 2. A *Stop* is defined by its name and its geographical coordinates (latitude and longitude). The *Route* class joins the stops using the *fromStop* and *toStop* relations. In this case study we only focus on a given line pathway: alternative route will necessary modify the original routing. The consumption attribute, defines the consumption factor to be applied to the given route according to expert knowledge with regards to slope and length.

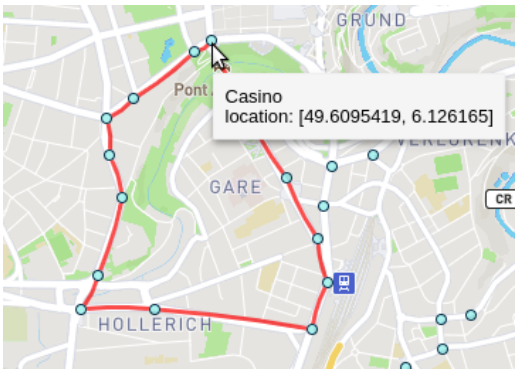

Figure 4: Map User Interface on top of JSMF.

```
var Stop = Class.newInstance('Stop')
Stop.setAttribute('name', String)
Stop.setAttribute('latitude', String)
Stop.setAttribute('longitude', String
    )

var Route = Class.newInstance('Route'
    )
Route.setAttribute('consumption',
    Number)
Route.setReference('toStop', Stop, 1,
    'fromRoute')
Route.setReference('fromStop', Stop,
    1, 'toRoute')

var Reloader = Class.newInstance('
    Reloader')
Reloader.superClasses = [Route]
```

Listing 2: Stops and route definition.

Then, in order to actually represent (and abstract) the physical system, a model conforms to this metamodel has to be instantiated. Here, multiple experts can come into play. For a better user experience with the model it is crucial to propose a simple yet efficient user interface on top of the JSMF model. In smart-city systems, a geographical representation (potentially including building in 3D) is the best way to interact with domain experts. It is easy to interface technologies like leaflet[8] and mapbox[9] with JSMF.

Through this interface (see Figure 4) the city bus responsible defines the bus stops (position, naming) as well as the actual routes between two stops. On top of each route, a consumption factor should be set. In the absence of actual data (e.g., starting a new line, using new kind of buses, etc.), it can be derived from the route path characteristic like the slope and length applying them a consumption factor on the battery (KwH/kilometers) and a mean consumption given by the bus constructor. It constitutes a first approxima-

[8]https://leafletjs.com/
[9]https://www.mapbox.com/

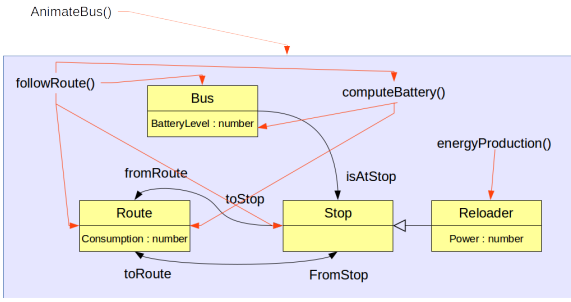

Figure 5: Dependencies relation between dynamic (functions) and metamodel elements.

tion for simulations and be update later with actual data.

An example of an instantiated metamodel is given in Listing 3, for sake of simplicity we have removed some of the attributes like the geographical coordinates.

```
var b1 = Bus.newInstance({identifier:
    'B1',lineNumber:11,batterieLevel
    :100})

var sA = Stop.newInstance({name:'A'})
var sB = Stop.newInstance({name:'B'})

var routeA = route.newInstance({
    consumption:7, fromStop:sA,
    toStop:sB})
```

Listing 3: Example of the model: bus b1 and the first stops and routes.

3.2.2 Dynamic Information

Once we have defined the static content of the MDDT, we need to provide, thanks to the diverse domain expertise, the way to animate our model and potentially provide simulation. We choose a function approach on top of our metamodel. The first demonstrator aims at simulating the buses route based on the previous model and metamodel. We have thus identified some of the function required to process or first (and partial) MDDT. As we work in a potentially evolving environment, at least the dependencies between the dynamic function and their corresponding metamodel elements should be modeled. The Figure 5 summarizes the dependencies relations.

The Listing 4 shows a simple JavaScript that manipulates the *Route*, *Bus* and *Stop* classes: since the bus is on duty, it follows the predefined path. We focus peculiarly on the *followRoute* function, which computes a new battery level at each followed route, time (and potentially time lost), and passengers entering and leaving the bus. It relies directly on the JSMF model with dot notation: going to the next destination is then simple as given by the model: b1.isAtStop[0].toRoute[0].toStop[0].

```
function animateBus() {
  while(b1.isOnDuty()) {
    var followed = followRoute(b1)
    b1.batteryLevel = followed.
    newBatteryLevel
    b1.isAtStop = followed.
    destination
    ...
}
function followRoute(b1) {
  var route = b1.isAtStop[0].toRoute
    [0]
  result.destination = route.toStop
    [0]
  result.newBatteryLevel =
    computeBattery(b1.batteryLevel,
    route.consumption)
  ...
```

Listing 4: Animation/Simulation of the model using a simple script.

The function that computes the (estimated) new battery level regarding the route can be given by a textual DSL that is representing a business rule or by an average consumption factor applied on the route consumption. At this stage of our prototype implementation, we have directly written the javascript function from general information estimated by experts values from a given formula.

For each segment of length (Le), the slope (Sl), the weather (We), the average weight (Pe) depending on the number of people (Pa) in the bus, possible roadworks (Wo) and traffic (Tr) are taken into account. Each parameter is expressed as a variable, allowing to approximate the emulated consumption. Knowing that an electric bus consumes on average 0.165 kW per kilometer, we defined the following equation:

$$Co = 0.165 * (\frac{Le}{Sl * We} + Wo + Tr)$$

The **Sl**ope is a coefficient in the interval $]0,7-1.3[$

$$Sl \in \mathbb{R}\ with\ 0.7 \leq Sl \leq 1.3$$

The Length of the segment is expressed in km $]3-10[$

$$Le \in \mathbb{R}\ with\ 3 \leq Le \leq 10$$

The **We**ather is a coefficient in the interval $]0-1]$

$$We \in \mathbb{R}\ with\ 0 < We \leq 1$$

Which gives in pseudo-JavaScript:
$We = Math.floor(Math.random() * 0.49) + 0;5$

People is the coefficient of the over-consumption depending on the total weight (average 70 kgs) of all the **Pa**ssengers including the driver [1-60]

$$Pa \in \mathbb{R}\ with\ 1 \leq Pa \leq 60, Pe = 70 * Pa * 1,01$$

Road Works are triggered by and event (Boolean) and a time length (min) [1,15], weighted by the coefficient of a stopped engine consumption

$$\forall E \neq 0 \; and \; t \in \mathbb{R} with 1 \leq t \leq 15 Wo = E * 1,02 * t$$

Which gives in pseudo-JavaScript:
$Wo = 1.02 * (Math.round(Math.random()) * Math.floor(Math.random() * 14) + 1$

Traffic is triggered by an event (Boolean) and a length (km) $[1,10]$, weighted by the coefficient of an idled engine consumption

$$\forall E \neq 0 \; and \; l \in \mathbb{R} \; with \; 1 \leq l \leq 10 Tr = E * 1.03 * l$$

Which gives in pseudo JavaScript:
$Tr = 1.03 * (Math.round(Math.random()) * Math.floor(Math.random() * 10) + 1$

We have very shortly illustrated that there are many kind of input in modeling such a system. As stated in (Bucchiarone et al., 2020), it can be very broad: scientists can provide advanced mathematical models, domain experts can deliver business rules, regulatory experts can restrict or scope with laws, etc. Managing different aspects in such a context requires a good coordination of models and modelling languages to ensure a coherent modelling landscape and common understanding of stakeholders under the DT challenges (Bjekovic et al., 2012). This is still a complete open challenge that we try to partially alleviate using a classical approach: a systematic modelling and dedicated languages / view for experts.

3.3 MDDT Lifecycle

We have thus designed our first backbone models on which we can play with to, e.g., simulate the physical system. This first partial MDDT has more or less consistency with the physical twin. It is assumed, in its early design time, to be an abstraction of the physical system. As a result, we can compare the execution of model with the similar model coming from the actual bus data (i.e., physical system), expressed in the same metamodel. From this comparison we can make evolution of metamodel, model (dynamic and static), up to a less partial digital twin. It is important to understand that we can infer refinement (Zolotas et al., 2019; Sottet and Biri, 2016) of the originally designed metamodel from collected information in the physical twin: notably new attributes identification.

Consequently, it is crucial to consider the maturation process of the MDDT under two axis: 1/a more precise view on physical system (e.g., knowing its border, better comprehension on requirements, etc.) 2/ the information collected on the physical knowledge which helps to improve the initial MDDT.

We can think of the following prototypical example situations from our case study:

- the revision of an attribute value: the route factor *consumption* attribute of the *Route* can be updated from the real physical system update.
- the revision of a dynamic aspect: the introduction of time.
- the revision of metamodel elements and related dynamic aspects taking into account the weather and its impact on battery energy consumption.
- change the *computeBattery()* function with regards to actual data and machine learning models. For instance, it can be executed using tensorflowjs (from a Keras model) inside the *computeBattery()* function as in Listing 5.

```
1 const model = await tf.
      loadLayersModel('localstorage://
      consumptionModel')
2 return model.predict(route,weather))
```

Listing 5: Loading a tensorflow model in Javascript.

More generally, we gradually fix the metamodels and models of the MDDT for both dynamic and static aspects. It requires the modelling framework supporting the MDDT to be flexible enough as one of both metamodels and models could evolve independently. Moreover, the dynamic part may fail to compute or simulate the model. Reasoning with the functions dependencies (whatever they are machine learning or written) is crucial to identify the blocking points. When one of this potential blocking point is identified (e.g., the model structure has changed, an attribute is not defined (either in the metamodel or the model)), we should then reason with uncertainty. Ideally, this is where approximation function based on past data can be used to provide, e.g., an absent attribute value. Similarly, to dynamic functions, approximation function should be modeled with regards to the backbone metamodel elements.

Moreover, it is important to warn the stakeholders and expose the modeling alternatives like (Famelis et al., 2012) when, e.g., a new weather concept is added, about the impact in terms of model element and computations. This feedback to stakeholders, decision making process is also an open challenge to deal with when considering MDDT.

4 CONCLUSIONS

This position paper presented our view on the design of a Digital Twin in a smart-city context. In such an open context, the system may not have clear boundary, this is especially true when discovering the area

or trying to apprehend an issue in the physical system. As a result, we propose to complete trendy data-driven approaches for DT with a flexible and integrative modelling approach.

Through this approach we defined MDDT prototypes; they are composed by executable models (defined in a static and dynamic ways) that are captured from a variety of inputs (e.g., different domain expertise, data consolidation, machine learning, etc). Those executable models allow notably to provide an easy to use simulation environment. In addition, MDDT prototypes should be able to evolve, being reinforced, some part being replace with data-driven ones (e.g., machine learning).

Accordingly, there are still a lot of challenges and future works to be tackled and implemented to fully support our vision. Let cite amongst other: ensuring the continuity during MDDT evolution, integration MDDT with a more data-driven approach (e.g., FiWare), multi-language and view for stakeholders involvement, and integration with other DT.

REFERENCES

Barricelli, B. R., Casiraghi, E., and Fogli, D. (2019). A survey on digital twin: definitions, characteristics, applications, and design implications. *IEEE access*, 7:167653–167671.

Bézivin, J. (2004). In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2):21–24.

Bjekovic, M., Proper, E., and Sottel, J.-S. (2012). Towards a coherent enterprise modelling landscape. In *5th Conf. on the Practice of Enterprise Modeling, Rostock, Germany, 2012*, pages 1–12. [Sl]: CEUR.

Bordeleau, F., Combemale, B., Eramo, R., van den Brand, M., and Wimmer, M. (2020). Towards model-driven digital twin engineering: Current opportunities and future challenges. In *Conf. on Systems Modelling and Management*, pages 43–54.

Bucchiarone, A., Cabot, J., Paige, R. F., and Pierantonio, A. (2020). Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling*, 19(1):5–13.

Chakraborty, S. and Adhikari, S. (2021). Machine learning based digital twin for dynamical systems with multiple time-scales. *Computers and Structures*, 243:106410.

Conde, J., Munoz-Arcentales, A., Alonso, A., Lopez-Pernas, S., and Salvachua, J. (2021). Modeling digital twin data and architecture: A building guide with fiware as enabling technology. *Internet Computing*.

Famelis, M., Salay, R., and Chechik, M. (2012). Partial models: Towards modeling and reasoning with uncertainty. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 573–583. IEEE.

Grace, P., Pickering, B., and Surridge, M. (2016). Model-driven interoperability: engineering heterogeneous iot systems. *Annals of telecommunications*.

Guermazi, S., Tatibouet, J., Cuccuru, A., Dhouib, S., Gérard, S., and Seidewitz, E. (2015). Executable modeling with fuml and alf in papyrus: Tooling and experiments. *strategies*, 11:12.

Kaur, M. J., Mishra, V. P., and Maheshwari, P. (2020). The convergence of digital twin, iot, and machine learning: transforming data into action. In *Digital twin technologies and smart cities*, pages 3–17. Springer.

Ketzler, B., Naserentin, V., Latino, F., Zangelidis, C., Thuvander, L., and Logg, A. (2020). Digital twins for cities: A state of the art review. *Built Environment*, 46(4):547–573.

Ladj, A., Wang, Z., Meski, O., Belkadi, F., Ritou, M., and Da Cunha, C. (2021). A knowledge-based digital shadow for machining industry in a digital twin perspective. *Manufacturing Systems*, 58:168–179.

Lektauers, A., Pecerska, J., Bolsakovs, V., Romanovs, A., Grabis, J., and TEILANS, A. (2021). A multi-model approach for simulation-based digital twin in resilient services. *WSEAS Trans. Syst. Control*, 16:133–145.

Mayerhofer, T., Langer, P., Wimmer, M., and Kappel, G. (2013). xmof: Executable dsmls based on fuml. In *International conference on software language engineering*, pages 56–75. Springer.

Moawad, A., Hartmann, T., Fouquet, F., Nain, G., Klein, J., and Le Traon, Y. (2015). Beyond discrete modeling: A continuous and efficient model for iot. In *Conf. Model Driven Engineering Languages and Systems (MODELS)*, pages 90–99.

Patni, H., Henson, C., and Sheth, A. (2010). Linked sensor data. In *2010 International Symposium on Collaborative Technologies and Systems*, pages 362–370. IEEE.

Peltonen, J. and Selonen, P. (2001). Processing uml models with visual scripts. In *Proceedings IEEE Symposia on Human-Centric Computing Languages and Environments (Cat. No. 01TH8587)*, pages 264–271. IEEE.

Raj, P. and Surianarayanan, C. (2020). Digital twin: the industry use cases. In *Advances in Computers*, volume 117, pages 285–320. Elsevier.

Salay, R. and Chechik, M. (2013). Supporting agility in mde through modeling language relaxation. In *XM 2013–Extreme Modeling Workshop*, page 21. Citeseer.

Singh, S., Shehab, E., Higgins, N., Fowler, K., Reynolds, D., Erkoyuncu, J. A., and Gadd, P. (2020). Data management for developing digital twin ontology model. *Proc. of the Institution of Mechanical Engineers*.

Sottet, J.-S. and Biri, N. (2016). Jsmf: a javascript flexible modelling framework. *FlexMDE@ MoDELS*, 1694:42–51.

Zarwin, Z., Bjekovic, M., Favre, M., Sottet, J.-S., and Proper, H. (2014). Natural modelling. *Journal of Object Technology*, 13:4–1.

Zolotas, A., Matragkas, N., Devlin, S., Kolovos, D. S., and Paige, R. F. (2019). Type inference in flexible model-driven engineering using classification algorithms. *Software & Systems Modeling*, 18(1):345–366.