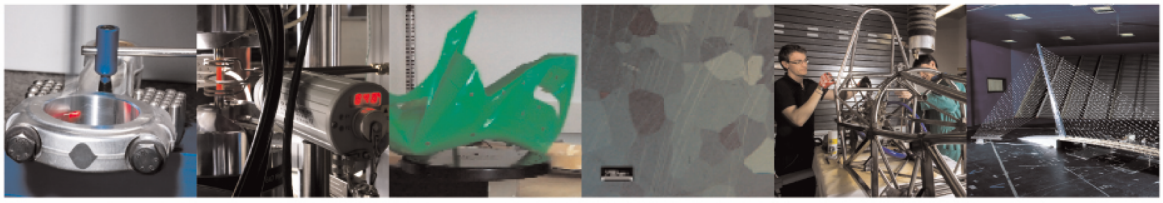
POLITECNICO
MILANO 1863

DIPARTIMENTO DI MECCANICA

mecc



Harmonising and integrating the Digital Twins multiverse: A paradigm and a toolset proposal

Cimino C.;Ferretti G.;Leva A.

This is a post-peer-review, pre-copyedit version of an article published in Computers in Industry. The final authenticated version is available online at:
https://doi.org/10.1016/j.compind.2021.103501

Harmonising and integrating the Digital Twins multiverse: a paradigm and a toolset proposal

ARTICLE INFO

Keywords:
Digital Twin
Industry 4.0
Advanced manufacturing
Cyber-Physical systems
Lifetime asset management.

Abstract

Digital Twins are of paramount relevance in the Industry 4.0 framework. However, the idea of Digital Twin has many different interpretations. These are tied to the intended use of a Digital Twin, thus to the viewpoint of the involved professionals (process designers, control specialists, managers, and so on). The said interpretations are often highly incompatible with one another, since they can involve as heterogeneous entities as a CAD drawing and a neural network. A convergence of Digital Twin interpretations is desirable, to take full profit of the contained knowledge. In this research we argue that this desired convergence cannot be found at the same abstraction level of the available Digital Twin interpretations, and calls for a higher one. We consequently propose a paradigm – that we name Digital Multiverse – to comprehend the major Digital Twin interpretations not only in the sense of data integration, which is the goal of promising complementary ideas like that of Asset Administration Shell, but also by establishing and enforcing consistency rules that involve both data and models. We also show some examples to support the usefulness and viability of our proposal.

1. Introduction

The Fourth Industrial Revolution (also called Industry 4.0, I40 for short) is strongly based on the interaction between the Physical World (PW) of machines and plants – in the broadest sense of the term – and a "Digital" World (DW). The DW can operate with the PW to establish Machine-to-Machine (M2M) communication and the (Industrial) Internet of Things or (I)IoT, implement automation, monitoring and online decision. The ultimate goal is that a PW and DW compound – often termed a Cyber-Physical System or CPS – can operate and maintain itself with no or minimal human intervention.

Besides hosting the Cyber part of a CPS, the DW can play a wider and more immaterial role. It can host various "representations" – for the moment, in the most general sense of the term – of the PW. These representations can serve as decision aids for design, maintenance, management, and the connected activities. Undertaking this role – in its numerous specialisations – means populating the DW with "Digital Twins" (DTs) of all or part of the CPS. For symmetry, the counterpart of a DT takes in this context the name of "Physical Twin" or PT.

As the term "representation" can have many interpretations, the same happens in the literature to the term "DT", see e.g. Sjarov *et al.* (2020), and even to the term "model". Discussing and organising this variety DT and model definitions is an important part of the presented research.

To start, we list some major DT interpretations induced by the viewpoint of the professional who conceives and employs a DT. People focused on equipment design tend to use "first-principle dynamic models", sometimes quite complex like FEM or CFD ones, as a *replica* of the PT part they are working on; here "model" means basically a system of equations. Control specialists prefer simpler "system-level models" (most frequently, block diagrams) to catch the relevant dynamics and tune the regulators; people in charge of performance assessment use similar "models" but may bring into play stochasticity and Monte Carlo techniques. Management, operations control and strategic decisions lie at more abstracted levels with respect to design and production control, and often rely mostly on analysing data, possibly with the aid of Artificial Intelligence (AI) and Machine Learning (ML); in this case"model" refers basically to some mechanism to interpret the said data, or more in general to take decisions (whence the terms "*data* model" and "*decision* model"). Apparently, should one ask each of the above professionals what a DT (or even a model) is, not only he/she would get different answers, but the entities mentioned in those answers (think e.g. of a CFD case, a continuous-time dynamic system, a Petri net, a 3D CAD *drawing*, a BIM[1] database, a set of historic trends, a neural network) would be radically heterogeneous from any mathematical and engineering standpoint.

* Support for this work came from a regional research and innovation project, that is not detailed at this stage to fully preserve the blind review process. This information, to insert here, is available in the submission data

[1] Building Information Model, a technology born to support facility management and more recently proposed also for production assets.

As our first claim, we argue that the set of Digital Twin interpretations is internally so incompatible, that no convergence or reconciliation can be found at the same abstraction level of those interpretations. As such, we are not adhering to any existing DT interpretation, nor introducing another one. We are proposing a higher abstraction level – that we name the "Digital Multiverse" (DM) paradigm for reasons explained in due course – to comprehend them all. In fact some "coordination" of the various DT interpretations has already been attempted, as a further evidence of how relevant the problem is. A notable result in this respect is the concept of Asset Administration Shell (AAS) typical of I40, see e.g. Wagner *et al.* (2017), while problems somehow analogous to ours are addressed e.g. in Moyne *et al.* (2020), that pairs DT interpretations to DT purposes, or in Villalonga *et al.* (2021), that coordinates different kinds of DTs. We are further discussing related work in Section 10, after presenting our approach, so as to better evidence the peculiarities of our research, and specifically the difference between "comprehending" in our sense and just "coordinating".

The paper is organised as follows. Section 2 details the research motivations and objectives, also clarifying the meaning we attribute to "models" and "data" in this context; Section 3 investigates the motivations for employing a DW, leading in Section 4 to proposing the DM paradigm; Section 5 deals with the necessity of introducing *consistency relations* that involve models and data *together*, and are a novelty of our paradigm; Section 7 identifies the enabling technologies for implementing the DM paradigm, and outlines their coordinated application toward the realisation of a DM toolset; Sections 8 and 9 show two application examples; Section 10 deals with related work, and finally Section 11 draws some conclusion, also outlining future research and application work.

2. Motivation and Research Objective

Anticipating from the literature review that we provide in Section 10, after presenting our ideas in full, we can broadly classify DT interpretations as follows.

- *According to usage*,
  - Offline, e.g. to analyse design alternatives without requiring any connection to the PW;
  - Online, e.g. for monitoring and/or maintenance purposes;
  - Online/offline combination, when an analysis in the DW is used to take (near) real-time decisions that are subsequently applied to the PW.
- *According to nature*,
  - Based on model(s) in the broadest sense of the term,
  - Based on data, either acquired from the PW or synthetic,
  - Based on both model(s) and data, as well as on input from experts.

No matter the usage, a DT invariantly serves to take decisions, and this leads us to a first problem. Those decisions ultimately impact the PW, more or less directly, and since in the PW decisions propagate by the laws of physics, each decision comes to alter a part of the PW that can be difficult to delimit *a priori*. If in the DW there is no equivalent for the laws of physics to propagate the effects of decisions, inconsistencies can arise. In this respect we notice that some literature DTs mostly observe the PT without implementing any action on it, while the same is not true for other DTs. As will emerge in the following, the paradigm we propose aims to suit both ideas.

As a second and connected problem, the above inconsistencies can be with the PW (which could in principle be revealed by measurements) or just internal to the DW. For example, a DT-originated decision on some part of the PW can invalidate some hypothesis for some other DT focused on "something else" in the overall described asset. In the absence of a means to spot this immediately, nobody will notice, and when that second DT is used for some other analysis, wrong results will be obtained.

In a large project, the above problems can lead to inefficiency – if not just errors – in the exploitation of the available knowledge; this provides the main motivation for our research. However, countermeasures are hard to take given the heterogeneous nature of DTs; how to devise a suitable paradigm for that purpose is the main question we address.

3. The Digital World

The DW contains entities that we divide into *models* and *data*. The difference is that models can *run* – in the broadest sense – fed with data, and produce other data—here too in broad sense, including e.g. decisions. We would like to stress here that for us a "model" does not necessarily mean "simulator"; a model could e.g. just elaborate some runtime data when a "warning" threshold is trespassed to extrapolate when an "alarm" will occur. We understand anyway that the above distinction is a bit *tranchant* and would not fit *as is* in many DT taxonomies. The non intersecting groups of "runnable" and "not runnable" DW entities serve only for introducing our proposal.

Notwithstanding a huge research *corpus*, we bear to state that the question "why is a DW desired?" has been hardly ever considered so generally. For our purpose we rephrase it as "what are the characteristics of the DW that the PW does not offer?". We answer that in the DW, contrary – this is fundamental – to the PW, the statements below hold true. Notice that some naturally pertain to a (simulation) model, others to data, some – not all – to both.

Everything is measurable. A simulation can e.g. determine the entire temperature field inside a part; in the PW this cannot be measured.

Every experiment is repeatable. A simulation can be re-run as many times as desired, for example to compare design solutions; resetting the PW to carry out two physical experiments with exactly the same initial condition is hardly ever – if ever – possible.

No information is ever lost. As everything is measured everything can be logged, and the entire set of available data – coming from simulations, experiments or elsewhere – can be queried for any desired purpose. In the PW, if a measurement was not taken it is just lost—and as said, there is no "redoing *exactly that* experiment".

Time is not an independent variable. In simulation, time can run faster than in the PW if enough computational power is available. The pace of simulated time is dictated by the complexity of the analysis and/or the required level of detail. Having a model running in real time, synchronised with the PW, is just one of the possible cases.

Time is reversible. In principle (we do not discuss mathematical intricacies here) a model can be inverted to start from the outcome and calculate the causes, even running time backwards. Also the entire state of a project can be orderly rewound to any instant in the past, to re-analyse some facts previously overlooked, and then set back to the current condition without compromising its integrity.

Phenomena can be made to appear or disappear. In the PW physics is physics, while in the DW one can decide that some phenomenon (e.g., oil compressibility) is irrelevant for a given study, and "fake" only for that study a world where it does not occur.

The DW has the nature of a multiverse. This is by far the most important point. In the PW there is *one* reality. In some cases different prototypes can be built in parallel to test design alternatives, but for cost and management reasons the goal is to keep them to a minimum (incidentally, simulation models are also called "virtual prototypes" Wang (2002)). However, when considering large production assets, especially if subject to frequent reconfigurations, the above remark is of scarce relevance. The key point is conversely that *in the DW parallel realities always co-exist*. This happens obviously when the PT is being designed but also during the PT operational life, for example to pre-test a reconfiguration, to investigate the causes for a malfunction, or to generate synthetic data for pre-tuning some data-based online management tool.

The multiverse nature of the DW is central to our research. We argue that the mentioned problems in correlating the various DT interpretations come primarily from not recognising that nature, and as a connected issue, from not dealing with data and models (in our sense) jointly.

We illustrate the last statement with an example. Suppose that at design stage, a team of analysts are using (simulation) models to test alternatives for different parts of the PT. Any such model relies on hypotheses about the boundary conditions presented to it by the rest of the PT. But in the rest of the PT alternatives are being studied as well. Hence analyst A could address his/her part of the project with wrong ideas about the part addressed by analyst B, which we express by saying that "their universes are not consistent". Also, the universes of A and B may differ not just by some parameter, but by the entire construction of some piece of equipment, which requires to ensure what we call "consistency of data and models".

In the absence of a paradigm for treating data and models jointly, a means to prevent inconsistencies is to "verticalise" the DW by activity purpose. For example, design activities can be carried out mostly using simulation models, and a team could set up internal procedures to design the components of an asset in a certain order, so that "blocking" decisions be always taken before the influenced ones. Of course, however, such procedures shall be specific to a certain

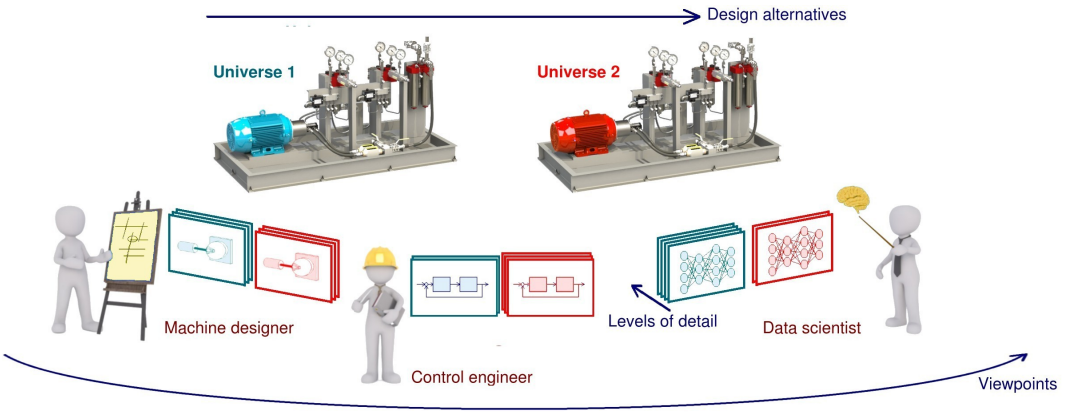


Figure 1: The multiverse nature of a Digital Meta Twin – each universe, each viewpoint (and possibly each level of detail) results in a DT along the corresponding interpretation; each DT is surrounded by a frame, and the frame colours distinguish the various parallel universes that co-exist in the DMT.

type of design, and to create them, one has to know *in advance* which decisions are blocking for others. This can be an issue in the case of a totally new design. On the contrary, maintenance activities can mostly use data, and possibly exploit those data also to keep the simulation models "current" with the PT.

We are not questioning the engineering and scientific value of such verticalised, "purposed" DTs, nor the value of asset-level coordinating approaches like the mentioned AAS. We just observe that if the DW is meant to offer all the characteristics above, apparently no DT interpretation proposed so far can be the unique choice. First, simulation must be possible at any stage of the design. We thus support the AAS idea that interpreting "twin" just as simulator is reductive. However, we also stress that besides hosting other types of "models" than simulators, a successful DW paradigm should allow for reversible time and multiple co-existing levels of detail. For example, referring to Figure 2 in Wagner *et al.* (2017), feedback in the design process should be possible not only from the final outcome but also for each stage of the design/operation and part of the asset, *even if the rest of the asset is not designed yet and only the boundary conditions it has to provide are stipulated*. Second, in the management of an asset there can be one "twin" aligned with the PT, but there are also a lot of other studies, alternatives, designs – in one word, parallel universes – to store in the knowledge base and possibly query in an orderly manner. Third, not only data *but also models* have to be "in accordance" with one another – in a sense already suggested but still to be specified – and the said consistency needs enforcing in a way that accounts for the inherently multiverse nature of the DW.

Accounting for this multiverse nature, together with introducing consistency relations involving both data and models, are distinctive characteristics of our approach. These provide a necessary complement for related research, as just sketched with AAS as an example. In addition, comprehending heterogeneous DTs under one paradigm is beneficial in terms of DT reusability, extensibility, and interchangeability. Studying the said multiverse nature is the subject of the following section.

4. Digital Twins as views on Multiverses

In our paradigm the fundamental entity is a "Digital Meta-Twin" (DMT). A DMT is associated to a unique PT and has the nature of a Digital Multiverse, whence the "DM" name. Each universe in a DMT corresponds to one set of choices for the available alternatives. The qualifying property of a DMT is that, when observed from a certain viewpoint, it appears as a DT of the corresponding PT in the sense that corresponds to that viewpoint.

With reference to Figure 1, for example, three analysts with different viewpoints, when looking at a certain universe (1 or 2, blue or red items and frames), see as "DT" what they expect. The machine designer sees CAD drawings, FEM models and the like, the control engineer block diagrams, the data scientists AI-based decision models, and so forth. Obviously, the same applies for each universe and detail level an analyst wishes to consider.

An analyst could also look at the DMT by viewpoint instead of by universe, to see all or a set of alternatives (universes) from that viewpoint, such as the control outcomes for the motor choices that distinguish universes 1 and 2 in Figure 1. The analyst can also optionally set a level of detail (e.g., no gear compliance) to further restrict the scope

if desided. Viewpoints can be defined by process or function, such as

- Design/reconfiguration (to set up the system and its control for the present purpose),
- Operation/scheduling (to allocate and manage resources and inventory at operation time),
- Maintenance (to ensure continuity of operation and health of the equipment),
- Interfacing (to properly design and manage connections with the outside world like suppliers),
- Financial (to forecast/assess/analyse technical costs and revenues).

Equivalently, one could define viewpoints based on roles such as machine/plant designer, control engineer, data scientist, facility manager, maintenance manager, and so forth. One may also introduce strategy-related viewpoints, for completeness, but this touches aspects that are not in the scope of this work.

Most important, a DMT is bound by convenient consistency rules, so that the DTs that it presents under the various viewpoints cannot contain any contradiction. The DMT level is the anticipated "more abstracted" one to reconcile DT interpretations, and defining what "consistency" means in this context is central to the research.

To approach this question, we need to investigate the *origins* of the multiverse nature of DMTs. One is apparently given by *design alternatives* for a single entity in the PT. However, within one of the universes in a DMT, the same PT entity (not a design alternative for it, that would be in another universe) can be described with different *levels of detail* and *types of representation*. The level of detail refers e.g. to accounting or not for some phenomenon, using a coarser or finer spatial discretisation, and so forth (incidentally, a dataset can have a coarser or finer spatial and/or temporal granularity as well). The type of representation refers to different modelling approaches. For example, a CNC can be represented with differential equations and electro-mechanical connections to handle the parts to machine and power, or with a "queue and server" model having part flows as numeric inputs and outputs. It appears then that various representation types for the same entity may not share the interface: some (e.g., the queue model) can be based not on physical but on measured/historical data (e.g., to predict future machining times), as well as electric components can be represented in the time or the phasor domain.

Summing up, the introduction of DM and DMTs paves the way to the desired convergence of data and models. A model takes data (a few physical parameters and inputs or a huge set of historical data, there is no conceptual difference), runs (alone or synchronised with the PT, there is no difference either) and produces data, to be stored for further models to run and in some cases also used immediately as for alarms. Running a CFD simulation or deep machine learning fall equally in this framework.

Our main problem is not yet solved, however. The remaining major issue is that the way data is elaborated by models affects the definition of the multiverse axes, typically as for detail level and representation type, because the way a model elaborates data is part of its behaviour. If this were not true, one could consider models the same as procedures (or programs) operating on a data base. It would make sense, for such procedures, to talk about the presence or absence of bugs, and if a procedure could be guaranteed bug-free (for example by formal analysis) then ensuring the consistency of data (a matter well studied in the database field) would be enough to ensure that no inconsistency could arise in a DM.

But as said, operating on the universes in a DMT very often means creating new models (continuing the database analogy, new programs). And even if these are all internally correct, their correctness *within the DMT* also depends on their relations with other models in the same DMT. For example, a modification in a "complex" model could invalidate a "simpler" one, meaning that the latter could still be *internally* correct, but if run, produce results that are inconsistent with the rest of the DMT.

A key point in this respect is that detecting such inconsistencies based on the produced data is problematic, and potentially very dangerous. Sticking to the example, the simple model could still produce results that are "correct enough" in some operating range, or not show its divergence from the complex one if not for some combination of inputs. This is why the problem may remain unnoticed long enough for taking several wrong decisions, with an impact of *a priori* unknown extent.

As such, within a DMT not only data, but also models must be bound by consistency relations. A DMT needs to be a Model and Data Base (MDB) where the two kinds of entity co-exist on a complete equality basis.

As a final remark, for some DT interpretations the correspondence with the PT (sometimes up to real-time connection) is required. In fact, one could divide the life of a DMT in two phases, before and after the PT starts to exist. Practically, these correspond to design and operation of the PT. We know that this is quite idealistic, as DTs need often

constructing when the PT already exists, but for the moment we stick to this view for clarity; we shall spend some words about the problem just mentioned later on.

In our paradigm, it suffices to state that when the PT starts existing, one of the universes in its DM has to keep "synchronised" with the PT. This means that any conclusion drawn from any DT offered by the DMT in that universe must agree with the same conclusion as drawn on the PT. We call this "universe P", with intuitive meaning. In the design phase universe P has no physical counterpart yet, but can already be interpreted as the "master" for the project.

5. Consistency relations in Digital Meta Twins viewed as Model and Data Bases

When models are created and manipulated through the life of a project, *relations* between them are implicitly introduced. If these are not guaranteed to hold true in the presence of different analysts concurrently modifying the model base, inconsistencies will eventually arise. To prevent this, a fundamental and qualifying characteristic of the DM paradigm is that relations among models (i) shall be instated explicitly in the knowledge base and not just "exist in the mind of the analysis", and (ii) shall be treated the same as relations among data. Furthermore, relations can exist among models and data as well.

Having relations that involve data and models equally is a unique peculiarity of our approach, making it different (for example and to prevent maybe the most likely confusion) from product/asset/project life cycle management ones. These see models as applications that consume and produce data, and the idea of model-data relation has no place in them. Without claiming exhaustiveness, we now list and briefly comment a few such possible relations.

Linearisation means that model A linearises model B around operating point O, and requires no explanation.

Phenomena neglection/simplification occurs when model A is a simplification of model B in that it neglects the set of phenomena P. Examples are hydraulic models accounting or not for fluid compressibility, mechanical ones considering compliance or not, and so forth. Alternatively, model A could provide for phenomena P a "simpler" description, for example linear elasticity instead of complicated stress/strain equations.

External interface equivalence indicates that model A is a simplified equivalent of model B seen at the terminal set S, and possibly on operating point O. For example, a single – in general nonlinear – hydraulic impedance between two flanges, can substitute a complicated circuit near some operating point. Analogously, a large model (e.g. a distributed-parameter one with fine spatial discretisation) can undergo order reduction, projection techniques, modal analysis or other procedures to yield another model with the same external interface but a lower dimension for the state space.

Description in other domain means that model A is a representation of model B in the modelling domain D (other than the one of A). For example, an AC electric component can be modelled in the time or in the phasor domain; a given machine can be described in the electro-mechanical domain or as a server in a queue network, representing its service (machining) time distribution.

Dependence on generated data is when some model parameters come from elaborations (whence the word "generated") of some data; no examples are needed, the reader certainly has some in mind. Another typical case – in fact just a variation but worth pointing out whatsoever – is when model B is data-based in nature, and the data to obtain it come from simulations of model A on a certain coverage C as for initial state and applied *stimuli*. An example could be a neural network trained with synthetic data that cover a conveniently qualified set of cases.

The set of relations above can be expanded, as said, and in addition, between two models more than one relation can exist at the same time. For example, the queue model of the machine (description in other domain) could take the probability distribution of its service time from some Monte Carlo analysis of its electro-mechanical companion (dependence on generated data).

A core problem is that whenever an action is taken on the MDB, all the actions required to preserve the instated relations should be triggered automatically, and if this is not possible or not convenient, all the involved analysts should be advised of the need to e.g. redo some simulation studies, watch the outcome and resolve the signalled inconsistencies. This of course requires human intervention, but is consistent with our scope. We are not aiming for a system to replace modelling and control culture, this would be just nonsense. We just want professionals who operate within an MDB to possess a human-readable manner to instate the required consistency relations once and for all when required. The role of a DM tool shall just be to enforce these on the MDB. In such a *scenario*, experts could be always sure that no inconsistency will go unnoticed, hence free to concentrate on exploiting their culture.

We can thus further qualify our expected result first as the definition of a structure for an MDB, and then as the choice of (i) a Model Description Language (MDL), (ii) a Data Description Language (DDL), and (iii) an Operations

Language (OL) to respectively represent models, data, and the operations on their compound. Once coordinated choices are made for the three, the abstracted idea of "observing a DMT from a certain standpoint" will possess the technological counterpart of "applying a *view*[2] to an MDB" as the path toward its implementation.

6. The DM paradigm – Model Description Language

For this we resort to Object Oriented Modelling (OOM) Scaglioni & Ferretti (2018). Some frameworks for DT construction already make use of OO languages; for example, Moyne *et al.* (2020) uses OOM to characterise the different types of DTs. Nevertheless, in such works consistency (in the sense above) is not addressed, while in our research, the choice of OOM explicitly aims for establishing consistency in an MDB. Moreover, OOM naturally lends itself to model reuse and extensions thanks to inheritance. Finally, OOM tools are inherently open to interoperation with others by means of standard interfaces like the Functional Mock-up Interface (FMI) briefly mentioned in Section 6.2 later on.

We thus start by abstracting the entities needed to instate the relations of Section 5. We specialise to the Modelica language, also in preparation for the following examples, but ideas are general at least within OOM. We then spend a few words on models not expressible as OOM, for completeness.

6.1. Preliminary definitions

Connector. The representation of a physical terminal (e.g., an electric pin, an input/output for a system, a mechanical flange, a surface) carrying variables (e.g., voltage, current, position, force, temperature, heat flow). The connection of connectors generates equations (e.g., voltages are equal and currents sum to zero, or positions are equal and forces sum to zero).

Interface. A set of connectors *shared by similar entities* (e.g., an interface made of two pins fits resistors, capacitors and the like).

Domain. A modelling theory or a set of theories, applicable to the entities to be described. Examples are continouus- or discrete-time dynamic systems, discrete-event systems, queue networks, Markov chains, time-domain or phasor representations of electric networks.

Type. The qualification of a category of entities (e.g., "resistor" or "conveyor belt") that can be described within one domain or a set of domains (e.g., for the belt, electro-mechanic or queue networks) and/or with different levels of detail (e.g., for a resistor, accounting or not for its parasitic inductance).

Primary Parameters. The set of all the "physical" parameters used when modelling a type, from which those pertaining to the involved domains can be derived (e.g., a time-domain representation of an inductor needs only the inductance, a phasor-domain one also the working frequency to compute the reactance); primary parameters are defined by the type and set by specimens, see later on.

Behaviour. A declarative specification of a modelled object, specifying the domain(s) involved, a unique interface, the contained equations and/or algorithms and the Behaviour Parameters, computed from the Primary ones so as to fit the behaviour (e.g., a transfer function for an RC filter could take the primary parameters R and C to compute its behaviour parameter as the time constant $\tau = RC$). Each behaviour represents a type with a specific interface, including a certain domain set and/or different levels of detail. For example one could model a resistor accounting for detailed electrical phenomena but neglecting thermal storage and release, or *vice versa*. Alternatively, and typically for digital control blocks, a behaviour can be an algorithm to describe – in this case imperatively – how the block works.

Prototype. An entity of a certain type that, pointing to a specific interface within those exposed by the behaviours in that type, makes the so selected behaviours available for parametrisation and use (e.g., a resistor represented in the phasor domain, with all the behaviours in the resistor type that share the phasor interface).

Specimen. A prototype with the Primary parameters values set (e.g., the 1kΩ resistor manufactured by ACME Corp., with parameters as per its data sheet); the specimen is the only place where parameter values are given, for each behaviour listed in the pointed prototype[3]. Some specimen parameters can be flagged as "set by instances". This feature – to not be abused – is useful for the typical case of blocks in a control scheme, where parameters pertain more to the scheme than to the individual blocks.

[2] In relational databases, as an example, a *view* is defined as a query on one or more relations; any operation on the view is automatically translated into operations on the relations from which the view is derived.

[3] In some OOM languages "default" parameter values can be required at the type level to allow for mathematical checks; for example, if a parameter is the size of an array, that must be set to verify that the containing behaviour has as many equations as variables. This is however a purely technical fact, inessential for our treatise.

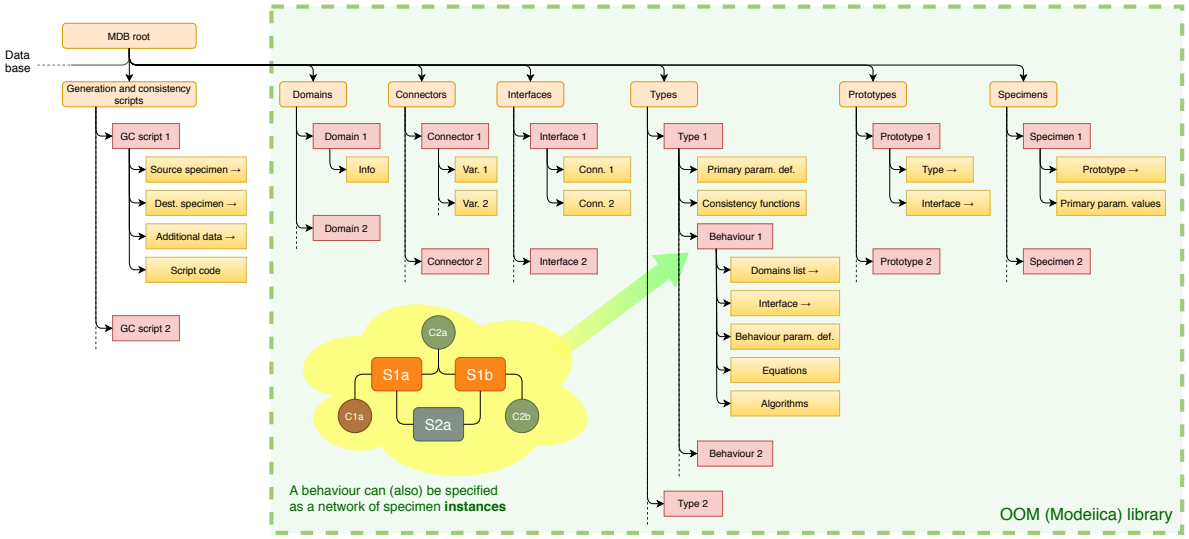


Figure 2: Structure od the model base in an MDB, abstracted view; the → symbol qualifies the entity as a pointer.

Instance. An occurrence of a specimen within a behaviour (e.g., each of the ACME 1kΩ resistors in the scheme for the phasor-domain representation of a circuit); notice that the specimen dictates the interface, thus constraining the choice of its behaviour—the only degree of freedom at this point.

Consistency function. A function, within a type, used to guarantee the consistency of all its behaviours. Such a function often computes the parameters for a behaviour from the physical ones of the type, as assigned by the specimen. This is particularly useful when for example behaviour parameters depend on one another in articulated manners, like 3D arrays of thermal capacities and conductances depend on material properties and spatial discretisation.

Generation and consistency script. A piece of code, in general not written in the MDL but rather exploiting the scripting language that most OOM tools offer, or also in the language of the employed DBMS (Data Base Management System), for "generating" a specimen (hence the required behaviour, type and prototype if needed) based on another one, and for maintaining consistency between the two by re-generating the destination whenever required. The variety of such scripts is connected to the relations to be instated. Referring to the set in Section 5, scripts can linearise, compute parameters based on simulations, apply formulædefined by the analyst for external equivalence, and so forth.

6.2. Model base organisation

We now present the model base organisation in a view to its OOM implementation. Despite we aim for a database-centric approach for project management, we decided to have the model base as a unitary element in the OOM language. This is because OOM tools can check mathematical correctness (as many equations as variables, no structural singularity, and so on). Re-implementing this precious capability in a database would be a large, bug-prone and ultimately useless effort.

The organisation of the model base is illustrated in Figure 2, in turn based on the definitions of Section 6.1. A behaviour can be a set of equations and/or algorithms, or can be specified by connecting specimen instances. This avoids distinguishing "simple" and "aggregate" models, to the advantage of simplicity. For example, Behaviour 1 of Type 1 is specified as a network of specimen instances (S1a and S1b of specimen S1, S2a of specimen S2). Of course none of these specimens is of Type 1, there can be no recursion in systems of equations. In turn, specimens S1 and S2 can either be "elementary" or of a type that has one or more behaviours specified as further networks of other specimen instances. The organisation just defined repeatedly applies throughout the aggregation hierarchy levels in the model base.

We end this section by briefly discussing how to accommodate for models that do not fit within OOM, as is the case for CFD or FEM ones. We first observe that such models are in nature keen to a high level of complexity and detail, hence not deriving from any other model in the sense of Section 5. As such, editing them with the native domain tool and producing an executable to be inserted as behaviour in the form of external binary, using the type parameters to

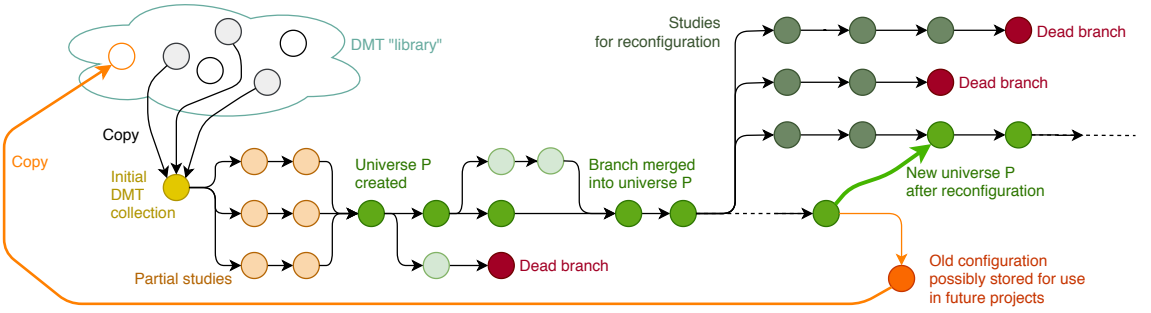


Figure 3: Evolution of a DMT over time along the life cycle of its PT in a simple illustrative case; the visual similarity with the way a project repository appears in typical VCS tools is quite apparent.

hold e.g. configuration files, is compatible with the proposed MDB structure. In some cases such complex models can generate simpler OOM ones; a study referring to distributed flexibility, where FEM models generate Modelica ones, is documented in Ferretti *et al.* (2014).

From the standpoint of the model base architecture, non OOM models can be seen as external binaries. An alternative is nowadays provided by the FMI standard, allowing to "wrap" models from tools that adhere to the standard so as to exhibit an OOM-compatible interface. However FMI allows to connect models for joint execution, but not to subject non OOM ones to the mathematical checks offered by native OOM tools.

We do not further discuss FMI to not stray from the scope of the paper, the reader can refer to Sun *et al.* (2011). We just add a remark about IP protection. If some profile-based access control to an MDB is provided, which is off-the-shelf technology, one could allow "guests" to instantiate a specimen, but only with FMU (Functional Mockup Unit) behaviours of conveniently chosen detail. Since an FMU is compiled, a guest could use that specimen in his/her DMTs, but not see its internals. The usefulness for part manufacturers, who can allow prospective customers to test the said parts in their designs in a standard manner and while keeping IP disclosure under control, is apparent.

7. The DM paradigm – Data Description and Operations Language

The idea of "storing models in a data base" is not totally new, see e.g. Maffezzoni & Girelli (1998), that constitutes an ancestor for our research. From they methodological standpoint our paradigm adds the idea of enforcing relations on *both* data and models, while as for implementation we can nowadays select and combine a wider and more powerful mix of technologies with respect to a single object-oriented database as in Maffezzoni & Girelli (1998). We concluded Section 5 by stating that we need an MDL, a DDL and an OL. We then discussed the MDL. We now spend some words about DDL and OL, and the available technologies.

For the DDL, we select the Relational Data Model (RDM). Strong support is available in the literature for this choice in fields like collaborative design Feng *et al.* (2009), project management Solihin *et al.* (2017) or operations and management Heaton *et al.* (2019). However, while the RDM pairs very well with the data part of an MDB, things are more subtle for the *model* part.

The point is that RDM and object orientation incarnate two different phylosophies: RDM is made to expose information in tables or views, while an object-oriented design aims for hiding (or better, encapsulating) information behind the public interfaces offered for outside access. As long as models are viewed as "programs" – recall Section 4 – the above diversity causes no conflict, but neither model-data nor model-model relations are possible. If models are just stored as data such relations are in principle possible, but the database manager must take complete responsibility about them. OOM tools internally store models with very articulated structures, thanks to which manipulations, checks and similar operations are possible; just storing models in a database with some interchange format does not give access to such capabilities.

Past attempts to solve the issue – see again Maffezzoni & Girelli (1998) – went through an object-oriented design for the model base, joined with some persistence mechanism. Our solution, see Section 6.2, leaves managing the model semantics to an OOM engine (a Modelica tool in our case) and allows the DDL to focus only on data—a task for which RDM is a quasi-ideal choice Feng *et al.* (2009); Solihin *et al.* (2017); Heaton *et al.* (2019). Of course the introduced

separation still requires a lot of implementation work to coordinate the OOM and RDM engines, but relieving the latter from model management in mathematical sense, makes the *scenario* free of conceptual obstacles. Technologically, this just means giving persistence to OOM entities with any available framework, and for operations related to model checks and/or consistency, make the RDM tool pass them over to the OOM engine, and get the result.

Having justifiably chosen RDM as the data model, the natural choice for the OL is SQL (Structured Query Language). However, the peculiarities of DMTs require some further considerations in two directions. The first one is managing the evolution of a DMT over time. The second one is managing models and data (in the broad sense suggested above) that are not part of any DMT; this requires a few more words.

Up to this point we said that "a DMT has a DM nature". But DMTs are not the only entities to have that nature. Organisations create and maintain model *collections* or *libraries* as sources for creating DMTs. The collection of all the DMTs for the projects that the organisation is managing, together with all the DMTs stored in the organisation library but not used in any current project, has itself a DM nature. Such a collection is not a DMT *in the strict sense, however,* because it has no PT and will never be observed nor analysed as such. Nonetheless, it requires to be orderly managed as an MDB.

To address both directions, and as a further distinctive character of our proposal, we suggest to complement the OL with a Version Control System (VCS) such as git Loeliger & McCullough (2012); incidentally, the term "branch" anticipated in the example of Section 5 comes from the VCS (git) jargon.

There is not the space here to describe in full the architecture of the OOM/RDM/VCS (operationally, Modelica/SQL/git) combination that we propose as the technology basis to realise tools for managing DMTs in the DM. We therefore provide just a sketch, in an attempt to touch the main points, prior to concentrating on the OOM part of the design in Section 8.

With reference to Figure 3, a DMT for a new project is generally created by taking elements from a "library" and *copying* them into the new creation ("initial collection" in the figure). Once the required elements are present – of course more can be added or created – some studies are typically carried out in parallel for parts of the overall design (units, machines, control layers, and so on). At some point in time these need to converge in the "master" design, which in our jargon means creating universe P. Further studies can be carried out, also involving the PT if this already exists. Some such studies will be dead branches, some will be realised and thus merged into universe P. Then, reconfigurations may be required: studies will be carried out, and when decisions are taken, one of the so created branches will take over the role of universe P. At any given time all or part of the DMT can be stored into the library for future use. Dead branches simply remain stored in the DMT.

Some remarks are now in order. First, elements taken from a library are *copied*. This prevents undue cross-DMT influences, and is mandatory for an orderly project management. In this respect, the absence of any distinction between elementary and aggregate DMT entities simplifies the *scenario*. Second, merging branches must be supported by any prospective tool from the point of view of the MDB management, but no automation is required as for the semantic content of the merged entities. Said more practically, merging branches *must be a collegial decision of human experts* as it entails project-level responsibility that cannot be devoted to any machine; the role of the tool is to track all operations and avoid MDB inconsistencies—not a negligible help at all, especially in large designs. Third, every time an analyst requires to store some result – e.g., a simulation run – the VCS shall create a snapshot of the *entire* DMT. Modern VCS tools are extremely efficient for this purpose, hence no time or storage space issues are to be expected. Of course a lot of "commits" (the git wording for "project snapshot") will be created, but since we are coupling VCS and a database, the latter will allow to retrieve the sought information quickly and easily.

8. A DMT-capable object-oriented model base

Given the success of OOM tools and the FMI standard, many organisations already possess collections of object-oriented models[4], however not structured as neede for our DMT-centred paradigm. Using the Modelica language, we now present a minimal sketch of such a structuring. The purpose is to show that making an object-oriented model collection "DMT-capable" just amounts to quite straightforward code refactoring, where all the contained knowledge (and incidentally also most of the code) is safely preserved.

In this example we do not bring in a DDL nor an OL for better clarity, hence the Modelica code will be slightly borderline with respect to Figure 2. This is because in a full implementation of our proposal the OOM (Modelica)

[4]In fact our paradigm is applicable also if the organisation model collections adopt non natively OOM tools such as Matlab/Simulink, entailing however some additional (just technological) intricacies that suggest to not treat the matter herein.

tool shall be in charge of mathematical checks, while consistent model manipulation shall also involve the database manager (SQL). For the same explanatory motivation, we limit the scope to one domain.

This said, our example refers to a resistor; "more or less detail" means accounting or not for its inductance and for the resistance variations with temperature.

Listing 1: Model interface example.

```
within ModelInterfaces;
partial model TwoPin_OneHP
  PositivePin p;
  NegativePin n;
  HeatPort    hp;
end TwoPin_1HP;
```

No matter the detail level, the interface is made of two electric pins and a heat port carrying a temperature and a heat rate (all defined in the Modelica Standard Library); the definition of this interface is shown in Listing 1. The type is defined by declaring all the parameters of all the behaviours (the equivalent of the primary parameters in this minimal case) by *extending* – i.e., inheriting – the required interface(s). For compactness, the equations that would inherently pertain to any behaviour can be added here. In our simple case we have no consistency functions. The type definition is in Listing 2.

Listing 2: Model type example.

```
within ModelTypes;
partial model resIF
        extends ModelInterfaces.TwoPin_OneHP
  parameter Resistance   R0 = 1;
  parameter Inductance   L  = 0.1;
  parameter HeatCapacity Ct = 10;
  parameter Temperature  T0 = 293.15;
  parameter Real         a  = 0.01;
end resIF;
partial model res extends resIF;
  Resistance  R;
  Voltage     v;
  Current     i(start=0);
  Power       P,Pt;
  Temperature T(start=T0);
equation
  v=p.v-n.v;
  0=p.i + n.i;
  i=p.i;
  P=R*i^2;
  T=hp.T;
  Pt=hp.Q_flow;
end res;
```

The above given, one can specify behaviour of different complexity as in Listing 3 (not exactly Modelica syntax to show alternatives compactly, "|" separates the alternatives).

Listing 3: Model behaviour examples.

```
within ModelBehaviours;
model B1|2|3 extends ModelTypes.res;
equation
  R=R0;  | R=R0*(1+a*(T-T0)); | same as B2
  v=R*i; | same as B1         | v=R*i+L*der(i);
  T=T0;  | Ct*der(T)=P+Pt;    | same as B2
end B1;
```

This allows to define the resistor prototype as in Listing 4, coming to specimens (we imagine here 1kΩ resistors by two brands) as in Listing 5.

Listing 4: Model prototype example.

```
within ModelPrototypes;
model res extends ModelTypes.resIF;
  replaceable model resB=ModelBehaviours.res.B1
              constrainedby ModelTypes.resIF;
  resB resb(R0=R0,L=L,Ct=Ct,a=a);
equation
  connect(resb.p,p);
  connect(resb.n,n);
  connect(resb.hp,hp);
end res;
```

Listing 5: Model specimen examples.

```
within ModelSpecimens;
model resistor_1k_ACME
  extends ModelPrototypes.res
          (R0=1e3,L=1e-6,Ct=5,alpha=2.5e-4);
end resistor_1k_ACME;
model resistor_1k_Wonka
  extends ModelPrototypes.res
          (R0=1e3,L=4e-6,Ct=2,alpha=3.5e-4);
end resistor_1k_Wonka;
```

Finally, Listing 6 shows a possible excerpt of Modelica code for the model of a circuit with two 1kΩ resistors, one per brand, and requiring two different behaviours.

Listing 6: Usage excerpt: using behaviour B1 for the ACME 1kΩ resistor R1 and behaviour B2 for the Wonka one R2.

```
...
ModelSpecimens.resistor_1k_ACME R1
 (redeclare model
            resB=ModelBehaviours.res.B1);
ModelSpecimens.resistor_1k_Wonka R2
 (redeclare model
            resB=ModelBehaviours.res.B2);
...
```

Even this simple example should convince of how compactly and clearly one can represent complex situations by following our proposal, hence of the advantages of making a pre-existing model base DMT-capable.

As for the problem that in many cases DTs need building when PTs already exist, we just draw the reader's attention on the orange path in Figure 3. As long as a DMT-capable model base is employed, one can create a "model" of an existing PT, and legitimately call this a *single-universe DMT*. In other words, one can operate as usual and create DTs with any interpretation, just caring that along this "usual" process the model base be organised in accorcance with per Figure 2. We do not further discuss this matter, however based on the above we conjecture that the adoption of our DM paradigm, if properly designed and managed, can be an acceptably smooth process.

9. An example on model relations

In this section we give a brief example of how crucial model-model relations are for DMT consistency. The example refers to the "description in other domain" relation, and considers a working station in which parts are heated in preparation to some subsequent operation. Given the variable condition of the incoming parts, the heater-to-part exchange coefficient is variable as well. We assume this variability to be characterised by a probability distribution p_{EC}, that can be obtained – this is important – with convenient tests *before* the station is built (we call such information *a priori* uncertainty). The heater is regulated by a temperature controller of PI type; we denote its gain by K.

For the station we have two behaviours. The former is a cyber-physical one representing the part heating phenomenon, the heater dynamics and the digital controller. The latter is a DEVS-type queue and server one, just characterised by the probability distribution p_{HT} of the heating time. The first behaviour is used to study the station and

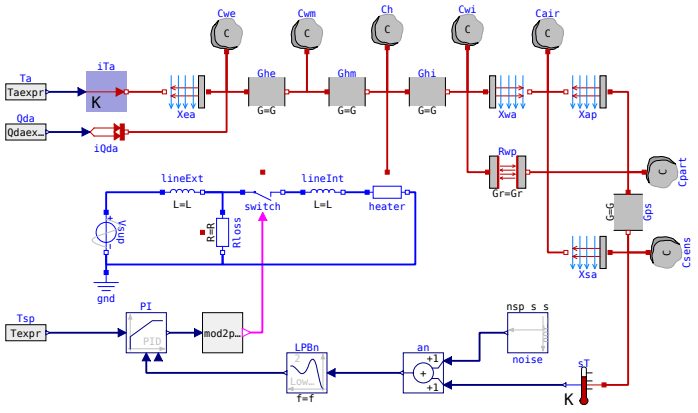


Figure 4: The complex physical behaviour referred to in the example.

control design, the second as part of larger factory-level models aimed for example at throughput analysis. The complex behaviour (the simple one is just a block) is illustrated as Modelica model in Figure 4.

The point is that p_{HT} (in our vision the *a posteriori* uncertainty, the one observed when the station runs) depends – among other facts not analysed here – on *both* the *a priori* uncertainty p_{EC} and the control parameter K. To witness that, Figure 5 reports two aspects of p_{HT} for the same p_{EC} and two values of K.

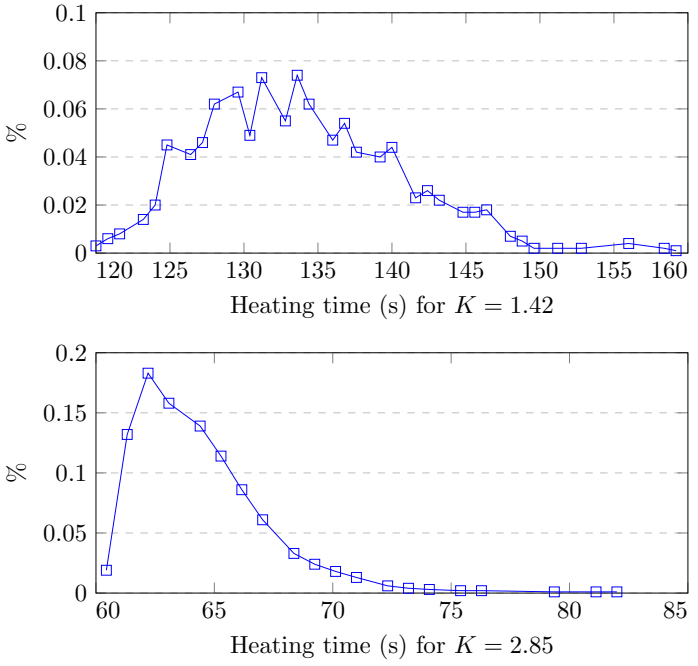


Figure 5: Heating time probability distributions for the two values of a control parameter.

Should the station designer decide to change K for any reason, the DEVS behaviour would need updating so as to maintain consistency. The same holds when the variation – hence, the *a priori* uncertainty – regards a process model parameters, as shown in Cimino *et al.* (2020b). Ongoing research activities deals with relating the *a priori* and *a posteriori* uncertainty in industrial complex cases, a drying process in a rotary table. Keeping the table speed fixed, the goal is to relate the final temperature of the dried compound with respect to its inlet temperature, that in turn depend on the inlet temperature control system. The results obtained from the industrial case – not entirely presented herein for space reasons – highlight the importance of instating those relation for a set of parameters variation. Nevertheless, in

the absence of a properly instated and automatically enforced "description in other domain" relation from the physical to the DEVS behaviour, as described by this paper, studies carried out with the DEVS behaviour may fail to represent reality as per the designed control.

The conclusions just drawn could apply to many different analyses throughout an asset life cycle. Most notably, for example, when control parameters are changed the "normal" system behaviour changes as well, and any online anomaly detection system must adapt to this "new normality" to not give erratic results (the interested reader can find in Cimino *et al.* (2020a) a discussion on this matter). Considering how many intertwined such cases can arise in a large design, the importance of the DM paradigm to follow the entire asset life cycle is apparent.

10. Related work

We now analyse some relevant literature to support our main statement that existing DT interpretations should be just different viewpoints of a single entity, hence that our proposal is in fact recognising and suitably formalising a matter of fact. The review Cimino *et al.* (2019) shows that DT applications can be grouped by development purpose. A DT can be used to support a supply chain Souza *et al.* (2019), for monitoring and improving production Angrish *et al.* (2017), for maintenance purposes Wang *et al.* (2015) or safety Kuts *et al.* (2017). In some cases it can also be used for more complex purposes, such as design Anand *et al.* (2018) or whole life cycle support Konstantinov *et al.* (2017), or to handle flexibility David *et al.* (2018).

Each purpose can be dealt with at a different level of detail: for example, a system can be monitored at the single process level Haag & Anderl (2018) or at the production line level Um *et al.* (2017). The same applies to more complex procedures, like the entire design of a machine Anand *et al.* (2018) or of a product/production environment Caputo *et al.* (2019), and sometimes even to fully redesigning a working environment Dupláková *et al.* (2019). Finally, each DT can have different representations, in that it can be based on simulation models or created from field data analyses. Indeed, together with an explosion in the number of papers on the DTs at large, the last years also exhibit a remarkable increase in the number of reviews on the topic, which categorise them in many different ways, each one with a different scope and purpose such as to systematise the DT roles Sjarov *et al.* (2020), to use modelling and simulation strategies in manufacturing Zhang *et al.* (2019); de Paula Ferreira *et al.* (2020) or to set up and apply different maintenance strategies Errandonea *et al.* (2020).

Overall, a lot of papers in the end foster – somehow unconsciously, we bear to say – the use of a DT for its DW nature. Nevertheless, they take only partial – if any – advantage of the multiverse nature of the DW, representing a "standalone" point of view, or a tailored – we said "verticalised" – application. Without diminishing the relevance of all such DTs, they are in fact all problem-specific in the sense we discussed above. Also, different DTs describe the same system from different viewpoints even for the same purpose, and *vice versa* the same viewpoint appears in DTs of various kinds – model- or data-based, most typically – at different levels of detail. This is not an issue *per se*, as different DTs can coexist in the same system; but once again, in this case ensuring consistency is a must.

Summing up, the literature confirms that the multiverse nature of the DW – let alone its exploitation – is largely missed. Also promising efforts to coordinate DT interpretations and deliver comprehensive lifelong asset management environments, most notably AAS (Wagner *et al.*, 2017), in fact do not encompass any multiverse character, nor any means to bind models and data by shared consistency rules – which is a significant extension of "interoperability" as intended e.g. in Platenius-Mohr *et al.* (2020) – at least in the sense we argued herein to be required.

On a slightly different but related front, in addition, DTs have been proposed for many different analyses during the life cycle of an asset, and even for life cycle management itself Macchi *et al.* (2018). This led to primarily focus on DTs as simulation environments in a digital or cyber world – connected to the physical one – and on some infrastructure that can keep record of the model and/or data used to perform those simulations. Certainly our proposal follows – but also extends – this research branch, that is undoubtedly our nearest neighbour in the literature. For example, Wright & Davidson (2020) consider the DT as based on models that reflect a physical entity and are constantly updated by data, while Lechler *et al.* (2020) refer to the DT as "the set of executable digital applications and functions utilizing the Engineering and Extension Models together with the digital Shadow in order to visualize, identify, predict or control states of a physical asset thereby generating additional value for the asset or its user", and He *et al.* (2021) reconsider the DT as based on data storage through the BIM concept applied to manufacturing.

Although all the above are definitely valuable proposals, in the authors' opinion, the missing element is (once again) the presence of a higher abstraction level that collects, structures and connects all the models and data together in the DW—where "connects" needs interpreting in the sense of our consistency relations.

11. Conclusions and future research

Introducing the different interpretations and declinations of the DT idea currently present in literature, this paper conjectures that they cannot be operationally reconciled and consistently related at the same abstraction level where they reside. For example, DTs are used in literature for the development of different applications that can be categorised as online or offline, depending on the presence of connection with the related PT. We concluded that the main DT actors are *models* and *data* (in the sense we suggested in the text); while the major reasons for the impossibility of a convergence of all the DT interpretations lies in the *multiverse nature* of the DW and the present inability of treating the said models and data in a DT on an equality basis. It is thus important to formally relate not only data to data, but also data to models and models to models.

Within the aim of filling the so identified gap, we proposed a paradigm named the Digital Multiverse (DM), where entities named Digital Meta Twins (DMTs) exist and evolve. Universes in the DM represent design alternatives; viewpoints reflect the possible representation types; detail levels correspond to coarser or finer phenomena descriptions. In this context, the main fundamental properties of a DMT can be summarised as follow.

- When observed from a given viewpoint and focusing on one universe, a DMT presents to the analyst a DT in the sense of that viewpoint and with the choices of that universe (and a certain level of detail). One of the universes, named P, follows the parallel life of the PT for the DMT.
- All the models and data are *together* bound by relations, so that no operation on any view on the DMT (i.e., on a certain presented DT) can lead to inconsistencies in the Model and Data Base (MDB) build through those data, models and consistencies.

Once outlined the paradigm, we discussed how to realise its core element, i.e., the MDB. We elicited a combination of three main technologies (OOM, RDM, VCS) to obtain the required Model Description Language, Data Description Language, and Operations Language. We gave (a few) examples to evidence the yielded advantages, also showing how existing model bases can be re-factored to be cast into the DM paradigm, to the advantage of acceptability.

As highlighted in the paper, and especially in the case studies, our proposal is however quite complex. It can however fit a large variety of application domains, as well as follow the entire life cycle of an asset. However, at this stage we do not have a wide enough *corpus* of experiences to support the above statement, hence we defer the necessary studies to future activities.

Future research will also detail the proposal up to a full specification, and include in the paradigm other analyses related to an asset life cycle. This will be fundamental to increase the inclusiveness of the approach. We shall also study how the proposed paradigm and technologies can be coordinated with AAS-based frameworks. Pilot implementations are also envisaged and some are underway, connected to industrial case studies.

References

Anand, S., Ghalsasi, O., Zhang, B., Goel, A., Reddy, S., Joshi, S., & Morris, G. 2018. Additive Manufacturing Simulation Tools in Education. *Pages 1–6 of: Proc. 2018 World Engineering Education Forum – Global Engineering Deans Council*.

Angrish, A., Starly, B., Lee, Y.S., & Cohen, P.H. 2017. A flexible data schema and system architecture for the virtualization of manufacturing machines (VMM). *Journal of Manufacturing Systems*, **45**, 236–247.

Caputo, F., Greco, A., Fera, M., & Macchiaroli, R. 2019. Digital twins to enhance the integration of ergonomics in the workplace design. *International Journal of Industrial Ergonomics*, **71**, 20–31.

Cimino, C., Negri, E., & Fumagalli, L. 2019. Review of digital twin applications in manufacturing. *Computers in Industry*, **113**, 103130.

Cimino, C., Leva, A., Negri, E., & Macchi, M. 2020a. An integrated simulation paradigm for lifecycle-covering maintenance in the Industry 4.0 context. *IFAC-PapersOnLine*, **53**, 10556–10561.

Cimino, C., Ferretti, G., & Leva, A. 2020b. The role of dynamics in digital twins and its problem-tailored representation. *IFAC-PapersOnLine*, **53**, 307–312.

David, J., Lobov, A., & Lanz, M. 2018. Leveraging digital twins for assisted learning of flexible manufacturing systems. *Pages 529–535 of: Proc. 16th IEEE International Conference on Industrial Informatics*.

de Paula Ferreira, William, Armellini, Fabiano, & De Santa-Eulalia, Luis Antonio. 2020. Simulation in industry 4.0: A state-of-the-art review. *Computers & Industrial Engineering*, 106868.

Dupláková, D., Flimel, M., Duplák, J., Hatala, M., Radchenko, S., & Botko, F. 2019. Ergonomic rationalization of lighting in the working environment. Part I: Proposal of rationalization algorithm for lighting redesign. *International Journal of Industrial Ergonomics*, **71**, 92–102.

Errandonea, Itxaro, Beltrán, Sergio, & Arrizabalaga, Saioa. 2020. Digital Twin for maintenance: A literature review. *Computers in Industry*, **123**, 103316.

Feng, G., Cui, D., Wang, C., & Yu, J. 2009. Integrated data management in complex product collaborative design. *Computers in Industry*, **60**(1), 48–63.

Ferretti, G., Leva, A., & Scaglioni, B. 2014. Object-oriented modelling of general flexible multibody systems. *Mathematical and Computer Modelling of Dynamical Systems*, **20**(1), 1–22.

Haag, S., & Anderl, R. 2018. Digital twin – Proof of concept. *Manufacturing Letters*, **15**, 64–66.

He, Rui, Li, Mingkai, Gan, Vincent JL, & Ma, Jun. 2021. BIM-enabled computerized design and digital fabrication of industrialized buildings: A case study. *Journal of Cleaner Production*, **278**, 123505.

Heaton, J., Parlikad, A.K., & Schooling, J. 2019. Design and development of BIM models to support operations and maintenance. *Computers in industry*, **111**, 172–186.

Konstantinov, S., Ahmad, M., Ananthanarayan, K., & Harrison, R. 2017. The cyber-physical e-machine manufacturing system: Virtual engineering for complete lifecycle support. *Procedia CIRP*, **63**, 119–124.

Kuts, V., Modoni, G.E., Terkaj, W., Tahemaa, T., Sacco, M., & Otto, T. 2017. Exploiting factory telemetry to support virtual reality simulation in robotics cell. *Pages 212–221 of: Proc. 2017 International Conference on Augmented Reality, Virtual Reality and Computer Graphics*.

Lechler, Tobias, Fuchs, Jonathan, Sjarov, Martin, Brossog, Matthias, Selmaier, Andreas, Faltus, Florian, Donhauser, Toni, & Franke, Jörg. 2020. Introduction of a comprehensive Structure Model for the Digital Twin in Manufacturing. *Pages 1773–1780 of: 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE.

Loeliger, J., & McCullough, M. 2012. *Version control with git: powerful tools and techniques for collaborative software development*. Sebastopol, CA, USA: O'Reilly Media.

Macchi, Marco, Roda, Irene, Negri, Elisa, & Fumagalli, Luca. 2018. Exploring the role of digital twin for asset lifecycle management. *IFAC-PapersOnLine*, **51**(11), 790–795.

Maffezzoni, C., & Girelli, R. 1998. MOSES: modular modelling of physical systems in an object-oriented database. *Mathematical and Computer Modelling of Dynamical Systems*, **4**(2), 121–147.

Moyne, J., Qamsane, Y., Balta, E.C., Kovalenko, I., Faris, J., Barton, K., & Tilbury, D.M. 2020. A requirements driven digital twin framework: Specification and opportunities. *IEEE Access*, **8**, 107781–107801.

Platenius-Mohr, M., Malakuti, S., Grüner, S., Schmitt, J., & Goldschmidt, T. 2020. File-and API-based interoperability of digital twins by model transformation: An IIoT case study using asset administration shell. *Future Generation Computer Systems*, **113**, 94–105.

Scaglioni, Bruno, & Ferretti, Gianni. 2018. Towards digital twins through object-oriented modelling: a machine tool case study. *IFAC-PapersOnLine*, **51**(2), 613–618.

Sjarov, M., Lechler, T., Fuchs, J., Brossog, M., Selmaier, A., Faltus, F., Donhauser, T., & Franke, J. 2020. The Digital Twin concept in industry–a review and systematization. *Pages 1789–1796 of: Proc. 25th IEEE International Conference on Emerging Technologies and Factory Automation*, vol. 1.

Solihin, W., Eastman, C., Lee, Y.C., & Yang, D.H. 2017. A simplified relational database schema for transformation of BIM data into a query-efficient and spatially enabled database. *Automation in Construction*, **84**, 367–383.

Souza, V., Cruz, R., Silva, W., Lins, S., & Lucena, V. 2019. A digital twin architecture based on the Industrial Internet of Things technologies. *Pages 1–2 of: Proc. 2019 IEEE International Conference on Consumer Electronics*.

Sun, Y., Vogel, S., & Steuer, H. 2011. Combining advantages of specialized simulation tools and modelica models using Functional Mock-up Interface (FMI). *Pages 491–494 of: Proc. 8th International Modelica Conference*.

Um, J., Weye, S., & Quint, F. 2017. Plug-and-Simulate within Modular Assembly Line enabled by Digital Twins and the use of AutomationML. *IFAC-PapersOnLine*, **50**(1), 15904–15909.

Villalonga, A., Negri, E., Biscardo, G., Castano, F., Haber, R.E., Fumagalli, L., & Macchi, M. 2021. A decision-making framework for dynamic scheduling of cyber-physical production systems based on digital twins. *Annual Reviews in Control*.

Wagner, C., Grothoff, J., Epple, U., Drath, R., Malakuti, S., Grüner, S., Hoffmeister, M., & Zimermann, P. 2017. The role of the Industry 4.0 asset administration shell and the digital twin during the life cycle of a plant. *Pages 1–8 of: Proc. 22nd IEEE International Conference on Emerging Technologies and Factory Automation*.

Wang, G.G. 2002. Definition and review of virtual prototyping. *ASME Journal of Computing and Information Science in Engineering*, **2**(3), 232–236.

Wang, X., Guo, Y., & Wang, Y. 2015. Automatic detection of regions of interest in breast ultrasound images based on local phase information. *Bio-medical materials and engineering*, **26**(s1), S1265–S1273.

Wright, Louise, & Davidson, Stuart. 2020. How to tell the difference between a model and a digital twin. *Advanced Modeling and Simulation in Engineering Sciences*, **7**(1), 1–13.

Zhang, Lin, Zhou, Longfei, Ren, Lei, & Laili, Yuanjun. 2019. Modeling and simulation in intelligent manufacturing. *Computers in Industry*, **112**, 103123.