Engineering of Trust Analysis-driven Digital Twins for a medical device

M.M Bersani[1], C. Braghin[2], A. Gargantini[3],
R. Mirandola[1], E. Riccobene[2], and , P. Scandurra[3]

[1] Politecnico di Milano,Italy
[2] Università degli Studi di Milano, Italy
[3] Università degli Studi di Bergamo, Italy

Abstract. The DT paradigm has emerged as a suitable way to cope with the complexity of analyzing, controlling, and adapting complex systems in diverse domains. For medical systems, however, the DT paradigm is not fully exploited mainly due to the complexity of dealing with uncertain human behavior, and of preventing sensitive information leakage (e.g., patient personal medical profiles). We present the first results of a long-term recently launched research aiming at engineering a DT for a medical device endowed with trust analyses techniques able to deal with human and environmental uncertainty, and security protection. As a proof of concept, we apply our DT vision to the case study of a mechanical ventilator developed for Covid 19 patient care. The long-term aim is engineering a new generation of lung ventilators where the use of a DT can prevent unreliability and untrustworthiness of a system where interactions, both physical (machine-patient) and operational (machine-medical staff), are characterized by the presence of uncertainty and vulnerabilities.

Keywords: Digital Twin · Medical Cyber-physical Systems · Trust analysis · formal methods

1 Introduction

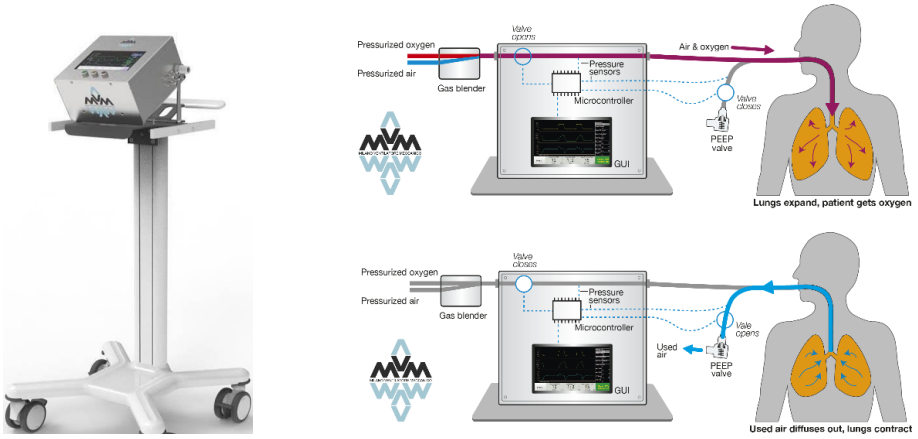
A Digital Twin (DT) is a machine-processable high-fidelity virtual representation of a physical system, called Physical Twin (PT), to which it is coupled through a continuous, bidirectional flow of data (e.g., monitored data and results of predictive/prescriptive analysis). The DT paradigm has emerged as a suitable way to cope with the complexity of analyzing, designing, implementing, controlling, and adapting complex systems belonging to diverse domains such as cyber-physical, business, and societal systems (e.g., [17, 24, 33]). In particular, one of the field in which the concept of DT has been largely used is manufacturing, where a categorical literature review [24] has set the stage for a consolidated definition of DT.

The healthcare sector is also keen to implement these technological solutions to enhance medical care and patient treatment. DTs can provide informed and relevant responses in real time, support decisions and assess risks for the patients through simulation and analysis of up-to-date models that represent physical agents and machinery pieces, and also be adopted for medical staff training. To this aim, serious steps have

been taken in creating DTs of patients as well as DTs of medical devices [4, 22], with potential multiple benefits for doctors such as discovering undeveloped illnesses, experimenting with treatments, and improving preparation for surgeries. However, there are several barriers to the adoption of DTs. Besides technological and methodological aspects, hindrances can be the complexity of modeling human behaviors, human physiology, and operational workflows, of dealing with uncertainty, and of preventing sensitive information leakage (e.g., patient personal medical profiles). In general, the DT full exploitation also heavily depends on the trust that stakeholders have in DTs and the insights they provide. The actual corpus of study mainly focuses on security and privacy [18] and overlooks many of the aspects that are of particular interest to the system's stakeholders. Indeed, a stakeholder will hardly rely on a DT that guarantees acceptable levels of security and privacy but fails to meet required performance, dependability, or safety levels, or that does not even reflect the mirrored medical cyber-physical system (hereafter indicated as MPT for medical physical twin), as it will likely provide out-of-time or wrong feedback.

In this paper, we present the first results of a recently launched long-term research aiming at engineering a DT for a medical device that makes use of formal methods and analysis techniques. In particular, we here contribute with a reference architecture model for DT trust assurance. We concretely show how DT components of this architecture model are built and connected by embedding formal models and quality analysis techniques for their use at runtime [5, 34] within DT engineering platforms emerging on the market. These last are to be intended as runtime implementation platforms and currently are mainly data-oriented since only offer data analytic services. A very preliminary version of our DT trust assurance vision was presented as a poster at the ICSA conference 2022 [6]. Here, we make a step ahead and present a reference architecture model that concretely realizes our vision by complementing current DT engineering platforms with formal modeling and analysis techniques for behavior-oriented analysis. Specifically, with this architecture we envision a solution aiming at: (i) taming the complexity in modeling the heterogeneity of the DT components that must be developed, deployed, and evolve together with their physical counterparts; (ii) increasing the level of trust in the results and prescriptions coming from a DT, despite uncertainties due to modeling approximations and incomplete or imprecise data collected in the field. We also concretely show the potential applicability of the proposed architecture model through a running example, namely the Mechanical Ventilator Milano (MVM) [3]. Such a system has been developed during the Covid 19 pandemic to answer the high request of mechanical lung ventilators. MVM is a low-cost and fast-to-develop medical system that has been successfully designed, certified, and is currently built and delivered (especially to emerging countries). Applying our vision of DT to the MVM enables the design and development of complex medical scenarios that take place in physical environments in which human agents (e.g., patients and medical staff) interact with a new generation of lung ventilators and the interactions, both physical (machine-patient) and operational (machine-medical staff) are characterized by the presence of uncertainty.

In the proposed architecture model, trust assurance is realized through model-based quality assessment techniques, allowing both *if-what* and *what-if* analyses. The first one is executed on the DT and allows for detecting violations of the quality and security



(a) MVM Ventilator (b) MVM supply for patient ventilation [9]

Fig. 1: Mechanical Ventilator Milano

requirements that may compromise the quality of the patient's interaction with the system. Hence, changes to the digital models are evaluated and possibly reflected into the MPT (e.g. the analysis over the ventilator model may show that it is not able to detect a high inspiratory pressure in a particular patient state; hence a modification of the model and possibly of the MPT is mandated). What-if analysis, on the other hand, applied on DT detects forthcoming criticalities and might suggest requirements evolution (e.g. the ventilator waveform analysis detects patient-ventilator asynchronies which require a change in the ventilator settings - like the trigger sensitivity). The proposed techniques can be opportunely combined by introducing methods that allow for the composition of analysis, and the combination of analysis results within a DT [32]. A major benefit in using composed model-based analysis is the ability to carry experiments that would be costly on a real system.

This paper is organized as follows. Section 2 describes the running example in the domain of medical cyber-physical systems. Section 3 presents our view of a Digital Twin, while Section 4 illustrates fragments of the formal models adopted for the running example. Section 5 shows an example of compositional reliability analysis trough a reliability model for the overall DT system, and also examples of analysis from a security perspective. Finally, Section 6 concludes the work.

2 Running example: MVM Case study

MVM (Mechanical Ventilator Milano) [3] is an electro-mechanical ventilator (see Fig. 1a), which is intended to provide ventilation support for patients that are in intensive therapy and that require mechanical ventilation. MVM works in pressure-mode, i.e., the respiratory time cycle of the patient is controlled by the pressure, and, therefore, this ventilator requires a source of compressed oxygen and medical air that are readily avail-

able in intensive care units. More precisely, MVM has two operative modes: *Pressure Controlled Ventilation* (PCV) and *Pressure Support Ventilation* (PSV).

Fig. 1b shows the inspiration and expiration breathing flows of a patient connected to the ventilator [9]. In the PCV mode, the respiratory cycle is kept constant and the pressure level changes between the target inspiratory pressure and the positive end-expiratory pressure. New inspiration is initiated either after a breathing cycle is over, or when the patient spontaneously initiates a breath. In the former case, the breathing cycle is controlled by two parameters: the respiratory rate and the ratio between the inspiratory and expiratory times. In the latter case, a spontaneous breath is triggered when the MVM detects a sudden pressure drop within the trigger window during expiration. The PSV mode is not suitable for patients that are not able to start breathing on their own because the respiratory cycle is controlled by the patient, while MVM partially takes over the work of breathing. A new respiratory cycle is initiated with the inspiratory phase, detected by the ventilator when a sudden drop in pressure occurs. When the patient's inspiratory flow drops below a set fraction of the peak flow, MVM stops the pressure support, thus allowing exhalation. If a new inspiratory phase is not detected within a certain amount of time (apnea lag), MVM will automatically switch to the PCV mode because it is assumed that the patient is not able to breathe alone.

To give an idea of the complexity of the entire MVM, its detailed behavior is described in the requirements documents which count altogether about 1000 requirements, each being a brief sentence. One document describes the behavior of the overall system, while 15 requirements documents describe the detailed behavior of software components. The controller itself has its own requirement document which consists of 31 pages and 157 requirements.

3 Our view of a Digital Twin

We take an ensemble modeling approach in which DT analysis tasks are carried out with multiple multi-paradigm models. Specifically, we propose a two-layer architecture: the *Physical Twin*, and the *Digital twin*.

The *Digital twin* layer realizes a *twin model graph* (see Fig.2) made of digital models of real-world entities of interest (things, places, devices, processes and people) connected via relationships. This layer is further split into two sub-layers: the *DT runtime models* and the *DT engineering technology*. In the following we describe these two sub-layers and detail how trust assurance could be realized according to this reference framework. Some concrete examples of runtime models and types of analysis for the MVM running example are instead given in Section 4 and 5.

3.1 DT runtime models

This is the highest layer of digital models and trust analysis. Digital models are analysis/analytical models used as *living models* at runtime [5,34]. The composition of model formalisms and property formalisms enables a global DT analysis for trust, considering both the heterogeneity of the evolving models and the uncertainties both at physical and digital level. DT analysis questions can be managed at the level of the models and data

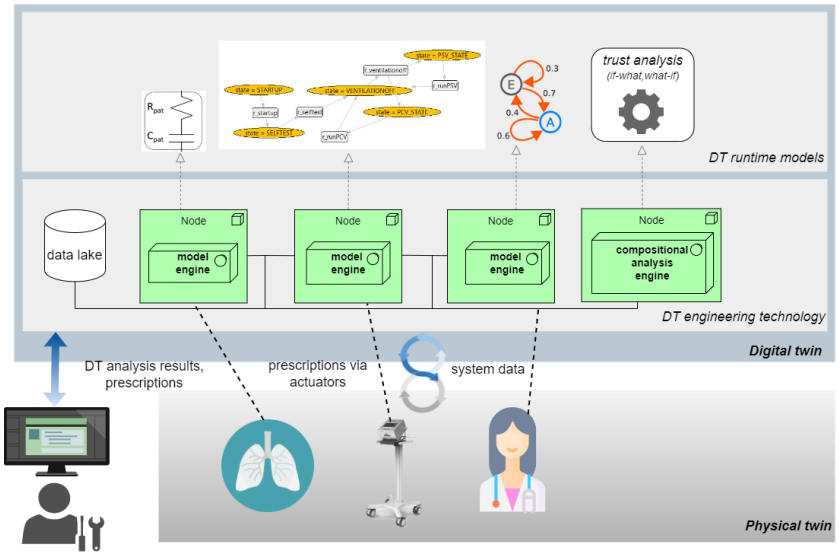


Fig. 2: DT engineering: a twin model graph contextualized for the ventilator case study. Dashed lines show the correspondences between system elements and digital entities.

involved, by defining and applying appropriate composition/decomposition relational operators (e.g., merge, union, focus, restriction, etc.). These operators are grounded on the semantic domains of the composed formalisms and their associated analysis techniques and engines [20].

3.2 DT engineering technology

This is the layer where the development, deployment, and interconnection of the DT-PT components occurs. It is also responsible for the execution and interconnection among runtime models to develop the basis of the twin model graph and accomplish the function of trust analysis. A DT engineering platform may be used as development infrastructure for such a scope. Such a technology would allow us to create a comprehensive digital representation of the individual PT entities/environments and types of relationships, but also to embed conventional model execution and analysis tools for enabling the use of formal/analytical models at runtime, and connect such models to each other. The DT engineering platform also facilitates us in exploring the data and the models in the DT graph (through a visual tool, or via API calls, or CLI commands) to view, query, and manage the models, their relationships and analysis results.

A central concept of the DT engineering platform is how to connect the runtime data to the model graph (PT-DT *system data*), and how to connect the models (and therefore the model engines) of the graph to each other. The DT engineering platforms currently available range from solutions coming from industry, such as Azure

Digital Twins[4], to research proposals coming from academia, such as a six-layer architecture enabling data and information exchange between cyberspace and the physical twin [29], or [23]. In such platforms, CPSs and DTs are integrated by using a common domain-specific language on which an appropriate communication interface and related communication infrastructure are based. In principle, to enable the communication among the model engines and coupling the simulation of runtime models, universal asynchronous messaging communication protocols designed for the Internet of Things (such as MQTT, AMQP, ZeroMQ, etc.) could be exploited in small and specific solution contexts. For example, for the MVM running example, in [8] and [7] examples of simple co-simulation infrastructures were realized to connect JVM-based implementations of model engines and system simulators by exploiting the ZeroMQ asynchronous messaging library[5] for distributed or concurrent applications. In particular, in [7] an Arduino-based prototype of the MVM controller (obtained by code generation from a formal model) is pairwise coupled with a custom-made Java breathing simulator so the MVM could communicate the status of the valves and therefore set the pressure of the ventilator and read breathing events; first a serial port was exploited, and then in a second version a communication channel was realized by exploiting ZeroMQ. More appropriate standards and related runtime infrastructures for distributed co-simulation in a federated and interoperable simulation environment – such as the IEEE 1516 High Level Architecture (HLA) and the Functional Mockup Interface (FMI) [2] – are also being adopted [21].

Another important concept for engineering a DT is the *twinning rate* [36], namely, how often the synchronization between the PT and the DT (in both directions), and therefore the trust analysis, should take place. Usually, in near real-time scenarios, the twins are always aligned, but there could be phases in which the data within the DT could be outdated w.r.t. the PT. Regarding the trust analysis, different analysis rates could be adopted, also depending on the type of analysis conducted. Analysis could be, for example, triggered on request, or executed periodically from time to time or according to a specific plan, or triggered as a consequence of critical or adverse events (such as death of patients, latency in the medical intervention, etc.), or even continuously in an underground mode with a certain time granularity.

3.3 Trust assurance

Trust assurance is realized through quality properties techniques allowing both if-what and what-if analyses. Specifically, if-what analysis exploits runtime QoS analysis techniques (such as analytical and simulation approaches [1], runtime verification [12], security and safety enforcement [16], and reactive adaptation [35]) to detect violations of the quality properties that may lead to evaluate changes to the digital models and possibly to the MPT. What-if analysis is mainly conducted from the adoption of forecasting methods: statistical techniques and machine learning algorithms, combined with proactive adaptation [35] to supply predictions of the evolution of the DT-MPT in the near

[4] `https://azure.microsoft.com/en-us/services/digital-twins/`
[5] `https://zeromq.org/`

future and the re-alignment of MPT w.r.t. DT (e.g. the runtime monitoring of the ventilator can detect critical situations like insufficient tidal inspiratory volume - computed by integration of the measured air flow - and signal it with an alarm for the medical staff).

Applying our vision of DT to the MVM enables the design and development of complex medical scenarios that take place in physical environments in which human agents of various kinds, such as patients and medical staff, interact with a new generation of lung ventilators; the interactions, both physical (machine-patient) and operational (machine-medical staff ruled by medical procedures), are characterized by the presence of uncertainty. In such scenarios trust, prediction, and adjustment play a central role. Based on our framework, a DT for a medical scenario, such as a pneumology ward using a system like MVM for the therapy of patients, can help clinicians evaluate and improve the care. Enhanced analysis and monitoring capabilities help the medical staff identify possible situations that may lead to a decrease in the quality of the service provided to patients caused, for example, by delays in the provision of treatment, by unsuitable instrument settings or potential failures of the devices. The outcomes of the analysis can therefore be useful to initiate a maintenance phase for instruments or procedures, and also support the design and testing of new medical procedures and more powerful and safer ventilators before their use with actual patients.

4 DT runtime models

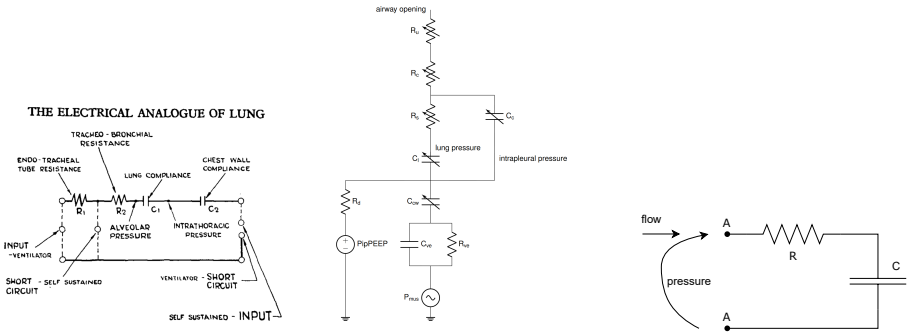
This section presents examples of analysis models used as living models at runtime within the DT.

Model of the lung. There exist many models for the human lungs, and most of them exploit the analogy with electrical circuits, where the voltage is the pressure and the current intensity is the air flow. By means of these models of increasing complexity, researchers aim to capture different aspects of human breathing cycle, like the capacity (often called compliance) of the lungs, the capacity and resistance of other parts of the human body (trachea or other parts), and other patient features (including the capability of spontaneous breath).

Three models of human lung are shown in Fig. 3. Fig. 3a reports a model, taken from [13], in which a simple non-sophisticated electric circuit show the input from the ventilator, several parts of the human lung, and their contributions in terms of compliance and resistance. By short-circuiting at the junction of R1 and R2, the diagram represents the conditions pertaining when the patient breathes spontaneously.

Simple lung models can be extended as shown in Fig. 3b, that reports a model presented in [15]. These models con be used for generating synthetic data sets for machine learning and for educational use.

The simplest possible model of lungs is shown in Fig. 3c and briefly illustrated in the following. The *capacity* or compliance (C) describes the elastic property of the respiratory system, and is usually expressed in ml/cmH2O. In patients with a normal lung undergoing mechanical ventilation, C is 50–60 ml/cmH2O. Decreased compliance may occur, for example, in the case of acute respiratory distress syndrome (ARDS).



(a) A simple model with spontaneous breath [13]

(b) A more complex model [15]

(c) The simplest lung model

Fig. 3: Lung models of different complexity

Monitoring and continuously estimating compliance in patients can provide information about the volume of the aerated lung.

The *resistance* (R) describes the opposition to a gas flow entering the respiratory system during inspiration, and can be calculated as the ratio between the pressure driving a given flow and the resulting flow rate. The dimension of resistance is usually cmH2O/(l/s). Estimating R is of extreme importance because it allows the doctors to choose the right inspiratory pressure and the right time cycle.

Model of the doctor. When designing a complex digital twin that includes human agents, the designer must take into account different sources of uncertainty caused by the autonomous action of humans. Human action is subject to numerous influences, mostly stemming from free will, prior experience and physiological factors (e.g. the fatigue, focus, etc.). These components can cause the expected human behavior to deviate from a known operational workflow. Moreover, human action can be subject to error. Even if the intention of a trained operator is to act with respect to a known plan, the action realized by the operator may not be implemented through the actions prescribed by the plan or may not fulfill some of the desired qualitative properties. As shown in [25, 26], it is possible to model the workflow of actions, certain physiological factors and human errors by means of Stochastic Hybrid Automata (SHA). A stochastic hybrid automaton is defined by a finite number of locations, modeling different operational states of the operator, and a number of continuous-time real variables, the evolution of which is described in each location by means of appropriate differential equations expressed with respect to real time (called *flow conditions*). Intuitively, when the automaton remains in a location for a certain time t, each time-dependent variable evolves as a function of t, and possibly other parameters. Real variables model physical quantities with complex temporal dynamics as a function of the current operational state. In [25, 26], an individual's muscle fatigue is modeled by states of fatigue and recover, each characterized by several exponential-type equations. In a SHA, the transition between one operational state and another occurs through the execution of an edge that

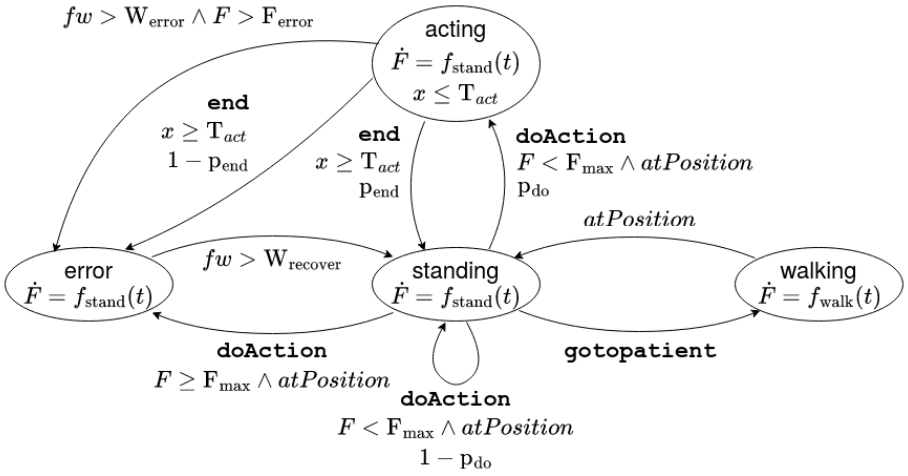


Fig. 4: SHA modeling a doctor collaborating with the ventilator.

connects the two locations. The execution of an edge corresponds to the occurrence of an event in the system. In SHAs, edges can be stochastic as they are associated with a probabilistic weight and labeled with conditions expressed by variables in the SHA. When an edge is executed, the associated condition is true at the time of execution.

Figure 4 shows a simple example of a SHA modeling four operational states of a doctor working in a ward. The SHA has three locations called *standing*, *walking* and *acting* representing, respectively, the actions of standing, walking in the ward and operation through the ventilator. The $error$ location models any potentially harmful situation that occurs during the activity with the ventilator, e.g., because of a wrong command issued with the dashboard. In each location, fatigue is the physiological quantity of interest, which is modeled by a variable F whose derivative is bound to a specific equation; in standing, acting and error, fatigue decreases (equation f_{stand}) while in walking, fatigue increases (equation f_{walk}). Some of the transitions are labeled with the actions $gotopatient$, $doAction$ and end which represent the customary events in a workflow of a doctor. The initial state of the automaton is $standing$, as it represents the doctor waiting for the start of the medical procedure or the occurrence of an alarm, which occurs when action $gotopatient$ is taken.

A medical procedure or an alarm implies an initial walking phase of the doctor leaving the office to join the patient in the ward. To keep the model simple, the walking phase is devoid of deviations caused by human free will but, in general, uncertainty can also characterize this phase. When the doctor is again standing because the patient has been reached (transition between walking and standing labeled with predicate $atPosition$), the diagnostic activity can begin; this fact is modeled by the event $doAction$. The action being modeled is a collaborative action that the doctor performs with the ventilator, and it represents the start event that occurs when the doctor issues a

command from the ventilator's dashboard. Upon the occurrence of this event, the doctor can start the activity (location *acting*) if his fatigue is below a limit value $\mathrm{F}_{\max}$, but even in this condition, the doctor may decide to delay the activity due to free will. The actual initiation of the diagnostic activity is modeled by the transition between location *standing* and *acting* labeled with probability p_{do}; while the postponement of the collaborative action is modeled by the self-loop on location *standing* labeled with probability $1-\mathrm{p}_{\mathrm{do}}$. However, if the doctor is too fatigued, i.e. when $F \geq \mathrm{F}_{\max}$, and the event $doAction$ occurs, it is possible for the doctor to make an error. This fact is modeled by means of the location *error* and the transition from location *standing* to location *error*. During the activity, it is possible for the doctor to make a mistake when fatigue exceeds a threshold limit and free will intervenes. This situation is modeled by means of the transition between *acting* and *error* labeled with the condition $fw > \mathrm{W}_{\mathrm{error}} \wedge F > \mathrm{F}_{\mathrm{error}}$. The variable fw models free will with a real value that is defined by simulating a coin toss when the automaton enters the location acting (other constructions are possible).

The doctor's collaborative activity with the ventilator (location *acting*), has in this example a fixed duration of $\mathrm{T}_{\mathrm{act}}$ time units. The measurement of the duration of the activity in the model is by means of a clock x which is reset when the transition between *standing* and *acting* occurs. If the doctor's activity proceeds to the end, i.e. when the activity has lasted $\mathrm{T}_{\mathrm{act}}$ units of time, the doctor notifies the end of the action. This fact is modeled by the event end. Even in this situation, due to free will, the interaction with the dashboard can be wrong. The end of the activity without error is modeled by the transition between location *acting* and *standing*, labeled with probability $\mathrm{p}_{\mathrm{end}}$; the occurrence of an error is modeled by the transition between location *acting* and *error*, labelled with probability $1 - \mathrm{p}_{\mathrm{end}}$. After the occurrence of an error, the free will of the doctor governs the return to a nominal operating phase and the reestablishment of a non-emergency working condition. This fact is modeled by the transition between *error* and *standing*, labeled with the condition $fw > \mathrm{W}_{\mathrm{recover}}$. Again, the value of fw is calculated while the automaton is in the *error* state and the return to *standing* occurs when the value of the human's free will is greater than a predetermined threshold value.

Models of the MVM controller. For modeling the behaviour of the MVM components, we used the Abstract State Machines (ASMs) formal method [10, 11], which is an extension of Finite State Machines (FSMs) where unstructured control states are replaced by states with arbitrarily complex data. ASM *states* are mathematical structures, i.e., domains of objects with functions and predicates defined on them. A *run* of an ASM model is a finite or infinite sequence s_0, s_1, ...s_{i-1},s_i ... of states of the machine, where s_0 is an initial state and the *transition* from state s_i to the next state s_{i+1} is obtained by firing the set of all ASM *transition rules* invoked by a unique *main* rule, which is the starting point of a computation step. The *update* rule, as assignment of the form $f(t_1, \ldots, t_n) := v$, is the basic unit of rules construction, being f a n-ary function, t_i terms, and v the new value of $f(t_1, \ldots, t_n)$ in the next state. By a limited but powerful set of *rule constructors*, function updates can be combined to express other forms of machine actions as, for example, guarded actions (`if-then`) and simultaneous parallel actions (`par`). ASMs are endowed with a set of tools, ASMETA [1], which provides the user with modeling notations and different analysis (V&V) techniques. In particular,

```
main rule r_Main =
par
if state = STARTUP then r_startup[] endif
if state = SELFTEST then r_selftest[] endif
if state = VENTILATIONOFF then r_ventilationoff[] endif
if state = PCV_STATE then r_runPCV[] endif
if state = PSV_STATE then r_runPSV[] endif
endpar
```

Code 1: MVM Controller main rule

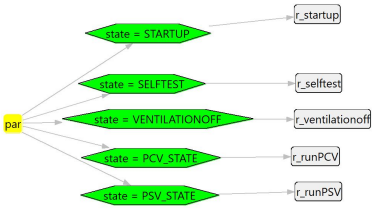


Fig. 5: RDT for the MVM controller

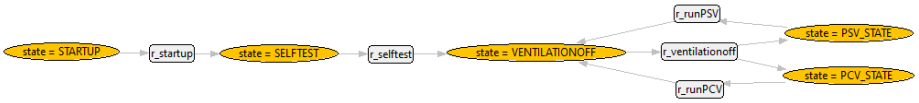


Fig. 6: MVM state diagram

a runtime simulation engine, `AsmetaS@run.time` [30], has been developed within ASMETA as extension of the offline simulator `AsmetaS` [19] to handle an ASM as a living model [5, 34] to run in tandem with a real software system.

MVM model. The *Controller* is the core component of the MVM device. Its model has been developed in ASMs through a sequence of model refinements: (1) The first model describes the transition between the main operation phases: startup, self-test, ventilation off, PCV, and PSV modes. (2) The second model introduces the modeling of inspiration and expiration in both PCV and PSV, (3) while the third model adds further MVM operation features (as the expiratory/inspiratory pauses, the recruitment manoeuvrer, and the apnea). (4) The last refinement step introduces (in both PCV and PSV) the transition between expiration and inspiration in case of pressure drop, and the transition between inspiration and expiration in case the pressure exceeds a threshold.

The ASM model shown in Code 1 refers to the more abstract level and specifies the controller's operation phases: the main rule specifies the transitions among the MVM states by setting the value of the state variable (initialized at the STARTUP value). Depending on the state value, the corresponding rule (not reported here) is executed.

Fig. 5 provides a tree-based graphical representation of the model structure by using an equivalent visual representation for ASMs. This will be exploited for developing a model for reliability evaluation (see Sect. 5).

The semantic visualization of the model, and in particular of the main rule, is shown in Fig. 6. It represents the MVM operation in terms of a control state machine: the value of the variable state is used as state mode to determine machine states.

Models interaction. Having multiple models allows applying analysis techniques to the whole systems, like the simulation of critical scenarios and the proper answer of the overall system.

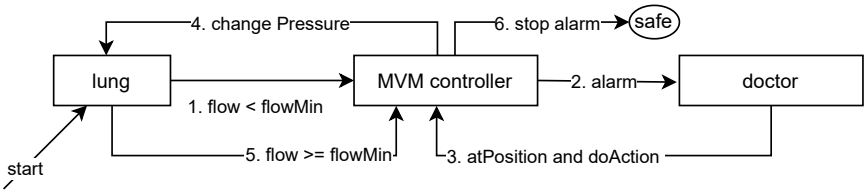


Fig. 7: *Low flow alarm handling* Scenario

For example, consider the **scenario** of *alarm handling due to a low value of breathing flow when MVM is ventilating the patient* (both in PCV or in PSV mode). The data flow among the lung, MVM controller and doctor models is captured by the communication diagram in Fig. 7. The implementation of this scenario, and, therefore, the model co-simulation and the concrete way in which data are exchanged among the models, depends on the specific runtime infrastructure adopted, as described in Sect. 3.

1. The model of lung (see Fig. 3c) reveals that the current value of breathing flow is below a minimum threshold (see condition *flow* < *flowMin* in Fig. 7) and alerts the ventilator.
2. The model of the controller (see Code 1) reveals the unsafe situation and raises an alarm to alert the doctor (rules `r_runPCV` and `r_runPSV` – not shown here – are responsible, each for the relative ventilation mode, for checking this unsafe breathing condition and raising the suitable alarm).
3. The doctor (see the model in Fig. 4) is at the position, he/she is not fatigued and free wills to do the right action; the transition from `standing` to `acting` is performed and the doctor, by the GUI of the MVM, `does the action` of increasing the value of the pressure flow delivered the patent.
4. The increased pressure value is used by the controller model to change the pressure delivered to the patient (rules `r_runPCV` and `r_runPSV` are responsible, in their respective ventilation mode, for performing the pressure change);
5. The model of the lung communicates to the ventilator a flow value above the minimum level (the condition *flow* $\geq$ *flowMin* holds).
6. Finally, the controller stops the alarm (rules `r_runPCV` and `r_runPSV` are responsible, in the respective ventilation modes they manage, to stop the alarms).

5 DT trust analysis

As described in Section 3, the *DT runtime analysis models* layer includes the definition of models and related quality-based if-what and what-if analysis techniques, which will empower this level with the ability to carry experiments that would be costly on a real system. Here we focus on if-what analysis type and provide some examples on how conventional formal analysis techniques can be used for the analysis of a safe interaction between humans and the system. If-what analysis is executed on the DT and allows for detecting violations of the quality and security requirements that may compromise the

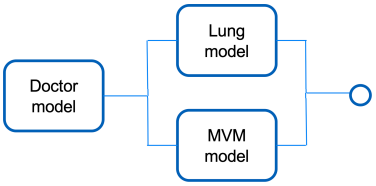


Fig. 8: Reliability block diagram for the MVM DT

quality of the patient's interaction with the system. Hence, changes to the digital models are evaluated and possibly reflected into the MPT.

From a *reliability perspective*, for example, we take into account the DT models we have described so far: (i) lung, (ii) doctor and (iii) MVM. From a reliability point of view, the DT can be computationally represented as a set of components, each representing, from an high-level point of view, one of the three models (i)-(iii), connected as in Figure 8, using a reliability block diagram (RBD) notation [31]. The rationale behind this RBD is that the overall system works when the doctor is working and when at least one between the lung and the MVM components is working. Adopting standard analysis techniques [31], the overall reliability can be computed as:

$$R_{DT} = R_D \cdot (1 - (1 - R_L) \cdot (1 - R_{MVM}))$$

where R_{DT} denotes the overall reliability, R_D is the reliability of the doctor that is computed solving the model in Figure 4 with the UPPAAL tool [14] to obtain its failure probability through stochastic model checking. Specifically, this amounts to determine the probability of the automaton reaching the error state. R_L represents the reliability of the lung, that it can be evaluated by the estimated values of C and R (see Fig. 3c) using a table that helps classifing the condition of the patient as obstructive disease, restrictive conditions, acute situation, or healthy. To each condition, it can be assigned a number in the interval [0,1], where 0 denotes a non functioning lung and 1 a perfect functioning one. Finally, R_{MVM} is the reliability of the ASM modeling the MVM controller. This is computed by exploiting the approach proposed in [28] that considers the internal structure of an ASM and computes its reliability inductively along the call tree of the ASM rules and the structure of the rule bodies.

From a *security perspective*, the analysis on the DT must guarantee that all interactions with the MVM are correct and satisfy the security requirements, even under a cybersecurity attack. In this case, two different analysis scenarios should be considered: (i) a wrong (either intended or unintended) behavior by an authorized user of the system; (ii) an attack by an external actor. The first scenario can be modeled by extending the model of the doctor with new locations and new actions, such as the doctor saving a patient's medical profile on a USB pen drive or selecting the wrong button in the MVM GUI. The second scenario exploits the ASM model of the MVM and the possibility to express distributed *multiple* agents with Abstract State Machines, who can interact synchronously or asynchronously with the other agents. The idea is to model both the doctor and the malicious user as an agent, and to verify by means of model checking if the security requirements expressed in temporal logic are satisfied. The approach is

similar to the one taken for security protocol verification [27], with the attacker trying to exploit the system vulnerabilities to perform an attack, such as using the USB port to upload unauthorized firmware, connecting via wireless communication capabilities and trying to impersonate the doctor, or changing MVM settings using the touchscreen.

6 Conclusion and future directions

We presented the first results in investigating the extent to which it is possible to engineer a DT for a medical device endowed with trust analysis. We envisioned a framework that makes use of a twin model graph for trust analysis that exploits well-known formal analysis techniques. We, in particular, focused on if-what trust analysis and illustrated with the help of the MVM case study a couple of composition analysis examples.

As future work, we plan to explore different research lines. We intend to investigate the DT engineering technology level by experimenting with different existing platforms. Concerning the DT runtime models, we plan to include different model notations and to devise ad-hoc compositional analysis techniques. For trust analysis, we plan to investigate on what-if analysis techniques, and on how to concretely integrate such analysis models and techniques in a systematic way within a DT engineering framework.

Acknowledgment

This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NextGenerationEU.

References

1. ASMETA (ASM mETAmodeling) toolset, https://asmeta.github.io/
2. Functional Mock-up Interface, https://fmi-standard.org/
3. Abba, A., et al.: The novel Mechanical Ventilator Milano for the COVID-19 pandemic. Physics of Fluids **33**(3), 037122 (mar 2021). https://doi.org/10.1063/5.0044445
4. Ahmed, H., Devoto, L.: The potential of a digital twin in surgery. Surgical Innovation **28**, 509–/510 (12 2020). https://doi.org/10.1177/1553350620975896
5. Bencomo, N., Götz, S., Song, H.: Models@run.time: a guided tour of the state of the art and research challenges. Software and Systems Modeling **18**(5), 3049–3082 (2019). https://doi.org/10.1007/s10270-018-00712-x
6. Bersani, M.M., Braghin, C., Cortellessa, V., Gargantini, A., Grassi, V., Presti, F.L., Mirandola, R., Pierantonio, A., Riccobene, E., Scandurra, P.: Towards trust-preserving continuous co-evolution of digital twins. In: 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C). pp. 96–99 (2022). https://doi.org/10.1109/ICSA-C54293.2022.00024
7. Bombarda, A., Bonfanti, S., Gargantini, A., Riccobene, E.: Developing a prototype of a mechanical ventilator controller from requirements to code with ASMETA. In: Proceedings First Workshop on Applicable Formal Methods, AppFM@FM 2021, virtual, 23rd November 2021. EPTCS, vol. 349, pp. 13–29 (2021). https://doi.org/10.4204/EPTCS.349.2

8. Bonfanti, S., Riccobene, E., Scandurra, P.: A component framework for the runtime enforcement of safety properties. Journal of Systems and Software **198**, 111605 (2023). https://doi.org/https://doi.org/10.1016/j.jss.2022.111605
9. Bonivento, W., Gargantini, A., Krücken, R., Razeto, A.: The Mechanical Ventilator Milano. Nuclear Physics News **31**(3), 30–33 (2021). https://doi.org/10.1080/10619127.2021.1915047
10. Börger, E., Raschke, A.: Modeling Companion for Software Practitioners. Springer, Berlin, Heidelberg (2018). https://doi.org/10.1007/978-3-662-56641-1
11. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer Verlag (2003)
12. Camilli, M., Mirandola, R., Scandurra, P.: Runtime equilibrium verification for resilient cyber-physical systems. In: IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2021, Washington, DC, USA, September 27 - Oct. 1, 2021. pp. 71–80. IEEE (2021). https://doi.org/10.1109/ACSOS52086.2021.00025
13. Campbell, D., Brown, J.: The Electrical Analogue of Lung. British Journal of Anaesthesia **35**(11), 684–692 (nov 1963). https://doi.org/10.1093/bja/35.11.684
14. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: UPPAAL SMC tutorial. International Journal on Software Tools for Technology Transfer **17**(4), 397–415 (Aug 2015). https://doi.org/10.1007/s10009-014-0361-y
15. van Diepen, A., Bakkes, T.H.G.F., De Bie, A.J.R., Turco, S., Bouwman, R.A., Woerlee, P.H., Mischi, M.: A Model-Based Approach to Synthetic Data Set Generation for Patient-Ventilator Waveforms for Machine Learning and Educational Use. Journal of Clinical Monitoring and Computing (2022). https://doi.org/10.1007/s10877-022-00822-4
16. Falcone, Y., Mariani, L., Rollet, A., Saha, S.: Runtime failure prevention and reaction. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification - Introductory and Advanced Topics, LNCS, vol. 10457, pp. 103–134. Springer (2018). https://doi.org/10.1007/978-3-319-75632-5_4
17. Fitzgerald, J., Larsen, P.G., Margaria, T., Woodcock, J.: Engineering of digital twins for cyber-physical systems. In: ISoLA 2020. p. 49–53. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-83723-5_4
18. Fuller, A., Fan, Z., Day, C., Barlow, C.: Digital twin: Enabling technologies, challenges and open research. IEEE Access **8**, 108952–108971 (2020). https://doi.org/10.1109/ACCESS.2020.2998358
19. Gargantini, A., Riccobene, E., Scandurra, P.: A Metamodel-based Language and a Simulation Engine for Abstract State Machines. J. UCS **14**(12) (2008). https://doi.org/10.3217/jucs-014-12-1949
20. Heinrich, R., Durán, F., Talcott, C.L., Zschaler, S. (eds.): Composing Model-Based Analysis Tools. Springer (2021). https://doi.org/10.4230/DagRep.9.11.97
21. Huiskamp, W., van den Berg, T.: Federated Simulations, pp. 109–137. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-51043-9_6
22. Jimenez, J.I., Jahankhani, H., Kendzierskyj, S.: Health Care in the Cyberspace: Medical Cyber-Physical System and Digital Twin Challenges, pp. 79–92. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-18732-3_6
23. Kirchhof, J.C., Michael, J., Rumpe, B., Varga, S., Wortmann, A.: Model-Driven Digital Twin Construction: Synthesizing the Integration of Cyber-Physical Systems with Their Information Systems. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. p. 90–101. MODELS '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3365438.3410941

24. Kritzinger, W., Karner, M., Traar, G., Henjes, J., Sihn, W.: Digital twin in manufacturing: A categorical literature review and classification. IFAC-PapersOnLine **51**(11), 1016–1022 (2018). https://doi.org/https://doi.org/10.1016/j.ifacol.2018.08.474, 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018
25. Lestingi, L., Askarpour, M., Bersani, M.M., Rossi, M.: Formal Verification of Human-Robot Interaction in Healthcare Scenarios. In: de Boer, F., Cerone, A. (eds.) Software Engineering and Formal Methods. pp. 303–324. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-58768-0_17
26. Lestingi, L., Sbrolli, C., Scarmozzino, P., Romeo, G., Bersani, M.M., Rossi, M.: Formal modeling and verification of multi-robot interactive scenarios in service settings. In: 2022 IEEE/ACM 10th International Conference on Formal Methods in Software Engineering (FormaliSE). pp. 80–90 (2022). https://doi.org/10.1145/3524482.3527653
27. Lilli, M., Braghin, C., Riccobene, E.: Formal Proof of a Vulnerability in Z-Wave IoT Protocol. In: Proc. of Int. Conf. on Security and Cryptography - SECRYPT,. pp. 198–209 (2021). https://doi.org/10.5220/0010553301980209
28. Mirandola, R., Potena, P., Riccobene, E., Scandurra, P.: A reliability model for service component architectures. J. Syst. Softw. **89**, 109–127 (2014). https://doi.org/10.1016/j.jss.2013.11.002
29. Redelinghuys, A.J.H., Basson, A.H., Kruger, K.: A six-layer architecture for the digital twin: a manufacturing case study implementation. Journal of Intelligent Manufacturing **31**(6), 1383–1402 (2020). https://doi.org/10.1007/s10845-019-01516-6
30. Riccobene, E., Scandurra, P.: Model-based simulation at runtime with abstract state machines. In: Communications in Computer and Information Science, pp. 395–410. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-59155-7_29
31. Signoret, J.P., Leroy, A.: Reliability Block Diagrams (RBDs), pp. 195–208. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-64708-7_15
32. Talcott, C., Ananieva, S., Bae, K., Combemale, B., Heinrich, R., Hills, M., Khakpour, N., Reussner, R., Rumpe, B., Scandurra, P., Vangheluwe, H.: Composition of Languages, Models, and Analyses, pp. 45–70. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-81915-6_4
33. Tao, F., Zhang, H., Liu, A., Nee, A.Y.C.: Digital twin in industry: State-of-the-art. IEEE Transactions on Industrial Informatics **15**(4), 2405–2415 (2019). https://doi.org/10.1109/TII.2018.2873186
34. Tendeloo, Y.V., Mierlo, S.V., Vangheluwe, H.: A multi-paradigm modelling approach to live modelling. Software and Systems Modeling **18**(5) (2019). https://doi.org/10.1007/s10270-018-0700-7
35. Weyns, D.: Software engineering of self-adaptive systems. In: Cha, S., Taylor, R.N., Kang, K.C. (eds.) Handbook of Software Engineering, pp. 399–443. Springer (2019). https://doi.org/10.1007/978-3-642-02161-9_1
36. Yue, T., Arcaini, P., Ali, S.: Understanding digital twins for cyber-physical systems: A conceptual model. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020. Lecture Notes in Computer Science, vol. 12479, pp. 54–71. Springer (2020). https://doi.org/10.1007/978-3-030-83723-5_5