Knowledge driven rapid development of white box digital twins for industrial plant systems

Amar B., Subhrojyoti R. C., Barnali B.
TCS Research, Pune, India
Email: amar.banerjee@tcs.com, subhrojyoti.c@tcs.com, barnali.basak@tcs.com

R. Dhakshinamoorthy, Seenivasan A., Naveenkumar S.
TCS Ltd., Chennai, India
Email: dhaks.r@tcs.com, a.seenivasan1@tcs.com, naveenkumar.subramani@tcs.com

IECON 2021 - 47th Annual Conference of the IEEE Industrial Electronics Society | 978-1-6654-3554-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/IECON48115.2021.9589912

***Abstract*—Industrial systems like power plants need a significant domain and engineering expertise for their efficient operations. Experts make decisions for multiple operational objectives such as minimizing cost, maximizing productivity, ensuring compliance to environmental conditions, etc. However, deciding strategies to achieve an objective can be highly non-trivial as it may lead to conflicting outcomes concerning other objectives. Modern digital twin(DT) technologies make it possible to build systems to support such decision-making to validate critical decisions. DTs are developed as one-off solutions for individual plants in the current practice, requiring high efforts and domain knowledge. Our work demonstrates the use of a knowledge-driven approach to reduce DT development efforts significantly. The main contribution of our work is an end-to-end architecture that allows us to explicate, structure, and reuse domain knowledge to semi-automate DT development for industrial plants. We integrate technologies like semantic web, model-driven engineering, and formal methods to realize the architecture. Our approach to developing a DT supporting a fault management scenario in a power plant reduced 70% development time and 50% manual efforts.**

***Index Terms*—knowledge-driven, digital twins, fault management, FMEA, transition system, state machines, domain-knowledge**

I. Introduction

Complex industrial systems like power plants need critical decision-making abilities to improve operational efficiency. For example, a decision to maximize productivity needs cross-checking against compliance with environmental regulations. However, arriving at such non-conflicting decisions is difficult even for experienced engineers and domain experts. Therefore, using digital twin(DT) technologies to aid such decision-making is common in the current practice.

DTs enable a virtual sand-box environment simulating the behaviour of a plant to perform analyses and predictions. However, the requirements and nature of a DT may vary for different situations and stakeholder needs. Therefore, the development of a DT is a collaborative task involving domain experts and engineers. In the current practice, DTs are developed in an ad-hoc manner relying heavily on the knowledge of domain experts and are carried out based on manual design & development efforts.

DTs can be of various types like white, black, and grey box. There is no one-size-fits-all approach to build DTs of various types. As a result, the approach to develop DTs of each type can be ad-hoc. A white box DT can be a system model developed using the knowledge of the system's structure, processes, and behaviour. The behaviour should get expressed as a discrete automaton, such as a finite state machine (FSM), or a continuous model, such as a physics-based differential equation. Black box twins are modelled based on emergent system behaviour using techniques like machine learning. Finally, grey box refers to an approach that combines white and black box models.

In our work, we focus on streamlining the approach to building white box DTs, which can address a common stakeholder concern such as fault management of a plant, by simulating the plant's discrete behavior[1] to enable reasoning and decision making. As discussed below, we provide an architectural approach that outlines the high-level steps needed to develop such a DT in a semi-automated manner.

1) Identify a common stakeholder viewpoint like fault management which require decision support in an industrial plant. Identify the type of DT that represents the viewpoint well. For E.g. an FSM based DT constructs a viewpoint that resonates with the stakeholder concerned with fault management.
2) Explicate the stakeholder's knowledge about that viewpoint into a knowledge structure that serves as the input to build the corresponding DT. E.g., For the fault management viewpoint, the knowledge elements are the actions and commands dealing with responses, events, alarms, and fault states in a system. This information is critical for developing the DT to aid decisions like taking actions for a specific fault ocurence.
3) Use the viewpoint knowledge to automatically derive an analyzable model of the DT, which can be reasoned and finally translated into the DT implementation.

Tata Consultancy Services Ltd. Research

The objective of our architecture is to design an approach that reduces time and effort for implementing the above steps. We employ semantic web[2], and a knowledge-driven approach (KDA)[3] to partially automate the above steps.

In section 2, we discuss the current development practice and challenges for DTs. Section 3 discusses our approach to capture and organize the domain knowledge using ontologies for stakeholder viewpoints. This section also discusses the role of knowledge to auto-generate a DT for a specific viewpoint such as fault management in a power plant. Section 3 also discusses *finite state-machines*[1] as a formal representation of the DT and outlines a strategy for translating ontological domain knowledge into this formal representation. Section 4 demonstrates our approach and results for a power plant fault-management scenario. Finally, in section 5, we discuss the future work and conclude the paper.

II. State of the art in building digital twins

This section discusses the state of practice in developing digital twins and summarizes the existing gaps in the approaches.

Marmolejo-Saucedo [4] describe a case study to design and develop a digital twin for supply chain process. The design process in the approach uses technologies such as simulators, solvers, and data analytic tools configured by experts from both the technical and business domains. The approach relies on manual effort and experiential knowledge to design the simulators, analytics algorithms, data models, and constraint solvers.

The application of digital twins in fault detection has been studied by Wang et al. [5] for smart manufacturing. The major contribution of the study is the digital twin construction steps:1) describing a digital model using the domain understanding 2) creating data analytics strategies based on the plant objectives and 3) creating knowledge bases from historical data from troubleshooting and diagnostic experts. . The approach depends on historical data, domain experts, offline analysis.

Gabor et al. [6] describe the architecture of digital twins for cyber-physical systems targetting safety engineering. This article describes a three-model approach to conceptualize digital twins. The elements of the three-model architecture are physical, cognitive, and contextual(world) models. The architecture describes the interactions between the three models and how the interactions could get realized with a digital twin. The proposed approach relies on experts from relevant domains who understand the connections across the models.

A knowledge and data-driven approach by Zhang et al. [7] propose a knowledge-driven framework for digital twin manufacturing cells (DTMC). DTMC supports autonomous manufacturing by intelligent perceiving, simulating, understanding, predicting, optimizing, and controlling strategies. DTMC's three critical enabling technologies include the digital twin model, dynamic knowledge bases, and knowledge-based intelligent skills. A use case for a digital twin robot for supporting its autonomous operations validates the claims in DTMC.

Boschert, Heinrich, and Rosen [8] propose the requirements and challenges in building the next generation digital twin, followed by an initial attempt in conceptualizing a next-generation digital twin. The study lists down the significant requirements for the future digital twins like 1) integration of multiple simulation technologies 2) integration of environment knowledge in twin operations and 3) identifying solutions for real-life concerns. Further, the study proposes structuring models and data using official standards, semantic annotations, and algorithms for the digital twin. The above propositions get addressed by employing domain experts and spending manual efforts to develop models, algorithms, and semantic structures.

In summary, digital twins' design and development are highly dependent on domain experts and manual efforts in the current state of practice. It relies on experts to provide the domain knowledge and developers to translate them into DT realizations manually. However, approaches ensuring the utilization of correct and validated domain knowledge towards realizing digital twins in an automated manner are highly desired to reduce their development time and efforts. In our work, we leverage the ideas of semantic web and Model-Driven Engineering (MDE) from the field of computer science to address the above-discussed gaps. We see benefits of using ideas from semantic web and MDE towards building an end to end architecture that a) ensures correct and complete domain knowledge required for DT development, b) ensures correct translation of domain knowledge into computer software, realizing the DT, c) minimizes additional efforts for testing and validating the developed DT, and finally, d) reduces DT development time. We discuss our architecture in detail in the next section.

III. Knowledge Driven Generation of Digital Twin

Figure 1 depicts a high-level solution architecture of our approach. It captures knowledge using a Control Natural Language (CNL)[9]. The knowledge can describe various viewpoints like fault management, environmental compliance, energy flows, etc., for any plant system. In our work, we focus on the fault management viewpoint for a power plant. The *knowledge-base* stores *fault types*, their *properties*, *relations* and *rules* for *fault conditions*. The fault knowledge translates into the structure of a *finite state machine*(FSM) representing the final DT for fault management. FSMs represent the discrete behaviour of plant systems well and provide a formal structure to reason and trace the behaviour of a system, hence making them suitable to represent plant DTs. We

use the definition of transition systems[10] to describe the FSM. A detailed discussion on the definition and representation of transition systems is given in the next subsection.

The translation is a two-step process; in the first step, knowledge gets translated into a *transition system* represented using a *control system model*. In step two, the *control system model* gets translated into an executable representation of the FSM. The FSM-based DT mocks the real-world behaviour for the power plant by receiving operational sensor data. The DT processes sensor-data by applying *fault conditions* and *transitions* to decide suitable *fault states*. The DT can perform analysis like predicting the next fault state, root cause for a specific fault, etc. In this paper, we focus on the architectural approach, leaving out the details about the usage aspects of the DT.

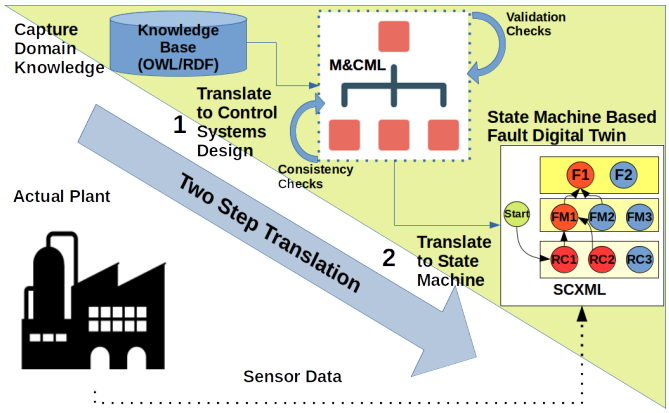

Fig. 1. High level solution architecture

A. Knowledge Base

Knowledge usually gets expressed as ontologies using logic-based languages. Formats like web ontology language(OWL)[2] describe semantic ontologies. It provides capabilities for reasoning, inferencing, querying, and advanced services to develop intelligent applications. However, the syntax for OWL is XML-based, making it difficult for manual specification. Custom GUIs like Protege or XML/OWL editors show usage challenges for describing large ontologies.

We use a controlled natural language called Semantic Application Design Language(SADL)[9] that provides an English-based syntax to describe ontologies. This language serves as a front-end for populating OWL structures and is supported by an Integrated Development Environment (IDE). The IDE provides support for semantic reasoning, inferencing, syntax validations, and correctness. SADL enables domain experts to play a more active and productive role in explicating their domain understanding using simple English sentences.

We first define a base ontology for fault management using SADL as shown in figure 2.

```
Component is a class. sensor_stream_data is a type of decimal.
Sensor is a class described by generated_data  with values of type sensor_stream_data.
//RelationTypes
HAS_FAULT describes Component with values of type Fault.
HAS_SENSOR describes Component with values of type Sensor.

Fault is a class. {FaultTree,FaultMode,RootCause} are types of Fault.

FaultTree is a class described by HAS_FAULT_MODE with values of type FaultMode.

FaultMode is a type of FaultTree described by HAS_ROOTCAUSE with values of type RootCause.

RootCause is a type of FaultMode described by HAS_DATA_RULE with values of type DataRule.

DataRule is a class described by process_data with values of type sensor_stream_data,
described by RULE with values of type ^Equation.
```

Fig. 2. SADL description of base fault ontology

In the base ontology, a *Component* has multiple *Faults* and *Sensors* producing *sensor_stream_data*. *Data_Rules* are conditions to check if a *sensor_stream_data* indicates anomalous behaviour. If *Data_Rule* is satisfied, then the *Component* is assigned a *RootCause Fault*. Combination of two or more *RootCauses* give rise to a *FaultMode*. Multiple *FaultModes* can be combined to create a *FaultTree*. The base ontology gets instantiated to describe specific contextual scenarios as instances of *Components, Faults, FaultTrees, FaultModes, RootCauses* and *Sensors* for a given system. This ontology can be modified based on different scenarios and needs of the DT. In summary, the fault ontology provides for the vocabulary to specify requirements of the fault management DT.

B. Translation from knowledge to DT model

Knowledge captured using SADL gets translated to an FSM-based DT using a two-step mechanism. The knowledge first gets converted into an abstract *control system model*.The *meta-model* for the *control system model* can be referred from M&CML description [11]. We discuss the rationale behind the usage of the *control system model* in the following subsection. The *control system model* gets transformed into a executable FSM representation of the DT. The two-step translation process creates checkpoints at each step to ensure validation and completeness of knowledge for the creation of DT. The validation and completeness checks could be manual or automated. Such validations during the translation process reduce the dependency on manual testing post-development phases. The translation can be further generalized using a multi-step process requiring a chain of translated models to be created and checked.

1) Translating Knowledge to Control Systems View: Control systems manage plant operations to perform device control, data sensing, fault handling, etc. Translating knowledge to a *control system model* enables a validation mechanism for the captured domain knowledge against the control systems domain and vocabulary. Any gap or inconsistency in the translated *control system model* implies gaps in the knowledge source that needs to get updated accordingly.

To define a formal *control system model* we refer transition systems[10]. Transition systems serve as the formal theory for FSMs and provide semantics to define a

machine with the desired behavior. A transition system S is a six-tuple $(X_S, X_S^0, U_S, \underset{S}{\rightarrow}, Y_S, h_S)$ where X_S is the *state space* of S, $X_S^0 \subseteq X_S$ is the set of *initial states* of S, U_S is the set of *inputs* or *actions* of S, $\underset{S}{\rightarrow} \subseteq X_S \times U_S \times X_S$ is the *transition relation* of S, Y_S is the set of *observable outputs*, and $h_S : X_S \rightarrow Y_S$ is the *view function* of the system. $(x, u, x') \in \underset{S}{\rightarrow}$ is written as $x \xrightarrow[S]{u} x'$. We assume all the systems in consideration are non-blocking (for every state x and input u there is at least one state x' such that $x \xrightarrow[S]{u} x'$.

However, the transition system model is theoretical and needs to be aligned or mapped with the control systems model to make the translation possible. We use M&CML [11] to capture the *control system model* using the vocabulary of the control systems domain. M&CML allows the description of controllers with concepts like *Commands-Responses, Events, Alarms, DataPoints* representing inputs and outputs for the controller. The *OperatingStates, Transitions, Entry-Exit Actions* represent the behaviour of the control system. M&CML builds the semantics of transition systems into it, making it possible to represent translated knowledge as formal FSMs.

Algorithm 1 translates the fault knowledge to the *control system model* in M&CML. Knowledge queries serve as input to the algorithm. Knowledge queries fetch knowledge elements from the captured fault knowledge based on conditions. E.g., *"select elements of type **Fault**"* is an example of a knowledge query. Queries are described using SPARQL[12] syntax and get mapped to the control system elements to populate the *control system model*. The algorithm maps the knowledge elements from the queries to the elements in the control systems model.

The populated model can now be checked for completeness and correctness against the *control system viewpoint*. E.g. if the *control systems model* is incomplete like missing *next-state* in a *transition*, we can infer that the knowledge provided for translation is incomplete. Similarly, if there is any correctness issues in the *control system model* like *cyclic-FSM*, we can conclude that the knowledge is incorrect from the *control systems viewpoint*. The *control system model* serves as an intermediate representation(IR) before it is translated into the FSM-based DT model, discussed in the following subsection.

2) Translating control systems model to FSM model: As discussed, the *control systems model* helps validate and reason for the consistency and correctness of the captured knowledge. However, the model itself is not executable and hence cannot get directly used as a DT. Hence, this needs translating the *control system model* to an executable representation. We use the SCXML framework [13], a Java-based state-machine engine, to represent FSMs using an XML-based configuration format and execute it. The SCXML format is a W3C standard for representing FSMs and serves as an executable format.

Algorithm 1: Translation Algorithm

Input: *Queries*
Output: *cntrl_mdl*

```
/* instantiate a new control system model     */
set cntrl_mdl ← new ControlModel
for query : Queries do
    knowl_ele ← query.execute()  // get knowledge
    for ke : knowl_ele do
        /* check type of instance              */
        if ke is a Fault then
            if ke is a FaultLevel and ke.level ==0 then
                cntrl_mdl.initStates ←
                  new State(ke.name)
            else
                fState ← new State(ke.name)
                trans ← new StateTransition
                trans.rules ← ke.Rules
                trans.nextStates ← ke.NextFault
                fState.transitions ← trans
                cntrl_mdl.states ← fState
        if i instanceof SensorData then
            cntrl_mdl.data ← ke.sensor_data
return cntrl_mdl
```

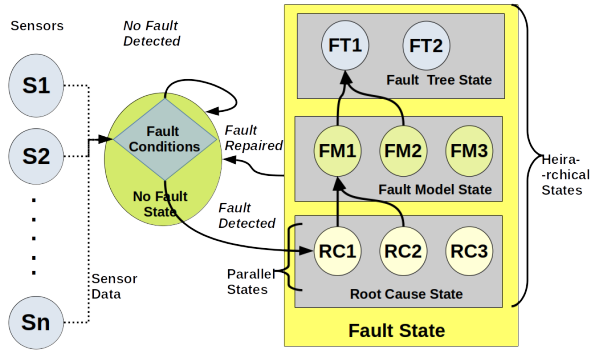

Fig. 3. Finite state machine model for fault management digital twin

Figure 3 shows the FSM structure to which the control systems model gets translated. We implement Xtend [14] based translator templates to translate M&CML model to SCXML FSM configuration. The fault FSM starts with an initial *NoFault* state when there are no faults. The *NoFault* state processes data from the deployed sensors as per the fault rules. Any violation of the rules results in the FSM transitioning to the *Fault* state. The *Fault* state is composed of hierarchical and parallel states. Each level in the hierarchy represents a fault type. Each level has multiple parallel fault-states for the instances for that particular fault type. The arrows represent fault propagation as state transitions. This FSM structure enables parallel detection of multiple faults and captures their

traces.

IV. Digital Twin Usecase & Results

We evaluate our approach by addressing a power plant system's failure mode and effect analysis (FMEA) use case. The use-case consists of multiple *Components* having different types of *Faults* as described in the base fault ontology. Based on the fault ontology in figure 2, we capture the contextual knowledge about faults in the power plant. Figure 4 shows the instances of various fault types and the fault detection rules for a *Boiler Component*. The different fault instances and rules get described using SADL and stored as fault knowledge.

```
Boiler is a Component. Boiler HAS_SENSOR A_Economizer_Outlet_O2_Analyzer_Average.

FaultTree_MainSteam_Temp is a FaultTree. Boiler HAS_FAULT FaultTree_MainSteam_Temp.

FaultTree_MainSteam_Temp HAS_FAULT_MODE Main_Steam_Temperature_Low.

Main_Steam_Temperature_Low is a FaultMode.
 //RootCause instances
{Low_AH_Performance,Excessive_Super_Heater_Spray_Flow} are instances of RootCause.
Main_Steam_Temperature_Low HAS_ROOTCAUSE Excessive_Super_Heater_Spray_Flow,
 HAS_ROOTCAUSE Low_AH_Performance.
 //DataRule instances
 AH_Seal_Leakage is a DataRule.
 Low_AH_Performance HAS_DATA_RULE AH_Seal_Leakage.
   //Rule Defenition
Rule AH_Seal_Leakage_Rule: if ((valve_pressure < 0.8 or vent_pressure <0.8) and
   flow_rate > 950) then Main_Steam_Temperature_Low HAS_ROOTCAUSE Low_AH_Performance.
```

Fig. 4. Describing fault instance knowledge using SADL

The captured domain knowledge gets translated into a control systems model represented using M&CML. We built a custom DSL to describe the translation algorithm and the mappings between the fault knowledge and control systems model, as shown in the figure 5.

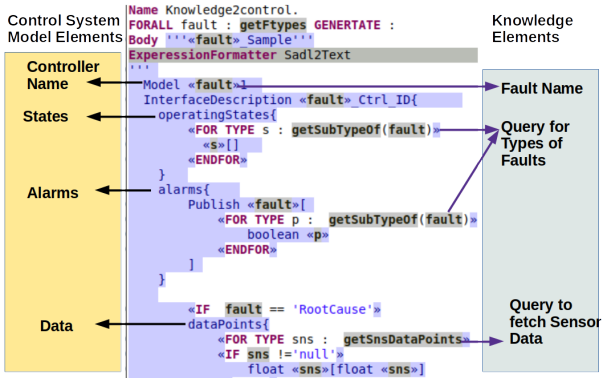

Fig. 5. Translation of fault knowledge to control systems model using DSL

The control systems model translates to SCXML FSM configuration using the Xtend-based translation templates. The generated SCXML based digital twin serves as a critical building block of the decision support system for fault management of the power plant. The generated SCXML-based DT gets deployed as a Java-based service in a webserver. The DT service may get used for various analyses like root cause detection, fault propagation scenario, prediction of upcoming faults etc. The deployment and usage details of the DT is out of scope for this paper.

```
<parallel id="FailureMode">
    <state id="Main_Steam_Temperature_Low" initial="Main_Steam_Te
        <state id="Main_Steam_Temperature_Low_Initial" >
            <transition cond="Main_Steam_Temperature_Low" target=
        </state>
        <final id="Main_Steam_Temperature_Low_data">
            <onentry>
            <log label="Fault_Main_Steam_Temperature_Low" expr="M
            </onentry>
        </final>
    </state>
    <transition target="FMEATree"/>
    <onexit>
        <assign location = 'FmeaTree_MainSteam_Temp_Low' expr = '
    </onexit>
</parallel>
```

Fig. 6. Generated SCXML description for fault management digital twin

The usage of service-based architecture makes it easier for consumers to interact with the DT as a service.

A. Evaluation & Results

We evaluate our approach by comparing it against the previously taken approach by our engineering team to carry out a similar FMEA analysis for a specific customer from the power plant domain. We understand that the general approach for performing FMEA is to collect details about plant faults and rules for their occurrences with the help of domain consultants who capture the information using spreadsheets. Spreadsheets make it difficult to carry out domain specific reasoning[15] by constructing integrated views of a system. The inputs in spreadsheets then get handed over to the engineering team, who manually translates the information into multiple software programs. For each program, the faults, their data sources and occurrence rules get hand-coded in Python-based scripts. As part of our experiment, we gather the metrics like the time taken by the engineering team to develop the code, the number of engineers involved, and lines of code written per digital twin that corresponds to an FMEA use case. Further, we record the same metrics from the use of our approach. Table I shows the comparison of the manual approach against our approach.

SN.	Fault Digital Twin Usecase	Hours	Dvlprs	LoC
Manual Effort for Digital Twin Construction				
1	Main Steam Temperature	18	2	500
2	Boiler Pressure	12	2	300
3	Air Supply in Plant	6	2	300
Knowledge Driven DT Generation				
1	Main Steam Temperature	3	1	220
2	Boiler Pressure	2	1	140
3	Air Supply in Plant	1	1	90

TABLE I
MANUAL VS KNOWLEDGE-DRIVEN DEVELOPMENT OF DT

As seen from the table, our approach outperforms the manual development of DT by almost reducing 70% time, 50% manual effort, and 60% lines of code (LoC). The translation approach to auto-generate the

FSM-based digital twin significantly reduces time and the need for manual coding and review, if needed. Early validation of the generated *control system model* also saves time and effort from post-development testing. A key reason for achieving effort efficiency is the reuse of knowledge in various use-cases. Once the knowledge gets captured for a particular use case, the same knowledge becomes available for other scenarios. The reduction in LoC is mainly because of reusing the off-the-shelf SCXML framework instead of implementing the code using programming languages, ensuring solution robustness, as the framework used is already well tested. We tested our knowledge-driven approach for three FMEA cases of the power plant. As reflected by the above comparison, we observed that our approach could be scaled up to auto-generate digital twins for similar needs with significantly reduced time and effort. Usage of utilities like SADL, M&CML, and SCXML framework may add additional training time and efforts on users, and we would address that going forward.

V. Conclusion & Future Work

In this paper, we discussed the role of DTs in making critical decisions for operating mission-critical plants. The current practice for developing DTs is highly dependent on domain experts and requires significant manual efforts. It is, therefore, highly time-consuming to build DTs for large and complex plants. We discussed a knowledge-driven approach for automatically generating a DT for fault management. First, the fault knowledge gets captured using a controlled natural language. Next, the captured knowledge translates into a finite state machine, representing the final DT. The translation is a two-step process; first, the knowledge gets translated into a transition system described using a *control system model*. Second, the *control system model* gets subsequently converted into an executable FSM description, which implements the required DT.

We demonstrated the application of our approach for generating a digital twin for an FMEA use case in a power plant. Our approach significantly reduces the time and effort to construct similar digital twins and is efficiently scalable for large and complex systems. Some of the challenges faced during this research were, identifying a generic schema to capture domain knowledge, designing the translation architecture, and reasoning about the need for an intermediate control systems model.

Going forward, we will apply our approach in generating digital twins for more stakeholder viewpoints like induration process, emission compliance, etc. We want to apply this approach to other plant systems and enhance it to support black, white, and grey box DTs. We would also extend the scope of our approach for addressing continuous and physics-based behaviour in industrial plants.

References

[1] Markus Skoldstam, Knut Akesson, and Martin Fabian. "Modeling of discrete event systems using finite automata with variables". In: *2007 46th IEEE CDC*. IEEE. 2007, pp. 3387–3392.

[2] Deborah L McGuinness, Frank Van Harmelen, et al. "OWL web ontology language overview". In: *W3C recommendation* 10.10 (2004), p. 2004.

[3] Matthias Glawe et al. "Knowledge-based Engineering of Automation Systems using Ontologies and Engineering Data." In: *KEOD*. 2015, pp. 291–300.

[4] Jose Antonio Marmolejo-Saucedo. "Design and Development of Digital Twins: a Case Study in Supply Chains". In: *Mobile Networks and Applications* 25 (2020), pp. 2141–2160.

[5] Jinjiang Wang et al. "Digital Twin for rotating machinery fault diagnosis in smart manufacturing". In: *IJPR* 57.12 (2019), pp. 3920–3934.

[6] Thomas Gabor et al. "A simulation-based architecture for smart cyber-physical systems". In: *2016 IEEE ICAC*. IEEE. 2016, pp. 374–379.

[7] Chao Zhang et al. "A data-and knowledge-driven framework for digital twin manufacturing cell". In: *Procedia CIRP* 83 (2019), pp. 345–350.

[8] Stefan Boschert, Christoph Heinrich, and Roland Rosen. "Next generation digital twin". In: *Proc. tmce*. Las Palmas de Gran Canaria, Spain. 2018, pp. 209–218.

[9] Andrew Crapo and Abha Moitra. "Toward a unified English-like representation of semantic models, data, and graph patterns for subject matter experts". In: *IJSC* 7.03 (2013), pp. 215–236.

[10] Paulo Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[11] Puneet Patwari et al. "M&C ML: A modeling language for monitoring and control systems". In: *Fusion Engineering and Design* 112 (2016), pp. 761–765.

[12] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. "Semantics and complexity of SPARQL". In: *ACM TODS* 34.3 (2009), pp. 1–45.

[13] Jim Barnett. "Introduction to SCXML". In: *Multimodal Interaction with W3C Standards*. Springer, 2017, pp. 81–107.

[14] Klaus Birken. "Building code generators for DSLs using a partial evaluator for the Xtend language". In: *ISoLA*. Springer. 2014, pp. 407–424.

[15] Jácome Cunha et al. "MDSheet: A framework for model-driven spreadsheet engineering". In: *2012 34th International Conference on Software Engineering (ICSE)*. IEEE. 2012, pp. 1395–1398.