EVALUATING OPEN-SOURCE TOOLS FOR HETEROGENEOUS MODEL-BASED DIGITAL TWIN DEVELOPMENT: A MICROBREWERY CASE STUDY

Ander Lee[a], David A. Manrique Negrin[a], and Loek Cleophas[a]

[a]Eindhoven University of Technology, the Netherlands
a.lee@student.tue.nl
{d.a.manrique.negrin,l.g.w.a.cleophas}@tue.nl

ABSTRACT

Digital Twins (DTs) are composed of a physical entity, a connected virtual entity comprised of heterogeneous components (i.e., simulation models and data sources), and specific services built on top of these entities. DT developers' challenges are integrating and orchestrating its components to build such services. Integration concerns the encapsulation and communication among the components, while orchestration focuses on execution aspects. In this study, we examine the potential for integration and orchestration of three frameworks suitable for DT design, with heterogeneous components, and paired with open-source tools tailored to support them. A microbrewery DT was built as a case study, supporting two services. Our findings reveal that while the frameworks facilitate DT development, they only partially address requirements for integration and orchestration. Particularly, the evaluated tools are complex to use to define orchestration, while integration support is limited. Consequently, such open-source tools benefit from an orchestration approach to reduce complexity and increase integration support.

Keywords: Digital twin, orchestration, integration, open-source frameworks, feature assessment.

1 INTRODUCTION

The inception of Digital Twins (DTs) can be attributed to the work of Grieves [1]. Our definition of DTs follows the conceptual model of Tao et al. [2]. Their model encompasses five components: physical entity (PE), virtual entity (VE), digital data (DD), services (Ss), and connections (CN). The VE consists of behavioral cross-domain software artifacts (i.e., simulation models), and the data from the PE. By combining data from PE and simulation models from VE, Ss can be built on top. This characteristic popularized DTs, showing their potential to enhance decision-making, optimization, and innovation.

A major challenge in DT implementation is the integration and orchestration of heterogeneous components such as simulation models and data sources [3]. Integration involves interface generation for inter-model communication, while orchestration manages the models' interaction and data exchange needed to deliver the intended services. Tools that facilitate these tasks should have features to support these activities to their successful implementation, given their complexity. Previous studies have investigated technologies enabling the development of DTs [4, 5]. However, to our knowledge, there is an absence of research focused on comprehending the essential integration and orchestration features necessary for tools to facilitate the development of DTs. Hence, our first research question is *RQ1. What are the essential features of a tool to ensure successful integration and orchestration in a DT*?

In addition, open-source (OS) tools play an important role in adopting new technologies because they facilitate their use in development [6]. OS tools furthermore are extensible and adaptable by their community,

Proc. of the 2024 Annual Simulation Conference (ANNSIM'24), May 20-23, 2024, American University, DC, USA
C. Ruiz-Martin, B. Oakes and R. Cárdenas, eds.

which is advantageous in research settings. On the other hand, research analyzing tooling features for DT development is related to commercial tools such as MindSphere, MATLAB, ANSYS Twin Builder, or Predix [7, 5]. We know of no such research related to OS tooling—including on the features of such tools for models' and tools' interaction and integration. Thus, our second research question is *RQ2. How robust is the feature support offered by OS tools to facilitate integration and orchestration in DTs?*

This paper aims to contribute to existing research by (1) detecting the essential features of tools to integrate and orchestrate a DT with heterogeneous components and (2) systematically analyzing and comparing three OS tools for DT development. By evaluating their features, we provide a comprehensive overview to aid researchers and practitioners in selecting tools for their use cases. Our research focuses on two development tasks: (1) the integration of heterogeneous models and (2) the design of the interaction and data exchange between such models, called model orchestration. In this research, we developed a case study to help us evaluate the tools: a microbrewery, characterized by its intricate interplay of chemical, thermodynamics, and biological processes. It presents a good context to investigate the efficacy of OS tools in DT development.

2 RELATED WORK

The effectiveness of a tool in supporting tasks relies on its features, which are essential for evaluating its capability. If a tool lacks a necessary feature, its effectiveness for that task decreases. This is particularly challenging in emerging technology paradigms like DTs. Previous studies have mainly focused on documenting the technologies and tools utilized in DT development.

Ademenko et al. [7] evaluated commercial tools such as MindSphere (Siemens), ThingWorx (PTC), and Predix (GE) for developing DTs. They list general features for the comparative evaluation of such tools for creating DT applications, including features such as user interface, data structure, documentation, and platform structure related to the application and its services. While they thus have clearly defined features for a systematic evaluation, they only consider commercial tools, and the features are rather general. In contrast, we focus on features required to build DTs with heterogeneous environments.

Qinglin et al. [5] describes technologies and tools that have enabled the development of DTs from 2017 to 2019. It mentions commercially used tools including the ones previously mentioned. It also lists features that each tool should have to facilitate DT development. Examples of such features are simulation, optimization and platform services, data transmission, and data collection. Although the paper has a comprehensive analysis of the features, they are also at a high level.

Gil et al. [8] made a survey related to OS frameworks for DTs. This survey analyzed 14 tools that meet their selection criteria and formulated a taxonomy to evaluate their advantages and limitations. The selection criteria have three main filters: 1) the framework claims to be for DTs, 2) it has a valid OS repository, and 3) its documentation is sufficient. In addition, its evaluation taxonomy has 10 features, inspired by the ISO 23247:2021: communication, storage, support for analytics, compositionally, support for physical intervention, scalability, standardization, reproducibility, interoperability, and community support. In contrast with our study, the scope of this survey is broader and does not particularly focus on DT with heterogeneous components and its features to ease their development.

Tao et al. [4] conducted a systematic search on DT modeling and the technologies facilitating their development using model-driven methods. Their study considers various aspects, including DT applications, classification of DT modeling, and the enabling tools and technologies for DT implementation. Notably, they addressed the integration of heterogeneous modeling technologies, referred to as *model fusion*. However, they primarily discusses how existing integration technologies such as FMI [9] are insufficient. Their study does not delve into the requirements or essential features necessary for achieving successful integration.

Finally, Kirchhof et al. [10] present a model-driven approach to integrate DTs. Specifically, their method focuses on facilitating the integration of PE and VE, and the mapping of data streams between them. The study identified requirements necessary for validating and generalizing their approach. However, we note that these requirements primarily address the heterogeneity of data between the PE and VE, rather than focusing on the heterogeneity of models within the VE. In addition, these requirements are still high-level.

As previously mentioned and in contrast to the above works, the motivation for our research is to analyze OS tooling for DT development, a research that is currently limited, addressed only in [8]. In addition, we aim to tackle the specific research gap related to *heterogeneous* models' integration and orchestration [3].

3 CHOICE OF OS FRAMEWORKS FOR DT DEVELOPMENT

To select the tools for this study, we consider three characteristics: 1) the tool has been used to build DTs, 2) it is open-source, and 3) it has an explicit approach to orchestrate its components (i.e, models and data). We define an explicit approach to orchestrating components as a method in which the developer explicitly defines the execution configuration and sequence of each component. An example of an explicit definition of component orchestration involves specifying the precise sequence of execution for each component, along with their respective start and stop times. We consider an explicit orchestration approach necessary due to the distinct execution or configuration requirements inherent in each service of the Brewery DT, and more generally any DT that aims to have multiple services re-using models and data sources.

The tool search aimed to identify implementations utilizing open-source tools and employing either model-based orchestration method or another explicit orchestration approach. Using Google Scholar, two queries were performed: one with the keywords "digital twin + implementation + platform + orchestration", and the other with "digital twin + model-based + platform + implementation". Papers related to practices or surveys were excluded; we focused solely on those reporting DT implementations. Only papers utilizing open-source tools were considered. Subsequently, the tools were analyzed to identify their orchestration approach, with only those employing an explicit method considered for inclusion in the study. The selected three candidates for the study are explained in the following subsections.

`TwinOps:` `TwinOps` is a framework combining DevOps and Model-Based code generation practices [11]. Its workflow has automated transformations referred to as *Model-to-Code-to-Target CI/CD*, illustrated in Figure 1. We see the `TwinOps` workflow as having four major components: (1) the **Model integrator** in which models are built and configured, including defining their interfaces; (2) the **CI/CD engine** orchestrates the models, collects data from sensors, and executes services such as model validation or data processing; (3) **Containerization** is a hub that stores versions of models' artifacts for the targets; and (4) **Targets** which refers to the deployment of the DT virtual entity and its connectivity to and from the physical entity.

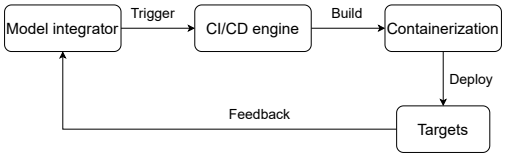

Figure 1: `TwinOps` workflow.

The `TwinOps` framework is not bound to specific tools, thus developers are free to choose their tools as long they follow the workflow depicted in Figure 1. In selecting the tools to use within `TwinOps`, we opted for tools which have been used previously in other DT development. For the **Model integrator**, we selected OMEdit [12], which is a graphical interface for editing Modelica models including FMIs. This OpenModelica tool has been used for developing DTs in [13, 14]. For the **CI/CD engine**, we have selected GitHub Actions [15]. These tools are configured using a pipeline approach, using the markup language

YAML [16]. GitHub Actions have been used for DT development in [17]. For the **Containerization**, we used the tool Docker [18], which is used in [11]. Finally, the **Targets** are the Raspberry Pi, as a gateway to the sensors and actuators, and a Windows computer, to execute the models.

`ThingsBoard:` `ThingsBoard` is an open-source platform that enables rapid development, management, and scaling of IoT projects [19], based on a Service Oriented Architecture. Figure 2 shows the corresponding workflow. In the study in [20], `ThingsBoard` served as the platform for the creation of DTs.

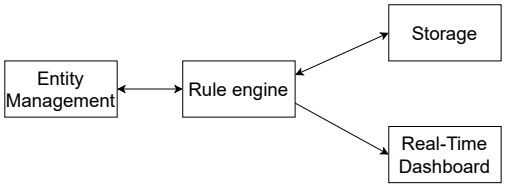

Figure 2: `ThingsBoard` workflow, with arrows indicating modification direction between stages.

`ThingsBoard`'s workflow has four components. First, **Entity management**; in this component all the entities and their relations (e.g., communication between models) are defined. An entity can be a device, which is a basic unit that transmits or receives data (e.g., sensor or models), or an asset that defines the relations among all the involved entities. Second, the **Rule engine** refers to the *rule chains* which define actions to execute when an entity receives data. To define a rule the developer has to learn the semantics of the rule engine, which uses rule nodes that can be functions of conditions and their relationships. Third, the tool supports **Storage**, either internally or through external storage. Fourth, the **Real-time dashboard** is a built-in user interface, where widgets can be added in a graphical form (e.g., charts).

`Ptolemy II:` `Ptolemy II` is a tool based on an actor-oriented architecture. It aims to support the analysis of, and experimentation with CPSs. It was applied for the development of DTs in [21]. For our research, we are interested in two of its components. First **actors**, an actor is an encapsulation of an entity such as a model or a sensor, and which communicates with other actors through messages [22]. Second, **directors** are responsible for defining the execution sequence (control flow) of the actors and their data (data flow). The directors are the orchestrators in `Ptolemy II`. Each director is implemented as an instance of a particular model of computation (MoC). `Ptolemy II` has nine MoC types implemented [22]. It is worth mentioning that the MoCs implemented in `Ptolemy II` have restrictions, including on combining them [22]. This was not an issue in the DT exemplar of our paper, but is a limitation [21].

Figure 3 shows an example with three `Ptolemy II` components. In it, the director defines the actors' execution. The actors communicate with each other via ports, exchanging data encapsulated as tokens.

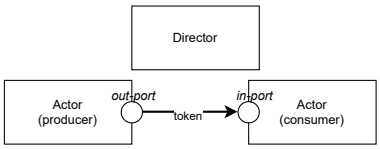

Figure 3: `Ptolemy II` components.

4 BREWERY DT AS CASE STUDY

This section motivates selecting a Brewery DT as a case study, and explains its components. In doing so, we follow Tao et al.'s [2] conceptual 5D model.

To evaluate the selected DT design tools, we build a DT with those five components. As the development of a DT typically requires substantial domain knowledge, we selected a domain in which one of the researchers

has practical and theoretical experience. In addition, its PE should be quickly buildable at limited cost. For these reasons, we selected a beer brewery DT. The conceptual model of Tao et al. is sufficiently general to cover DTs for bio-manufacturing processes [23].

The Brewery DT is defined to cover the fermentation process of the beer. A schematic representation of the DT is shown in Figure 4. Its PE is the fermenter that has three sensors: a Tilt hydrometer, which measures

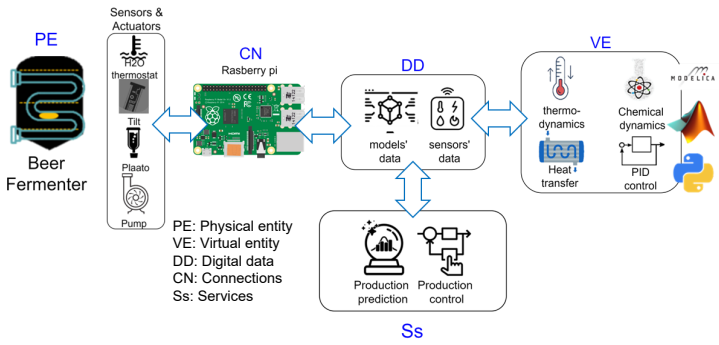

Figure 4: Brewery DT components.

the temperature and amount of alcohol of the beer; a Plaato sensor, which measures the CO_2 produced and the ambient temperature; and a cooling water thermostat. It also has one actuator: a cooling water pump controller. The sensors and actuators are represented on the left of Figure 4. The VE, depicted on the right of Figure 4, consists of four behavioral models: fermentation chemical kinetics, reaction thermal dynamics, heat transfer of the fermenter with the environment, and a PID (Proportional–integral–derivative) controller. The models were built in different modeling engines (i.e., Modelica, Python, and Matlab) to have a heterogeneous simulation modelling environment, because DTs require heterogeneous models to address different aspects of a system [24]. The DD is produced by the sensors and the models, and depicted in the middle of Figure 4. To connect (CN) the sensors and actuators with the VE, we used a Raspberry PI as a gateway, as all the sensors and actuators have an API that is compatible with it. Finally, the services (Ss) are *Production prediction* and *Production control* as shown in Figure 4 on the bottom; these are discussed next.

5 RESEARCH METHOD

This section describes the method followed to analyze the open-source tools. It is divided into three steps. First, DT service selection; second, feature identification; and third, linking service evaluation to case study.

5.1 DT service selection & requirements

We based our services selection based on Udugamat et al [23], who state that a DT of a bio-manufacturing process has five maturity levels: 1) steady-state model, 2) dynamic model, 3) validated model, 4) digital shadow (DS), and 5) model-based control. We decided to follow these maturity levels to develop our DT. The first three maturity levels were satisfied by creating the brewery fermentation model and validating this with the behavior of the fermenter. To satisfy the fourth level and fifth levels, we selected two services. The service *Production prediction* (S1) aims to predict future alcohol production considering the current alcohol and fermentable sugar states, which we can measure using our DT's sensors. Moreover, service S1 aims to enable analysing alcohol production, facilitating the comparison between the "actual" production versus the "ideal" production, at different stages of fermentation. This service satisfies the fourth maturity level by receiving data from the PE but not communicating back; and hence satisfies Udugama's DS definition. The second service we selected is *Production control* (S2), which uses water cooling to control fermentation

temperature and hence optimize alcohol production. This service satisfies [23]'s last maturity level. To define the DT services we have elicited a set of requirements, which we grouped into three categories. The first category covers general requirements, not specific for any service but related to integrating and orchestrating systems containing heterogeneous models and data sources: GR1. Support the integration tasks (i.e., model interfacing & data exchange) of the heterogeneous models & data from all sources (i.e., sensors, actuators & database), GR2. Support the definition of the orchestration (i.e., model configuration & workflow) for each service of the DT, and GR3. Provide optimal execution of diverse types of models and data exchange among them. The second category is related to the **production prediction service** requirements: S1R1. Include a data storage component and retrieval mechanism to handle the data generated by the models in each simulation time-step, and S1R2. Should support an effective mechanism to define an initial model configuration & define a workflow to automatically re-configure models during run-time.Finally the third category related to the **production control service** requirements: S2R1. Should optimally execute the control workflow, from the signal detection, process, model execution, and control action and S2R2. Should have a mechanism to track origin and destination of the data in each component.

5.2 Feature identification

To define what features to evaluate, we conducted a literature search of DT development requirements. We aimed to identify what features are required for VEs with heterogeneous models, particularly related to the models' integration and orchestration. We again used Google Scholar to conduct our literature search, focusing on digital twin (DT) development in industry, particularly those with simulation models and employing model-based techniques. Our search query was: "digital twin + simulation modeling + development + industry application" We specifically targeted literature across various domains that discussed DT development, revealing system requirements and properties. Papers focusing solely on individual DTs were excluded. Additionally, we searched for common practices for developing systems with heterogeneous models, using the query "heterogeneous modeling + model-based integration". We aimed to find literature related to current practices in achieving interoperability with such models. We prioritized papers containing explicit requirement elicitation or properties necessary for the successful implementation of heterogeneous modeling systems. The features we identified are shown in Table 1 and answer our RQ1.

Table 1: Integration (I) and orchestration (O) features for DTs.

ID	Feature	Reference
Integration		
I1	Interface generation	[25, 26]
I2	Modularity	[27, 28]
I3	Data exchange consistency	[29, 26]
Orchestration		
O1	Configurability	[25, 27]
O2	Workflow automation	[30, 28, 31]
O3	Model diversity management	[25, 24]

Note that Table 1 does not result from an exhaustive investigation of integration and orchestration requirements, but shows what we consider the minimum tool features required for these development activities. *Interface Generation (I1)* refers to a systematic method, supported by the tool, to generate 'glue code' to combine models. *Modularity* refers to the decomposition of the models and the capability of tracing data from the DT's components. *Data exchange consistency* refers to semantic and syntactic consistent data transfer between components. *Configurabilty* considers the model parameters adjustability. *Workflow automation* considers the scheduling of model execution: trigger definition, steps of execution, and algebraic

loop solving. Finally, *Model diversity management* is the time synchronization among different models; e.g. merging discrete-time and continuous-time models or signaling between timed and untimed models.

5.3 Linking feature evaluation to case study

We assess the efficiency of each of the tools to ease the implementation of the DT by realizing the requirement identified in Section 5.1. To facilitate a comparative analysis of the selected tools, we establish an evaluation criteria, referred to as Key Performance Indicators (KPIs), to effectively assess the fulfillment of each requirement. The defined KPI are enlisted in Table 2. In addition, this Table contains the mapping between the KPIs, the requirements, and the features of Table 1.

Table 2: KPI mapping to the integration and orchestration features.

Service	ID	Units	KPI	Req	Integration	Orchestration
General	LoA	%	Level of automation	GR1,GR2	I1,I3	O1,O2
	UpD	s	Updating delay	GR3	I3	O3
S1	HiD	bool	Data history	S1R1	I1,I3	O2
	Adap	bool	Adaptability	S1R2	I1,I2	O1,O2
S2	Tra	level	Traceability	S2R1	I3	O2,O3
	Lat	ms	Latency	S2R2	I2, I3	O2

In addition, developers have execution time requirements for the models and data transfer. We call this KPI the *updating delay*. For this KPI we measured the execution time from the moment the model received data to start processing it, to the models' output generation. First, a tool should support the automation of tasks such as model-to-model communication, data exchange consistency, initial and run-time configuration of models, and workflow definition. We define the *level of automation* as the *percentage of steps that is automated* among the steps needed to define such tasks. E.g, if a tool supports all four task's automation as previously explained, except for run-time configuration, then the level of automation would be 75%.

For the *Production prediction* requirements, we defined two extra KPIs. First, the service requires to store and have access to the historical data produced by the VE. This data is used to compare actual and predicted data, which enables the production analysis. Hence, the KPI *historical data* is defined as the availability of historical data for the service. This requires support on data exchange, interface definition, and workflow definition. Second, the production prediction service requires that each time it is triggered, a model configuration is changed. We name the corresponding KPI *adaptability*. Thus, different model configurations should be supported, through a workflow definition. E.g., in every new production prediction service call, the configuration of the chemical kinetics model requires to be updated. It requires updating the initial values of alcohol concentration and time. The configuration is updated using the latest sensor data. Also, the variables within the models that need re-configuration should be accessible through their interfaces.

Finally, we defined two extra KPIs for the *production control* requirements. First, this service requires that the developer can trace the data back to its original source. It is crucial to have this capability in order to assess the impact of control actions. Thus, we define the *traceability* as the level of traceability a developer is able to achieve. This KPI requires modularity in its design, to identify the producer of the data and have a clearly defined workflow. Second, a control service is based on a timely response. Hence, it is crucial to define a KPI related to the time elapsed during the models' execution and the control action. We define the KPI called *latency* for this purpose. This KPI is correlated to the approach the tool uses for different model execution and its data exchange. The *latency* is measured from the moment of sensor data acquisition to the control signal execution on the actuator. Hence, in contrast to the *updating delay*, *latency* also considers the models' output processing towards a control command.

6 EVALUATION RESULTS

This section shows the results of the evaluation of the KPIs (Table 3) and of the features (Table 4).

6.1 KPI evaluation results

The results of the KPI evaluation are shown in Table 3. The Table lists all the KPIs with the measurement units. The different columns show the evaluation result for each framework.

Table 3: KPI results of the evaluation; [s] and [ms] represent seconds and milliseconds respectively, μ is the mean, σ is the standard deviation.

KPI	TwinOps	ThingsBoard	Ptolemy II
LoA [%]	75	75	50
UpD [s]	$\mu = 54.8$, $\sigma = 4.12$	$\mu = 35.2$, $\sigma = 0.4$	$\mu = 26.4$, $\sigma = 1.02$
HiD [bool]	excluded	included	excluded
Adap [bool]	included	included	excluded
Tra [level]	low	high	medium
Lat [ms]	$\mu = 249$, $\sigma = 20$	$\mu = 75.4$, $\sigma = 1.02$	$\mu = 88$, $\sigma = 9.98$

In *Level of Automation (LoA)* we required support for four tasks: interface implementation, data exchange definition, and storage, models scheduling definition, and configuration definition. Table 3 shows `TwinOps` and `ThingsBoard` to be frameworks with higher levels of automation, requiring a single manual task. The `TwinOps` manual task is the data storage implementation, in which we use Kafka [32]. `ThingsBoard`'s manual task is interface implementation, as it does not support this feature. We implemented it using FMPy [33], which has an orchestrator for models encapsulated as FMUs. We defined an interface between the FMPy application and `ThingsBoard` using a TCP-IP connection. In this way, the interface enabled `ThingsBoard` to send control signal and exchange data with the models, while FMPy executes the commands and transfers data back to `ThingsBoard`. `Ptolemy II` requires two manual steps. The first is a modification of its support for FMI models and Simulink models. `Ptolemy II` required changes in the MATLAB actor to support a Simulink model interface. The second manual step is the implementation of data storage, because `Ptolemy II` does not support it. We use Kafka [32] to implement the data storage.

To measure the *Updating Delay (UpD)*, we performed five cycle runs, and measured the models' execution time in each cycle. We limited our runs to five cycles due to time constraints. The implementation of each tool and the enhancements of the simulation models accounted for the majority of the project duration. The *Updating Delay (UpD)* results show that `Ptolemy II` has the smallest mean, THINGSBOARD the smallest standard deviation. The high *UpD* mean results in `TwinOps` are due to its cloud host CI/CD engine. The *UpD* mean results in `ThingsBoard` are due to the orchestration engine which combines remote procedure calls and RESTful protocols. In contrast, `Ptolemy II` runs locally.

The *Historical Data (HiD)* KPI shows that only `ThingsBoard` has support for data storage. As mentioned in the LoA KPI, `TwinOps` and `Ptolemy II` required a database (DB) implementation to perform the data storage. Thus, they do not support this historical data KPI.

The *Adaptability (Adap)* KPI results show that `Ptolemy II` does not support this KPI well: to access particular configuration parameters of the models, we needed to change all models. The parameters needed to be converted into inputs to have access to them while using `Ptolemy II`. However, `TwinOps` and `ThingsBoard` support configuring models by using their pipeline and rule chain definitions, respectively.

The *Traceability (Tra)* KPI is measured in three levels: high, medium, and low. The results show that `TwinOps` hides the traceability of the data due to the CI/CD stage. This stage causes the data coming from

the models to be invisible to the developer. This effect is amplified by the containerization. Conversely, `ThingsBoard` ensures data properties and format. It enables data labeling to ease its traceability during its execution. Thus, `ThingsBoard` supports a high level of traceability. Finally, `Ptolemy II` offers medium support for traceability due to its actor-oriented architecture, enabling tracing the original producer of data. Nevertheless, it does not support data transformation between actors or data properties traceability (e.g., label data or store original properties). Thus, data consistency depends highly on the developer.

Finally, to measure the *Latency (Lat)*, we performed 5 cycle runs, and measure the models' execution time and the processing time to generate the control signal in each cycle. The results shown in Table 3 resulted from the statistical analysis of those cycles. *Latency (Lat)* KPI results are similar to those for *UpD*. `TwinOps` has a high mean and standard deviation compared to the other two frameworks, again due to the CI\CD engine latency. Nonetheless, this KPI shows that `ThingsBoard` seems to handle latency better than `Ptolemy II`; its mean and standard deviation are both lower.

6.2 Feature evaluation results

Table 4 shows the results of the evaluation of the features identified in Table 1. The results are presented in levels. The level *High* states that the feature is supported completely; *Intermediate* that the feature has some functionalities that are not supported; and *Limited* states that the feature is either not or poorly supported.

Table 4: Features (from Table 1) evaluation results. Results are High, Intermediate, or Limited.

Feature	TwinOps	ThingsBoard	Ptolemy II
I1	Intermediate	Limited	Intermediate
I2	Limited	High	Intermediate
I3	Intermediate	High	Limited
O1	High	High	Limited
O2	High	High	Limited
O3	Intermediate	High	Intermediate

The results for the `TwinOps` framework show it to support the majority of features. However, features *I1* and *I3* are supported at an intermediate level. *Interface generation (I1)* has intermediate support because the tool OMEdit only supports FMI and Modelica models. If a developer is required to integrate a model that cannot be encapsulated in an FMU (e.g., a model from Aspen plus), the developer would be forced to implement the interface manually. The *Data exchange consistency (I3)* feature is partially supported because we were required to integrate an external DB manually, which required extra manual effort to achieve consistency. This framework supports *Model diversity management* at an intermediate level, due to its low support for traceability. Particularly, the *Model diversity management* requires good traceability to detect errors or inconsistencies in the execution of models that have different notions of time (e.g., continuous and discrete time). Finally, `TwinOps` has a limited support for *Modularity*. The limited support is because the workflow of `TwinOps` requires constructing two layers (CI/CD and containerization) over the integration process (performed in the OMEdit tool). Those processes, especially the containerization, hinder the traceability of the model data. Thus, to maintain traceability, we would be required to define and implement a mechanism to trace the data to its source for every input and output, increasing development complexity. The features *Workflow automation (O2)* and *Configuration (O1)* are evaluated high for `TwinOps`. However, changing the configuration of a model or defining different data exchange patterns (e.g., FIFO) requires a complex setup in the CI/CD engine. This increases the complexity of the orchestration.

The results show that `ThingsBoard` supports the majority of the features at a high level—with one exception, *Interface generation (I1)*. This feature requires an external tool (FMPy) to implemented the model interfaces, and which supports only FMU models [9]. Hence, similarly to `TwinOps` limitation, if a model

is created in an engine that does not support FMI encapsulation, a manual interface implementation is required. Features such as *Configuration (O1)* and *Workflow automation (O2)* have an evaluation results high, however, similarly to `TwinOps`, the rule engine has high complexity to define model orchestration. An example is for configuring a model time step, this task requires a rule that has a similar structure and semantics to a general-purpose language. Hence, the rules definition does not ease the orchestration one.

The results from `Ptolemy II` show intermediate or limited support for all the features. It shows limited support for I3, O1 and O2. *Data exchange consistency (I3)* has limited support because there is no support for automated consistency checking. *Configurability (O1)* also has limited support, requiring the conversion of model configuration parameters into runtime input. This requirement is outlined in Subsection 5.1 due to the production prediction service. *Workflow automation (O2)* is constrained as manual implementation of the DB connection for data storage was necessary. Additionally, directors, acting as orchestrators, had to be modified to define data orchestration to the DB. For *Interface generation (I1)*, `Ptolemy II` shows intermediate support. It supports FMI and MATLAB model integration, but support for Simulink models requires modifications to the actors. *Modularity (I2)* has intermediate support, as the tool's agent-based architecture creates modular support by separating models and sensors in the application. However, `Ptolemy II` requires an external DB to support traceability of data generated by each model (encapsulated by actors). Finally, `Ptolemy II` provides intermediate support for *Model diversity management (O3)*. It can execute diverse models using various MoCs, but while new MoCs can be defined, this requires extensive changes.

7 DISCUSSION AND CONCLUSION

We investigated the capabilities of open-source tools to support the development of DTs. To achieve this goal, we selected three open-source frameworks that are suited for developing DTs: `TwinOps`, `ThingsBoard`, and `Ptolemy II`. In addition, we selected a case study, a Brewery DT, to evaluate the selected frameworks. We conducted literature research to investigate the set of features required to develop DTs with heterogeneous models and data sources. We also associate those features with the requirements and KPIs of the case study. We then built the Brewery DT and evaluated each KPI and feature in this context.

We see three limitations of our study. Firstly, the identification of essential features is not exhaustive due to limitations in the review of current literature on DT development practices. However, the results of this research represent a step towards an evaluation taxonomy, albeit not necessarily a fully general or comprehensive one. Despite this, the study successfully develops a use case based on the identified essential features. Secondly, the analysis relies on the authors' perspective, introducing potential bias. However, efforts were made to establish a method for measuring and mitigating this bias. Lastly, the specificity of the brewery use case, with its specialized domain and reliance on dynamic models, may limit the generality of the findings to also cover other domains. Nevertheless, the study is relevant to DT development, with its reproducible analysis approach and the findings being valid for application in various domains.

Our findings for *RQ1. What are the essential features of a tool to ensure successful integration and orchestration in a DT?* are provided in Table 1. This table contains six fundamental features any tool should have to ease the effort of integrating and orchestrating a DT with heterogeneous components. These features can serve as a taxonomy in future studies, for assessing the effectiveness of a tool in supporting the integration and orchestration of DTs. Despite its limitations, this taxonomy represents an initial step toward establishing evaluation criteria for the completeness of tool support.

For *RQ2. How robust are the feature support offered by OS tools to facilitate integration and orchestration in DTs?*, our findings are shown in Table 4. Based on this table, we can conclude that the selected frameworks do not completely support the minimum features necessary to build a DT. In particular, *Interface generation (I1)* is lacking in all the frameworks. This feature is critical to building a DT with heterogeneous simulation models in its VE effectively. Furthermore, we noticed that manually implementing a new interface is a com-

plex process. Hence, a mechanism to guide the extension of new interfaces is also needed. This aligns with the interdisciplinary collaboration within the DT development and underscores the limitations of existing integration technologies, exemplified by FMI's insufficient support.

In addition, we analyze the features of orchestration *Configurability (O1)* and *Workflow automation (O2)* in Section 6.2. We conclude from the analysis that the orchestration approaches of each OS tool are highly complex or restrictive. The tools `ThingsBoard` and `TwinOps` use approaches that do not reduce the complexity of defining the orchestration. The orchestration definition of the orchestration for both tools tends to be long and complex due to the languages used. The tool `Ptolemy II` uses a formal orchestration definition based on MoC (Model of Computation), which has restrictions in combining them. If a DT orchestration requires a special definition, `Ptolemy II` would require major changes.

In summary, existing OS tools fail to meet the requirements we outlined on the elicited features, particularly concerning interface generation. Furthermore, there is a pressing need for these frameworks to decrease the complexity associated with development substantially, especially for model orchestration approach.

As future work, we will investigate a domain-specific languages to define a framework supporting all features and performing well on all KPIs, reducing the complexity of orchestration features such as *Configurability (O1)* and *Workflow automation (O2)*. Moreover, given the significance of *Interface generation (I1)*, we aim to develop a mechanism to ease creating and implementing interfaces for new modeling engines.

REFERENCES

[1] M. W. Grieves, "Virtually intelligent product systems: Digital and physical twins," in *Complex Systems Engineering: Theory and Practice*. American Institute of Aeronautics and Astronautics, Inc., Jan 2019.

[2] F. Tao and M. Zhang, "Digital twin shop-floor: A new shop-floor paradigm towards smart manufacturing," *IEEE Access*, vol. 5, pp. 20 418–20 427, Sep 2017.

[3] M. van den Brand, L. Cleophas, R. Gunasekaran, B. Haverkort, D. A. Manrique Negrin, and H. M. Muctadir, "Models meet data: Challenges to create virtual entities for digital twins," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, Dec 2021, pp. 225–228.

[4] F. Tao, B. Xiao, Q. Qi, J. Cheng, and P. Ji, "Digital twin modeling," *Journal of Manufacturing Systems*, vol. 64, pp. 372–389, 2022.

[5] Q. Qi, F. Tao, T. Hu, N. Anwer, A. Liu, Y. Wei, L. Wang, and A. Nee, "Enabling technologies and tools for digital twin," *Journal of Manufacturing Systems*, vol. 58, pp. 3–21, Jan 2021, digital Twin towards Smart Manufacturing and Industry 4.0.

[6] G. von Krogh and S. Spaeth, "The open source software phenomenon: Characteristics that promote research," *The Journal of Strategic Information Systems*, vol. 16, no. 3, pp. 236–253, Sep 2007.

[7] D. Adamenko, S. Kunnen, and A. Nagarajah, "Comparative analysis of platforms for designing a digital twin," in *Advances in Design, Simulation and Manufacturing III*, V. Ivanov, J. Trojanowska, I. Pavlenko, J. Zajac, and D. Peraković, Eds. Cham: Springer International Publishing, Jun 2020, pp. 3–12.

[8] S. Gil, P. H. Mikkelsen, C. Gomes, and P. G. Larsen, "Survey on open-source digital twin frameworks–a case study approach," *Software: Practice and Experience*, vol. n/a, no. n/a, Jan 2024.

[9] Open Source Modelica Consortium, "Functional mock-up interface," 2023. [Online]. Available: https://fmi-standard.org/

[10] J. C. Kirchhof, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "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, ser. MODELS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 90–101.

[11] J. Hugues, A. Hristosov, J. J. Hudak, and J. Yankel, "TwinOps - DevOps meets model-based engineering and digital twins for the engineering of CPS," *Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings*, vol. 94, p. 668, Oct 2020.

[12] Open Source Modelica Consortium, "OMEdit – OpenModelica connection editor," 2023. [Online]. Available: https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omedit.html

[13] P. Fritzson, "The Openmodelica Environment for Building Digital Twins of Sustainable Cyber-Physical Systems," in *2021 Winter Simulation Conference (WSC)*. Phoenix, AZ, USA, 2021: Institute of Electrical and Electronics Engineers Inc., Dec 2021, pp. 1–12.

[14] P. Aivaliotis, K. Georgoulias, Z. Arkouli, and S. Makris, "Methodology for enabling digital twin using advanced physics-based modelling in predictive maintenance," *Procedia CIRP*, vol. 81, pp. 417–422, 2019, 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, Jun 12-14, 2019.

[15] GitHub Inc, "GitHub Actions documentation," 2023. [Online]. Available: https://docs.github.com/en/actions

[16] YAML YamlLanguage Development Team, "Yaml," 2021. [Online]. Available: https://yaml.org/spec/1.2.2/

[17] I. Mizutani, G. Ramanathan, and S. Mayer, "Integrating multi-disciplinary offline and online engineering in industrial cyber-physical systems through devops," in *Proceedings of the 11th International Conference on the Internet of Things*, ser. IoT '21. New York, NY, USA: Association for Computing Machinery, 2022, p. 40–47.

[18] Docker Inc, "Docker documentation," 2023. [Online]. Available: https://docs.docker.com/

[19] ThingsBoard, "ThingsBoard - open-source IoT platform," 2023. [Online]. Available: https://thingsboard.io/

[20] M. Machado, E. Oliveira, L. Silva, J. Silva, and J. Sousa, "Development of a Digital Twin for Additive Manufacturing," in *Proceedings of the ASME 2021*, ser. ASME International Mechanical Engineering Congress and Exposition, vol. Volume 2B: Advanced Manufacturing. Virtual conference: The American Society of Mechanical Engineers, Nov 2021, p. V02BT02A019.

[21] D. A. Manrique Negrin, L. Cleophas, and M. van den Brand, "Using Ptolemy II as a framework for virtual entity integration and orchestration in digital twins," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, Oct 2021, pp. 233–236.

[22] C. Ptolemaeus, *System design, modeling, and simulation: using Ptolemy II*. Mountain View, California: Ptolemy.org, 2014, vol. 1.

[23] I. A. Udugama, P. C. Lopez, C. L. Gargalo, X. Li, C. Bayer, and K. V. Gernaey, "Digital twin in biomanufacturing: challenges and opportunities towards its implementation," *Systems Microbiology and Biomanufacturing*, vol. 1, pp. 257–274, Mar 2021.

[24] R. Eramo, F. Bordeleau, B. Combemale, M. v. d. Brand, M. Wimmer, and A. Wortmann, "Conceptualizing digital twins," *IEEE Software*, vol. 39, no. 2, pp. 39–46, Mar 2022.

[25] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, and H. Tummescheit, "Model-based integration platform for FMI co-simulation and heterogeneous simulations of cyber-physical systems," *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, vol. 96, pp. 235–245, Mar 2014.

[26] M. Segovia and J. Garcia-Alfaro, "Design, modeling and implementation of digital twins," *Sensors*, vol. 22, no. 14, 2022.

[27] L. Zhang, L. Zhou, and B. K. Horn, "Building a right digital twin with model engineering," *Journal of Manufacturing Systems*, vol. 59, pp. 151–164, Apr 2021.

[28] S. Karolius and H. Preisig, "Developing simulation tools for interdisciplinary modelling," *Proceedings of The 59th Conference on Simulation and Modelling (SIMS 59), 26-28 September 2018, Oslo Metropolitan University, Norway*, vol. 153, pp. 210–215, Nov 2018.

[29] X. Fang, H. Wang, G. Liu, X. Tian, G. Ding, and H. Zhang, "Industry application of digital twin: from concept to implementation," *The International Journal of Advanced Manufacturing Technology*, vol. 121, no. 7, pp. 4289–4312, Aug 2022.

[30] D. J. Wagg, K. Worden, R. J. Barthorpe, and P. Gardner, "Digital Twins: State-of-the-Art and Future Directions for Modeling and Simulation in Engineering Dynamics Applications," *ASCE-ASME J Risk and Uncert in Engrg Sys Part B Mech Engrg*, vol. 6, no. 3, p. 030901, May 2020.

[31] F. Bordeleau, B. Combemale, R. Eramo, M. van den Brand, and M. Wimmer, "Towards model-driven digital twin engineering: Current opportunities and future challenges," *Communications in Computer and Information Science*, vol. 1262 CCIS, pp. 43–54, 2020.

[32] Apache Software Foundation, "Apache Kafka," 2023. [Online]. Available: https://kafka.apache.org/documentation/

[33] T. Sommer, "FMPy," 2023. [Online]. Available: https://pypi.org/project/FMPy/

AUTHOR BIOGRAPHIES

ANDER LEE did his MSc in embedded systems at Eindhoven University of Technology (TU/e) during the project's development phase. He additionally holds a BSc in Electrical and Computer Engineering from the National Chiao Tung University. Presently, he is employed as a software testing engineer in industry. His e-mail address is a.lee@student.tue.nl.

DAVID A. MANRIQUE NEGRIN is a PhD candidate at TU/e's Software Engineering and Technology cluster. He holds a Bachelor in Chemical Engineering, and Master's degrees in Operational Research and in Automotive Technology. With eight years of industrial experience in Manufacturing and Sustainability, David's research interests include domain-specific languages, digital twin design, and systems engineering. His e-mail address is d.a.manrique.negrin@tue.nl.

LOEK CLEOPHAS is an assistant professor at TU/e and a research fellow at Stellenbosch University, with industrial startup experience. His research focuses on digital engineering, model analysis and management, and model consistency and orchestration in heterogeneous settings such as high-tech systems engineering. His e-mail address is l.g.w.a.cleophas@tue.nl.