

MASTER

Investigating frameworks for integration and orchestration a case study on a microbrewery digital twin

Lee, Ander

Award date:
2022

[Link to publication](#)

Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain



Department of Mathematics and Computer Science
Software Engineering and Technology Cluster

**Investigating frameworks for
integration and orchestration—a case
study on a microbrewery digital twin**

Master Thesis

Ander Lee
1539302

Supervisor:
dr.ir. Loek Cleophas

Mentor:
ir. David Manrique Negrin

Committee:
dr.ir. Ion Barosan
dr.ir. Jeroen Keiren

Version 5.1.5

Eindhoven, October 2022

Abstract

Digital twins (DTs) are beneficial to the management of complex product life cycles because they can offer insights from both physical and virtual worlds. In order to reach its full potential, a DT often incorporates a broad range of models and toolsets. The integration and orchestration of these components pose a challenge to the DT developer. In this project, we develop a case study of a microbrewery DT and investigate frameworks that enable integration and orchestration techniques in order to simplify the design process for the trend of growing count of models in DTs.

A DT is the composite of elements from five dimensions, namely physical entities, virtual entities, services, data, and the interconnections between them. In a closed-loop, the virtual entities collect data from the physical entities, apply the data analysis services, and eventually optimize the physical entities.

As DTs become increasingly complex, diverse tools and heterogeneous models inevitably must be brought together in order to cover all aspects of the systems. From there, the challenges in integration and orchestration arise. In short, integration concerns the encapsulation and the interface between virtual entity models. On the other hand, orchestration addresses the problem of execution sequences.

This study set out to investigate frameworks that embody techniques for integration and orchestration. The purpose was to identify good practices which could be reused in DT developments. A microbrewery DT was built as a case study. For now, it supports two services, namely Production Prediction and Production Control. We selected and applied three frameworks of distinctive styles to demonstrate the services. They were TwinOps—inspired by the DevOps principle; ThingsBoard—an open source platform based on the Internet of Things (IoT) and Service-Orientated Architecture (SOA); and finally Ptolemy II—based on an actor-oriented architecture which aims at experimentation in cyber-physical systems (CPS). We consider these three frameworks for their wide variety of uses, and presumed suitability for DTs.

The results indicate that a distributed framework architecture, such as TwinOps and ThingsBoard, enables a high degree of modularity and configuration which supports integration of diverse components. The extensive configuration options also contribute to a higher automation level for orchestration, as initial setup is paid off by automated processing during operations. In contrast, in a monolithic framework like Ptolemy II, the core functionalities have been unified in a self-contained package rather than assembled from a series of deployable modules or pipeline stages. As a result, it can benefit from smaller communication overhead in integration, which suggests better system timeliness. However, the same tight coupling characteristic may also restrict the flexibility of the orchestration.

Our findings highlight possible techniques for integration and orchestration. The comparison of the frameworks should be of guidance to developers considering extending a framework that suits their DT use cases.

Acknowledgement

My special thanks go to my mentor David Manrique Negrin for his contributions to the project. He was the main supplier of the apparatuses for the brewery as well as the chemistry models. In addition, David has offered insightful advice on the design of the experiments, and the writing of this thesis.

I also want express my gratitude to my supervisor Loek Cleophas. He has monitored my progress and patiently reminded me to focus on the big picture throughout.

Finally, I am immensely grateful for my friends and family who have been an unfailing source of encouragement and reassurance.

Contents

| | |
|---|-------------|
| Contents | iv |
| List of Figures | vi |
| List of Tables | viii |
| 1 Introduction | 1 |
| 1.1 Context | 2 |
| 1.2 Domain overview | 2 |
| 1.3 Challenges | 4 |
| 1.4 Problem definition | 5 |
| 1.5 Project Scope | 6 |
| 1.6 Research questions | 6 |
| 1.7 Report structure | 6 |
| 2 Background | 7 |
| 2.1 State of the art | 7 |
| 2.1.1 Monitoring | 7 |
| 2.1.2 Modelling | 8 |
| 2.1.3 Controlling | 8 |
| 2.2 Related work | 8 |
| 2.2.1 Integration | 8 |
| 2.2.2 Orchestration | 10 |
| 2.2.3 DT in bioprocessing | 13 |
| 2.3 Summary | 14 |
| 3 Methodology and Design | 15 |
| 3.1 Architecture of the microbrewery DT | 15 |
| 3.2 System context description | 17 |
| 3.3 Requirements identification | 18 |
| 3.4 Use case description | 20 |
| 3.5 Selected frameworks | 22 |
| 3.5.1 TwinOps | 22 |
| 3.5.2 ThingsBoard | 23 |
| 3.5.3 Ptolemy II | 23 |
| 4 Implementation | 25 |
| 4.1 Production Prediction (S1) demonstrations | 25 |
| 4.1.1 S1 in TwinOps | 25 |
| 4.1.2 S1 in ThingsBoard | 27 |
| 4.1.3 S1 in Ptolemy II | 28 |
| 4.2 Production Control (S2) demonstrations | 29 |
| 4.2.1 S2 in TwinOps | 29 |

| | | |
|----------|--|-----------|
| 4.2.2 | S2 in ThingsBoard | 30 |
| 4.2.3 | S2 in Ptolemy II | 31 |
| 5 | Evaluation and Discussion | 32 |
| 5.1 | S1 KPIs evaluation | 32 |
| 5.2 | S2 KPIs evaluation | 35 |
| 5.3 | Summary of framework comparison | 37 |
| 6 | Conclusion and Future work | 38 |
| 6.1 | Conclusion | 38 |
| 6.2 | Future work | 40 |
| | Bibliography | 42 |
| | Appendix A DT workflow in bioprocessing domains | 47 |
| | Appendix B Microbrewery physical workbench | 49 |
| | Appendix C Kafka primer | 51 |
| | Appendix D FMI revisit | 52 |
| | Appendix E Human-in-the-loop control scheme | 53 |

List of Figures

| | | |
|------|---|----|
| 1.1 | Classification of digital twin | 3 |
| 1.2 | 5D view of DT model | 4 |
| 1.3 | Integration (right) and orchestration (left) | 5 |
| 2.1 | FMI for model exchange (left), and FMI for co-simulation (right) | 9 |
| 2.2 | SysML diagram taxonomy | 11 |
| 2.3 | DevOps concept portrayed by the infinite loop | 11 |
| 2.4 | TwinOps forward and feedback loop | 12 |
| 2.5 | MoCs relationship | 13 |
| 3.1 | 5D view of the microbrewery DT | 15 |
| 3.2 | Services of microbrewery DT in different product lifecycle phases | 16 |
| 3.3 | System context of the brewery DT | 17 |
| 3.4 | Errors that can be detected by ontology checking | 19 |
| 3.5 | Use cases for S1 scenario | 21 |
| 3.6 | Use cases for S2 scenario | 21 |
| 3.7 | TwinOps workflow | 22 |
| 3.8 | ThingsBoard workflow | 23 |
| 3.9 | Meta model of Ptolemy II | 24 |
| 4.1 | TwinOps implementation workflow for S1 | 25 |
| 4.2 | The SSP file rendered in OMEdit | 26 |
| 4.3 | The CI (left) and CD (right) jobs and their outputs | 26 |
| 4.4 | S1 entity management in ThingsBoard | 27 |
| 4.5 | S1 rule chain in ThingsbBoard | 28 |
| 4.6 | A snapshot of the real-time dashboard in ThingsBoard for the <i>heat transfer model</i> | 28 |
| 4.7 | Ptolemy II design for S1 | 29 |
| 4.8 | The internal implementation of FMU Proxy | 29 |
| 4.9 | The SSP layout of S2 | 30 |
| 4.10 | S2 rule chain in ThingsBoard | 30 |
| 4.11 | Integration of S2 to the root rule chain | 30 |
| 4.12 | A snapshot of S2 real-time dashboard | 31 |
| 4.13 | The encapsulation of S2 | 31 |
| 4.14 | S2 design in Ptolemy II | 31 |
| 5.1 | S1 KPI2: updating delay | 33 |
| 5.2 | S2 KPI2: latency | 36 |
| A.1 | The five-step implementation | 47 |
| B.1 | The fermenter before (left) and after (right) being filled up with wort. | 49 |
| B.2 | More devices at the workbench | 50 |
| B.3 | Network topology of the workbench | 50 |

| | | |
|-----|---|----|
| C.1 | An example Kafka architecture | 51 |
| D.1 | Incorrect function calling sequence by Ptolemy II | 52 |
| E.1 | The client interface | 53 |
| E.2 | The original S2 (top) and the human-in-the-loop control scheme (bottom) | 53 |
| E.3 | Water circuit setup | 54 |

List of Tables

| | | |
|-----|--|----|
| 3.1 | Highlighted characteristics of integration and orchestration | 18 |
| 3.2 | S1 requirements | 19 |
| 3.3 | S1 KPIs | 20 |
| 3.4 | S2 requirements | 20 |
| 3.5 | S2 KPIs | 20 |
| 5.1 | S1 KPI1 comparison | 33 |
| 5.2 | TwinOps initial configurations overview | 35 |
| 5.3 | ThingsBoard initial configurations overview | 35 |
| 5.4 | Framework comparison | 37 |
| 6.1 | S3 requirements | 40 |
| 6.2 | S4 requirements | 40 |
| 6.3 | S3 KPIs | 41 |
| 6.4 | S4 KPIs | 41 |

Chapter 1

Introduction

The first chapter introduces the context and the main concepts of a digital twin (DT), followed by describing the current challenges associated with integration and orchestration of models in a DT. This leads to the motivation for constructing a microbrewery case study in order to investigate frameworks for integration and orchestration.

The DT concept was first introduced by Grieves in 2002 [1]. He referred to it as a concept for Product Lifecycle Management (PLM). The concept asserts that systems are dual in nature, that is, a system has a physical side and a virtual side. The premise of the model is that the two sides are bound together via a bidirectional data link.

The association to PLM is that a DT is a dynamic system that adapts to specific stages in the product's lifecycle. The stage changes made in either space are promptly translated to the subsequent outcomes in the other space. For example, the physical product may deliver data about itself or the environment to the virtual space. On the other end, the developer could tune the design on the virtual side, resulting in calibration of the physical product. This bidirectional relationship in DTs makes a clear distinction from simulations, which are frequently confused with DTs. In short, we consider that DTs go a step further by using simulations and analyses to influence the outcomes in physical space.

After Grieves' proposal of using DTs for PLM, the early adopters were from the aerospace sectors such as NASA and US Air Force [1]. The lifecycle of an aircraft normally lasts several decades [2]. Taking a single snapshot, an aircraft is a complex composition of multi-disciplinary outputs. For instance, the jet engine is a mechanical system; the fuel system is largely based on chemical reactions; and the communication equipment is the result of electrical and computer designs. The aerospace industry has shown the utility of DTs regarding products with a broad development scale. As time passed, many more industries quickly caught up with the trend of multi-disciplinary and long-term product lifecycle. Production factories these days use dedicated software to analyze the efficiency of operations and optimize them on-the-go [3]. Hospitals also use digital assets to aid experiments on drugs and patients [4]. As a result, the study of DT development becomes more and more important.

The number and variety of DTs continue to expand these days. In the physical space, this is attributed to advancements—alongside reduction of costs—in wireless sensor networks. In the virtual space, more sophisticated modeling methods are appearing; especially, there has been a rapid progress in big-data models and artificial intelligence models [5]. Under this premise we notice a problem of how to manage all the available components together in a DT in order to support the product lifecycle. To have heterogeneous tools cooperating we need integration and orchestration. These two notions include aspects as focused as data format conversion, as well as bigger considerations like inter-model scheduling. We want to investigate what are the key characteristics for integration and orchestration.

Identifying the integration and orchestration techniques is merely the first step to improve a DT development process. Applied separately, individual techniques can hardly reduce the complexity of the process on their own. They need to be arranged in a coherent order and be governed

together by operating principles. This is why we need to experiment with different frameworks for integration and orchestration in this project. A framework, simply speaking, embodies these techniques and consolidates them into a certain workflow pattern, laying out a guideline for a developer to follow. Adopting a framework when constructing DTs also ensures good practices could be reproduced consistently, as it encourages common resource utilization, for instance, through templates or libraries. Alternatively, workflow stages could also be established in the framework as discussed in the later chapters.

An end goal of a DT is to support services in a product lifecycle. In this project we want to search for what approaches, offered by the frameworks, have the potential to enable accomplishing this goal. We choose a microbrewery as the case study, the core services we will focus on are related to the production yield aspect of the beer. A key process of a brewery is fermentation, which typically spans two weeks. The process has various variables that could be monitored by sensors, such as the alcoholic content and the temperature. Moreover, the conditions of the process are relatively mild, i.e., low temperature, medium high pressure, which allows easy observations. The fermentation process is a well studied phenomenon, making it straightforward to construct and evaluate. Taken together, these conditions within the case study give us great flexibility to test the different frameworks.

1.1 Context

As claimed by market analysts [6, 7], the emerging trend of Industry 4.0 is characterized by widespread digitalization. Hence DTs are becoming more widely adopted as industry moves toward a digital transformation which sets out to improve the transfer of information across the value chain. This will allow stakeholders to share knowledge about the designs, conditions, and logistics of their products and operations through digital artifacts. By facilitating the virtual mock-up of the real product, one can overcome the temporal and spatial limitations of experimenting in the physical world, thus greatly reducing the overall cost.

The concept of DT is often discussed together with the model-driven system engineering (MDSE) paradigm [8]. In MDSE, a complex system is treated holistically. It is described in terms of the interaction between systems, or better known as “system of systems” [9]. The system architecture is represented as models in order to support requirements, design, analysis and verification activities throughout the lifecycle. The use of modelling exploits recurring design patterns, hence helping to drive a consistent specification without significantly increasing costs as the system depth—the total number of abstraction layers—extends. It is found in many studies that DTs which follow MDSE have the potential to bring the following benefits [10–15]:

- Reduced maintenance costs during a system’s lifecycle.
- Reduced errors and inconsistencies across multiple iterations.
- Improved multi-disciplinary collaboration, i.e., engineers from different job functions are able to quickly grasp the high-level overview of the design.

1.2 Domain overview

In this section we first discuss the DT classification. After that, we provide an overview a five-dimensional view of DT which helps us to assess the individual components and their interactions.

Classification

Although the term “digital twin” has been mentioned in much literature, such studies seldomly share the same definition [16]. In some studies the definition centers on to multi-scale simulations working together [17]. In other studies it is defined as a high fidelity digital replica of the physical asset [18]. In order to find a definition that fits our case study, and to keep the later discussions

consistent, we will adopt the classification proposed by Kritzinger et al. [19]. According to the classification, ‘digital twin’ can be roughly divided in three categories based on the level of data integration between the physical assets and their virtual counterpart (illustrated in Figure 1.1):

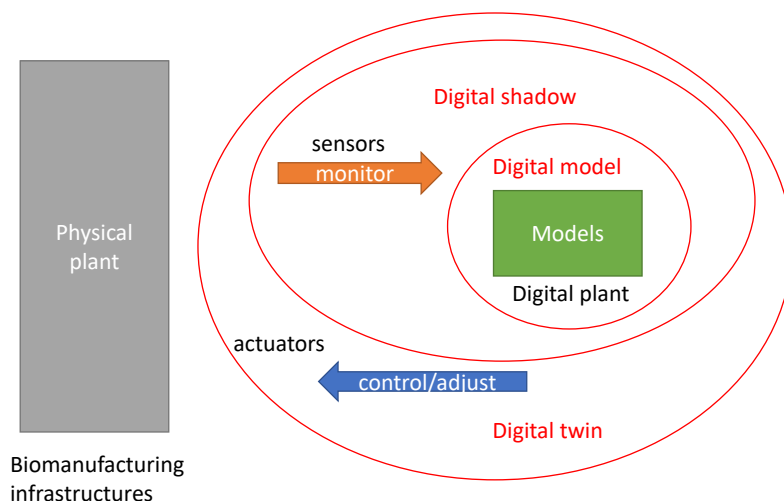


Figure 1.1: Classification of digital twin

- **Digital model:** The virtual space does not involve data exchange with the physical space in an automated fashion. Hence the changes in one part have no direct effect on the other unless updates are performed manually.
- **Digital shadow:** There exists an automated one-way data communication from the physical object to the virtual one.
- **Digital twin:** Bi-directional data exchange between the physical space and the virtual space is automated. Changes made in the physical space will be reflected in the virtual space and vice versa.

In a manufacturing plant setting, data delivered from the physical world to the virtual world is used for monitoring. The reverse direction amounts to the adjustments of actuators. It is important to note that adjusting an actuator in a DT often has a different connotation with the traditional control scheme. In an old-fashioned control style, the computed value is sent to the actuator as a direct activation of certain operations; whereas in a DT, the computed values more likely belong to the parameter set of a complex controlling model that influence the actuators in more nuances, such as adjusting the weighting factors.

A fully automated DT might be unfeasible in certain domains. In particular, for the bio-chemical domain, Udugama et al. [20, 21] recognize that human interventions are still necessary in many scenarios. In practice, it is often the case that DTs deliver the suggestions to the operator through a human machine interface (HMI) first; and the operator can act on the available data. This concept is called human-in-the-loop, and is demonstrated in [22] for an athlete DT, where the coach manages the fitness plan for the athlete based on the reports given by the DT. Since this project concerns a microbrewery which also lies in the bio-chemical domain, the relaxed definition of DT is considered relevant.

Another variant of DT workflow which specifically targets the bioprocessing theme is introduced in [21]. One might find the alternative relevant if the DT includes a multitude of static and dynamic chemical processes. More details can be found in Appendix A.

Dimension

This section introduces a DT model by Tao et al. [23], focusing on the interconnections between different parts. They propose a five-dimensional view of DT model shown in Figure 1.2. Each dimension is described as follows:

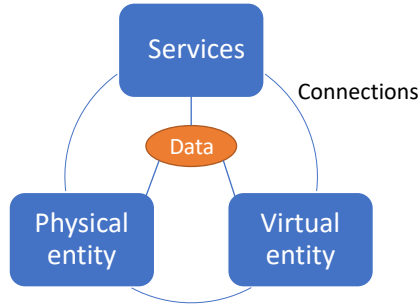


Figure 1.2: 5D view of DT model

- **Physical entity (PE):** A collection of sensors and equipment working collaboratively to collect real-time data for the intended services, meanwhile receiving control orders from the virtual world.
- **Virtual entity (VE):** A digital mirror, which contains models simulating the physical counterpart with high fidelity. Calibration strategies are generated through comparing the models with entities to support models' evolution.
- **Services:** Provides various services to support the management and control of PEs as well as the operation and evolution of VEs. Services are usually requested by the users to fulfill certain functionalities.
- **Data:** A shared storage consisting of the raw data from PEs, VEs, and services. It may also be responsible for merging data from various sources to a fused format.
- **Connections:** The connections for all four of the above-mentioned dimensions. Relevant concepts include communication protocols, access ports, etc.

As we can see the five elements form separate dimensions, yet it has to be ensured that they can inter-operate correctly and effectively. A main goal of integration and orchestration is to address this issue. In particular, the interoperability among VEs is what we will concentrate most on in this project, for the reason explained in the following section.

1.3 Challenges

This section describes the difficulty of managing multiple models in a DT. We give explanations of why the solutions available in commercial platforms are insufficient. Hence it motivates us to look for frameworks that address the challenges. We also acknowledge that generalizing the approaches is equally important as solving for the specific microbrewery case study.

As DTs become increasingly powerful and used for complex systems, diverse tools and heterogeneous models inevitably must be used to cover all aspects for DT constructions. In this step, the challenges in integration and orchestration arise. The terms integration and orchestration can be understood as:

- **Integration** couples different models by encapsulating their properties and operations, followed by interfacing the abstract representations with a proper communication method.

- **Orchestration** dictates models’ execution schedule and arranges the event queue for the events of a particular execution.

Commercial platforms like Simulink [24] or CATIA [25] tackle the issue by expanding their ecosystem with proprietary plugins and extensions so the users can rely less on external implementations. Although these platforms still have some third-party tool support—often by means of imports—they are not promoted as common practice, and the users are more likely encouraged to use the built-in toolbox for better reliability. In the work of van den Brand et al. [26], a similar observation is mentioned, stating that many commercial frameworks support integration and orchestration only to a selective modeling toolset. Therefore, we need to find a framework that extends the support to more models.

The research on an open framework that incorporates integration and orchestration as fixtures in the design workflow remains scarce. As the number of models and their interactions increases, the complexity of control and data flows will grow significantly. Adopting a framework allows the integration and orchestration to scale proportionally, and be replicated consistently. This ultimately contributes to streamlining of DT developments.

Another study produced by Negrin et al. [27] investigated the integration and orchestration concerning an autonomous truck at a distribution center. This study signifies the importance of considering the specific DT’s purpose and use case context, while extracting the universal properties of integration and orchestration which can be applied in other domains.

1.4 Problem definition

Figure 1.3 illustrates the main ideas which are covered by integration and orchestration respectively. On the right side, it shows that integration comprises two aspects, namely encapsulation and interface. Encapsulation refers to the abstraction of a model, as well as discerning the relevant inputs and outputs from the rest of internal operations of the model. The common notion of a “black box” system is a form of encapsulation. This allows the interaction with other models to take place without redundancy while still retaining all necessary information. The interface, on the other hand, is responsible for building a bridge of communication agreed by the involving models, such that the correctness—both numerical and semantic—of data can be reliably maintained. A common way of interfacing is by using an established protocol, so all data transmissions will adhere to a fixed set of rules.

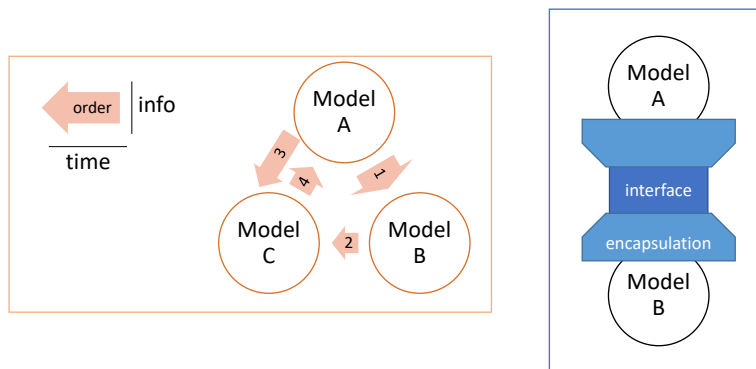


Figure 1.3: Integration (right) and orchestration (left)

On the left side, the concept of orchestration is shown. It encompasses two parts: (1) The scheduling relation between different models; in the figure it is represented by arrow shapes. The numbering in the arrows indicates the execution order. The difference in execution time incurred is represented by length variety of the arrow shapes. Furthermore, different information types may

also be transferred from one model to another, whether it is a single trigger, or a stream of data, etc. The widths of the arrow shapes are used to highlight such discrepancy. (2) The handling of events, that is, the policy to control how event instances are accessed in the event queue [28]. In here the events refers to internal events of a particular execution.

While (1) looks at individual model as an abstraction, (2) focuses on the task sequence within the model in order to ensure a deterministic outcome. A situation that might require event handling, is when several models are scheduled to execute concurrently, their respective events might arrive and leave in different order.

1.5 Project Scope

This is an exploratory project of the possible frameworks for integration and orchestration techniques, supported by the implementation of the microbrewery DT as case study.

The fermentation process models in the DT are given, they are developed by tutor Manrique Negrin, a hobby brewer and a chemist by training. Hence the details of these models or simulation algorithms are not within the scope. A comparative study of the three selected frameworks—detailed in Chapter 3—will be conducted in parallel to the DT services development. Despite the considered use case being influenced by a bio-chemical theme, we strive to maintain universality in the analysis, such that the findings can be applied in other domains as well.

Functional and system level testing of the DT are also part of the implementation scope in order to evaluate the performance outcomes.

1.6 Research questions

Given the premises, in this study an attempt is made to answer the following research questions:

- **RQ1:** What are the key ingredients for integration and orchestration of models in different services for a microbrewery DT?
- **RQ2:** How can these ingredients be generalized to benefit other application domains?
- **RQ3:** In what circumstances do the selected frameworks best fit these ingredients?

Collectively, the answers to these research questions will lead to expanding our knowledge of: *What are considered good practices for the integration and orchestration of DT models?*

1.7 Report structure

The rest of this report is structured as follows:

- Chapter 2 presents the state of the art regarding the technologies that constitute a DT. We will also examine a number of related works which address the topic of integration and orchestration.
- Chapter 3 describes the high-level architecture of the design process. The proposed services as well as their requirements and KPIs will be elicited. We will discuss the selected frameworks for integration and orchestration.
- Chapter 4 describes the workflow of individual frameworks, from initial setup to resultant implementation.
- Chapter 5 explains and compares the results between the frameworks, then summarizes the findings.
- Chapter 6 uses the findings to answers the research questions, and discusses future work.

Chapter 2

Background

This chapter first covers the state of the art in DT technologies in Section 2.1, specifically the three major aspects of monitoring, modelling, and controlling. Although these aspects do not directly address the integration and orchestration topics, however, since they are the core parts of a DT (as explained in Section 1.2), the DT services will certainly rely on them to achieve the intended outcomes. We reckon it is important to be informed about the state of the art in each aspect, so that when building a DT we can take this state of the art into account—including its effects on integration and orchestration.

In Section 2.2 the discussion will shift toward the available integration and orchestration approaches, and how they have been used under the context of bioprocessing DTs.

2.1 State of the art

We review the latest developments of monitoring, modelling, and controlling respectively. Naturally, the DT services are built upon these aspects, implying the frameworks for integration and orchestration have to accommodate them in order to ensure the services running well.

2.1.1 Monitoring

In DT development, the monitoring of the physical world is normally performed using sensors. Clusters of sensing nodes may work collaboratively as enabled by Internet of Things (IoT) technologies.

Hard sensors normally refer to the electronic instruments that detect and collect measurements directly from their vicinity, for instance, a thermostat or pressure gauge. More advanced instruments like spectroscopies—based on the interaction of electromagnetic waves and molecular bonds to measure biomass—are gaining ground [29]. By virtue of their computing capability of properties that are obscured or out of reach to human perceptions, these sort of advanced sensors can be placed in harsh environments such as an agitated bioreactor to provide process data while maintaining non-invasive to the system. In general, high accuracy and low latency are advisable for generating reliable online data.

Soft sensors are the inferential sensing technology that estimate process variables that cannot be directly measured, by combining data collected from hard sensors into equations and a mathematical model. Based on the techniques, they can be categorized roughly to two classes, namely *model-driven* or *data-driven* [21]. *Model-driven* soft sensors are built upon first principles models, which rely on in-depth knowledge of the target process, and that implies higher degree of complexity. The advantage of a model-driven approach is that it has a solid foundation based on physical laws and theories, and that allows an easier generalization of the process provided the user also possesses sufficient know-how of the field [30]. Alternatively, *data-driven* sensors are fed massive data quantities in order to generate predictions. Common approaches include black-box

models such as MultiVariate Data Analysis (MVDA), a form of statistical model that processes multiple variables simultaneously and eventually aims to decrease the dimensionality. Another data-driven approach that is rapidly gaining popularity is Artificial Neural Networks (ANN), thank to the wide availability of hardware support like Graphical Processing Units (GPUs), and software library support such as TensorFlow [31], and PyTorch [32].

2.1.2 Modelling

Rather similar to the classifications of soft sensors described above, modeling approaches can be sorted by their level of abstraction [33,34]. At the higher level, techniques like first principles and mechanistic models in biochemical domains; or Computer-Aided Design (CAD) and topological models used in mechanical domains [35] are considered knowledge-based, i.e., more demanding in domain expertise, hence requiring less online data to construct. In contrast, at a lower level of abstraction, Artificial Intelligence (AI) related techniques rely more heavily on empirical data.

The step for merging the monitored data into the model states is known as data assimilation. Recursive parameter estimation is an approach for data assimilation. It refers to taking old estimates (old model parameters) obtained from fitting one set of data points to generate new estimates when new data points (new observations) are added to the original data set [36]. Another technique within the same category is extended Kalman filter, which is broadly used to perform state estimation [37,38].

Although the models used in this project are given, it is still useful to have an understanding of what modelling techniques are used these days, because different techniques consume and generate different forms of data that may require different treatments of integration and orchestration.

2.1.3 Controlling

We can consider controlling as the last piece in DTs that “closes the loop” between the virtual world and the physical world. Proportional–Integral–Derivative (PID) controllers have been used in industrial control for decades. Their popularity can be attributed the robustness and simplicity in a wide range of industrial settings [39]. On the other hand, the advancement of high volume data acquisition enables growing adoption of model predictive control (MPC), which goes beyond merely a reactive mechanism as PID, also taking into account the contextual knowledge of the underlying complex process. Hong et al. [40] examine the advantage of a hybrid scheme that combines adaptive model-based feedback with direct PID control for optimizing startup, changeover, and shutdown. The study also discusses functionally partitioning components to improve flexibility and reduce operation cost for constructing hybrid models.

In the control service which will be introduced in Chapter 3, we use the PID approach for its simplicity as a proof-of-concept. Nonetheless, as the microbrewery DT will continue to expand and mature, we do not rule out the possibility of migrating to the MPC approach in the future.

2.2 Related work

This section first introduces the recent progress in integration and orchestration approaches. They serve as the references for the designs to consider when choosing the frameworks in Chapter 3. Real case studies of DT will also be described in order to show how the integration and orchestration approaches could improve DT operations.

2.2.1 Integration

Integration is a key step of models’ communication and execution. In a production plant context, communication infrastructures commonly comprise the following aspects [29]:

- A supervisory control and data acquisition (SCADA) system that handles the interfacing of monitor and control operations. Its components include remote terminal units (RTU) for

processing commands; programmable logic controllers (PLC) for processing control signals; HMI for aiding the operator in various actions.

- Data standards such as XML, JSON that deliver string-value pairs of the monitoring/controlling variables. It may also contain information about the computational pipeline of the process.
- Communication protocol stacks that address each Open Systems Interconnection (OSI) layer. Common patterns such as client/server in TCP/IP, or publish–subscribe in Message Queuing Telemetry Transport (MQTT) are well documented in IoT literature.

It is important that the above infrastructures are classified and understood well, because in our DT construction they will become the underlying elements supporting the services to communicate information. In Chapter 3 and 4, some of the above notions will reoccur sporadically.

Apart from the generic infrastructures, we further introduce two open source standards that target specifically the integration in modelling, namely Computer-Aided Process Engineering (CAPE)-OPEN [41], and Functional Mock-up Interface (FMI) [42, 43].

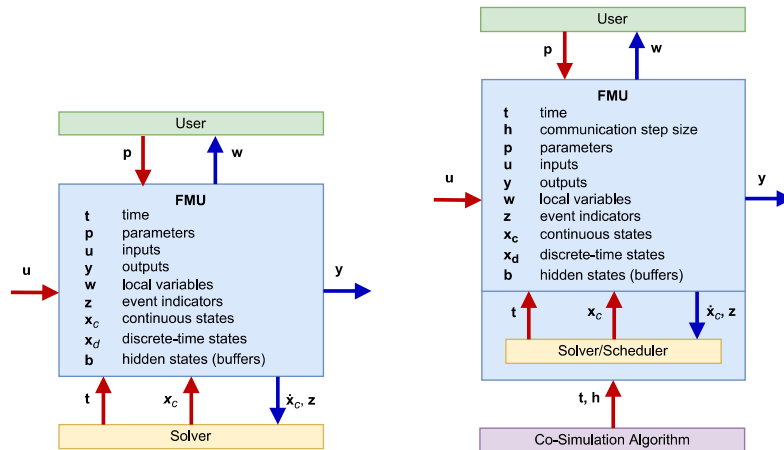


Figure 2.1: FMI for model exchange (left), and FMI for co-simulation (right) [44]

In short, CAPE-OPEN is a standard of a component-based approach to process simulation [45], especially addressing the chemical manufacturing domain. The standard can be seen as having two parts. The first is Process Modelling Components (PMC), they represent functionally separated building blocks such as thermodynamic and physical properties engines, or numerical solvers that compute highly nonlinear equations which arise from the flowsheet. The second one, Process Modelling Environment (PME), is essentially a flowsheet—a common diagram used by chemical engineers to indicate the flow of plant processes and equipments—that utilizes services from PMCs, and is supposed to handle the related connections seamlessly. CAPE-OPEN compliant simulations made by different vendors are able to maintain consistent interoperability without “glue code” or other manually coded wrappers.

While CAPE-OPEN primarily pertains to the chemical industry, FMI is applicable to more general Cyber-Physical Systems (CPS). FMI handles the interfacing of functional mock-up units (FMU), which is the encapsulation of a model in XML format. The XML schema could contain model variables, time-step information, etc. The APIs of the FMI and the specifications of the FMU depend on the choice of interface types. There are currently three types, which are model exchange, co-simulation, and scheduled execution. Since the third type is relatively new—introduced in FMI 3.0—it has not been fully supported by many development environments, thus we will focus on the first two types. They are described as follows (see also Figure 2.1):

- **Model Exchange:** exposes a numerical algorithm (e.g. ODE) to an external solver of an importer (the simulation environment). Using this solver, the FMU is evaluated at a specific time instant.
- **Co-Simulation:** implements not only the model algorithm, but also the required solution method. The data exchange between FMUs is restricted to discrete communication points, thus the co-simulation algorithm (serving as the master algorithm) is shielded from how individual FMUs advance time internally.

In brief, CAPE-OPEN targets a specialized domain, in this case, the bio-chemistry production. FMI is more generic such that it addresses to a wider range of models. Both approaches are valuable to the DT developer as different DT services may include a narrow set of model domains, or they could utilize models from considerably diverse disciplines.

2.2.2 Orchestration

This subsection covers four distinctive approaching styles for orchestration:

- General-purpose modeling language.
- Development-and-Operations (DevOps).
- Service-orientated architecture (SOA).
- Actor-orientated model.

The styles are deliberately chosen because they originate from different software domains. We welcome the diversity because it helps to find out what strategies are suitable for further developing as a framework for our DT. A general-purpose modeling language such as Systems Modeling Language (SysML) [46] is crucial for system engineers to convey designs. The DevOps paradigm is based on the increasing demands of rapid software production by business enterprises. SOA is adopted broadly in cloud computing and IoT applications. Finally, an actor-orientated model like Ptolemy II [47] is inspired by the proliferation of CPSEs which is followed by the desire for an experimentation platform.

General-purpose modeling language

As general-purpose modeling language, Unified Modeling Language (UML) and SysML are two of the more well-known ones. In fact, SysML is a dialect of UML targeting especially MDSE applications, which makes it more relevant under the DT context. SysML—and UML, to a vast extent—emphasizes in providing rich static and dynamic behavioral information in the form of diagrams. To gain an idea of how SysML can describe interactions of a complex system, we summarize its taxonomy in Figure 2.2.

The structure (static) diagrams can represent the system structures in varying degrees of transparency, for instance, Block Definition (black-box), Internal Block (white-box), or Requirement (declarative). The behavior (dynamic) diagrams represent the interactions of internal parts. For example, Activity Diagram describes the flow and the decisions of the component processes. Sequence Diagram extends the precision further by describing the sequence of the processes and what data they propagate in each step.

The various diagram types in SysML enable the inclusion of system descriptions on different abstraction levels, so throughout all stages of the developmental cycle, the stakeholders—likely from various disciplines—can understand relevant diagrams without much efforts. However, one aspect SysML does not concentrate on is the ability to coordinate with external toolsets, especially if they do not adhere to the modeling language syntax in the first place. One who wishes to establish a “live link” to SysML modelling environment from the outside will often find themselves working considerably on a conversion layer. Due to this reason, we incline not to consider it in the framework selection of this project.

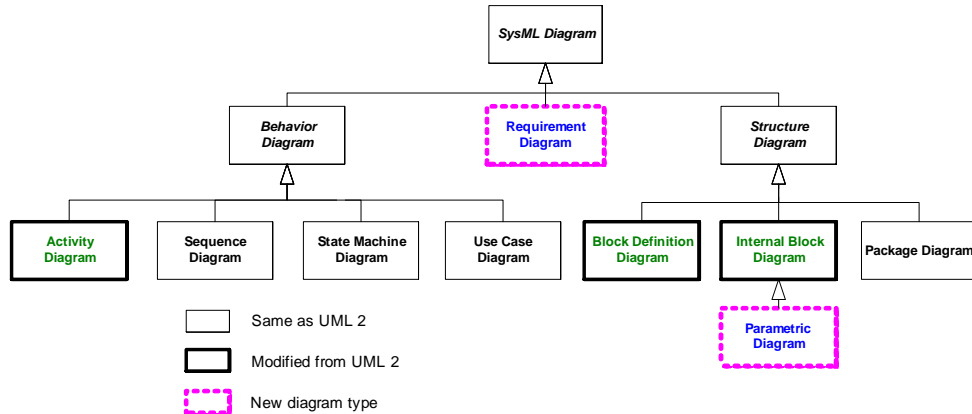


Figure 2.2: SysML diagram taxonomy [48]

SysML is a modelling language, not a framework per se. Yet there exists a number of platforms which utilize SysML to accomplish modelling and design activities, including the orchestration of models within the platform environment; one of them is IBM Rhapsody [49]. This kind of platform inherits the previously mentioned disadvantages of SysML, i.e., cumbersome to incorporate external toolsets that do not adhere to the SysML syntax. However, we find the diagram approach in SysML keeps the execution workflow of the components very well organized. This is an attractive property for orchestration that we would like to have in our DT too.

DevOps

DevOps practice has received much attention, attributing to its emphasis on fast provisioning of business processes [50]. The core idea is that the development and operation are merged into one continuous loop of forward delivery and feedback (see Figure 2.3), often referred collectively as continuous integration and continuous deployment (CI/CD).

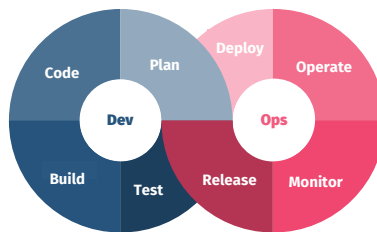


Figure 2.3: DevOps concept portrayed by the infinite loop

Hugues et al. [51] borrows this idea and proposes a DT variant of DevOps called TwinOps. In TwinOps, the “Dev” part transcends to DT model integration and target code generation; and the “Ops” part is overloaded with data collection and analysis in DTs. As illustrated in Figure 2.4, the black arrows are the code generation forwarded to various targets, and the orange arrows represent the data analytic feedback. In the work of Hugues, the exemplified case study of a building monitoring system utilizes the technology stack consisting of AADL, LNT, C, FMI, and Azure to build a pipeline. The AADL toolchain is responsible for specifying the modeling architecture as well as requirements. The LNT target enables model-checking. The C/FMI target supports modeling and simulation of the virtual entity. Finally the C/Azure target leverages a containerized cloud system to deploy execution command on the physical entity.

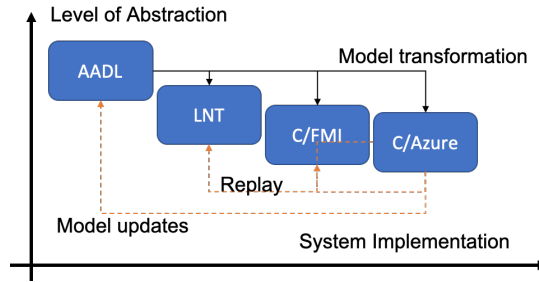


Figure 2.4: TwinOps forward and feedback loop [51]

It is worth to note that under the DevOps presumption, the orchestration is implicit in TwinOps. The orchestration is embedded in the CI/CD pipeline for which the user takes the responsibility to configure the steps for transporting and transforming the model states.

TwinOps will be taken as a framework to be investigated further in Chapter 3.

SOA

As cloud services continue to gain popularity, there is a number of studies [52–54] that look into constructing DTs in line with SOA. In this approach, it regards either the VE models or the services in DTs as independent microservices. The VEs are often deployed as containers, and orchestrated by off-the-shelf applications such as Kubernetes [55], for example, Kubernetes could use DNS for service discovery and exploits network traffic controls for scaling. This view of DTs provides the benefits of rapid deployment, as well as auto monitoring, scaling, and load balancing among other features which are found in commercial cloud services.

As the number of entities introduced to the existing microservices pool continues growing, we can establish a machine-to-machine (M2M) network. An M2M network is characterized by localized—in the sense of not requiring a powerful computing node from another distant network—exchanges of data and services among the entities. It is contrary to the old-fashioned approach where only one central mainframe responsible for all services. This feature is considered attractive under the DT orchestration context, since DT models are also logically separated; can exchange data and contribute to the services.

SOA will be the basis of the ThingsBoard framework which we choose to investigate further in this project. More will be discussed in Chapter 3.

Actor-orientated model

An actor-orientated model as used by the Ptolemy II framework coordinates actors in various models of computation (MoC) while maintaining strong semantics of each individual model [28]. The term “actor” refers to a self-contained component in the system that can interface with other components, comparable to the “object” in object-oriented programming (OOP). The main difference is while OOP objects interface through “methods”, the actors interface primarily through “ports”. The notion of MoC refers to an abstract collection of rules that govern the interaction between components in a design. It is analogous to the “laws of physic” that are used to describe a given system. Figure 2.5 shows¹ a summary of the relationship between MoCs, which are represented as “domains” in Ptolemy II.

We will consider Ptolemy II as one of the frameworks to be investigated further in Chapter 3. Below is a list of some MoCs that we consider more applicable to the DT contexts of manufacturing and production plant:

¹In the figure, the meanings of the colors were never specified in the original source

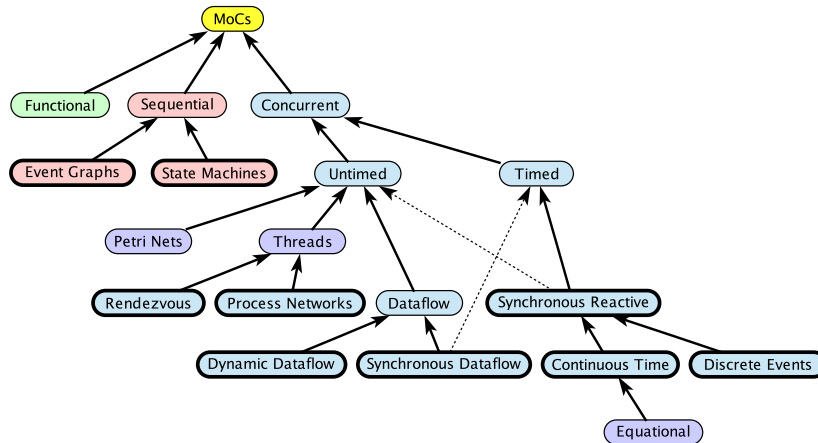


Figure 2.5: MoCs relationship [28]. The nodes with thick edges are currently supported by the latest Ptolemy II version. Bold arrows indicate primary associations. Dotted arrows denote alternative associations with either timed or untimed mode.

- Process network (PN) [56] : scheduling for concurrent distributed processes. It has a benefit of determinacy as long as there is a unique solution to the balance equations of the nodes within the network.
- Finite state machine (FSM): captures control-dominated behaviors.
- Discrete event (DE): suited for modelling the behaviors of complex systems over time, e.g., queuing systems.
- Continuous time: essential for solving ODEs.

2.2.3 DT in bioprocessing

In this subsection a number of case studies will be reviewed. They all share the common theme of bioprocess production, which is also shared by the microbrewery DT in our case. We will briefly discuss the integration and orchestration techniques used by these studies.

A study of enzyme production [57] proposes a model-based strategy to maximize the final fill of a fed-batch process. The study demonstrates a multi-layer approach for orchestration. The cascaded layers are separated by a supervisory layer and a regulatory layer. The supervisory layer implements a mechanistic model that calculates the required start fill. This is the model-based batch planning for initial conditions. The model's parameters are re-fitted using a least squares approach. As for the regulatory layer, the calculated feed rate is adjusted by a PID controller. The results show that orchestrating an additional layer makes the final yield more predictable, which in turn makes the downstream resource allocation easier to manage.

Lopez et al. [58] propose a DT of ethanol fermentation. The researchers use a data-driven soft sensor that takes the online spectroscopy measurements to compute the glucose concentration—referred as process variable (PV)—in real-time. PV is then used as the input to a PID algorithm in order to generate a final control signal—referred as manipulated variable (MV)—that adjusts the feed rate of the controlled pump. The PV-MV transformation in this example showcases how monitoring models and controlling models can be integrated.

In their manufacturing platform for antibodies, Feidl et al. [59] manage to build a process-wide control with a SCADA system. The system collects unit-relevant data streams from each process unit, then converts to a centralized data storage, which contextualizes and adds a timestamp to each data point, in which the data is transformed to process-relevant. Afterward, the SCADA

system is able to send newly determined setpoints to the respective local control units. Hence, an automated end-to-end integration of the supervisory control with the data acquisition system is achieved.

Eppinger and colleagues from Siemens [60] design a DT for ketchup production. The control objectives are evaluated by a set of Key Performance Indicators (KPI). The DT firstly obtains the model parameters from historical data through a machine learning based analysis. A hybrid model that combines an equation-driven model and a data-driven model is then developed before being applied a model order reduction process, such that it can be made compatible with the real-time hard sensors. Once the reduced model is generated, the soft sensors—referred as virtual sensors—can be synthesized and be used to predict the KPIs. Finally, given all available information, the agent of the reinforcement learning algorithm executes actions toward the physical plant and tunes itself with respect to target KPIs by using feedbacks. This study explains a method to orchestrate hybrid models under time critical constraints.

2.3 Summary

In Section 2.1 we surveyed the recent development of DTs regarding monitoring, modelling, and controlling. They are not directly within the topics of integration and orchestration, but the information will be helpful in choosing technologies for our DT building blocks. Besides, some properties of integration and orchestration in fact arise under the influences of these aspects, as they are arguably the backbone of all DT services.

In Section 2.2, we described the approaches for integration, the approaches for orchestration, and multiple DT case studies in the bioprocessing domain, in that order. For integration standards, we presented CAPE-OPEN and FMI, the former targets bio-chemistry applications, while the latter is more general. For orchestration strategies, we reviewed four distinctive styles but decided only three are suitable for our DT. The SysML approach, despite its excellence at propagating system-level descriptions, lacks the features for collaborating with external toolsets.

Chapter 3

Methodology and Design

We have highlighted the close relationship between DTs and MDSE in Section 1.1, hence we choose to adopt the steps in MDSE methodology to present our DT system design. Our steps are as follows:

1. Describe the system context
2. Identify the requirements
3. Describe the use cases
4. Select the frameworks

Before proceeding to the steps, it is worth to show an overview of the DT architecture.

3.1 Architecture of the microbrewery DT

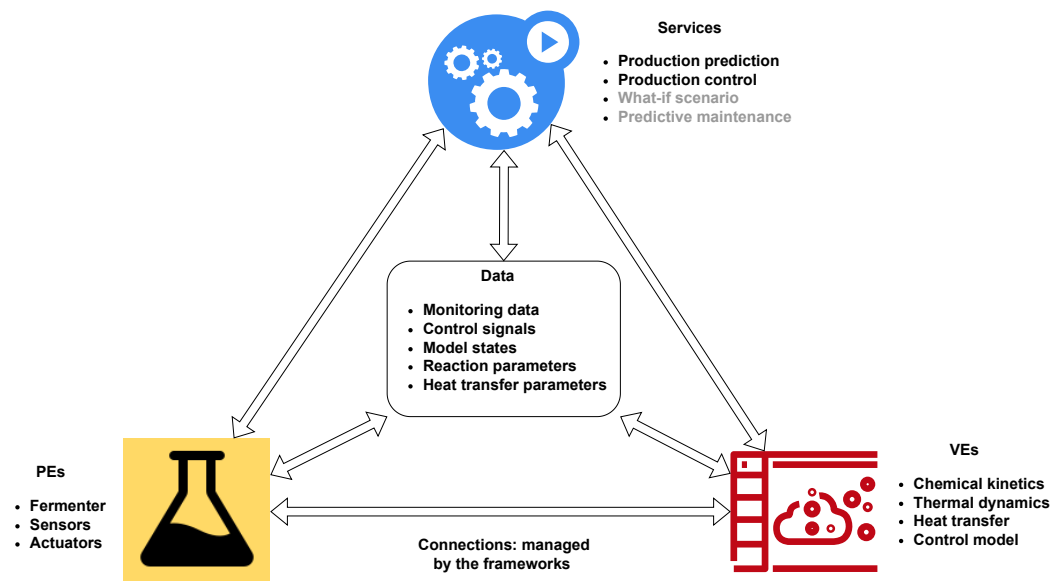


Figure 3.1: 5D view of the microbrewery DT

The high level architecture is described using the 5D view (see also Section 1.2), illustrated in Figure 3.1.

- **PEs:** The *fermenter* (also known as bioreactor) is the vessel that holds the wort—the liquid extracted from the mashing process during the brewing of beer. The sensors monitor the temperatures—both inside the fermenter and from the environment, the ambient humidity, and the gravity in the fermenter. The *actuator* is a water pump that can influence the temperature in the system. Appendix B provides further details of the workbench setup.
- **VEs:** The *chemical kinetics model* is a dynamic model that computes alcohol and yeast concentrations. The *thermal dynamics model* calculates the amount of heat being generated in the system. The *heat transfer model* predicts the future wort temperature. Finally the *control model* produces commands for the actuator in order to alter the temperature to a target value.
- **Services:**
 - **Production prediction (S1):** predict the properties of end-product and whether its quantity and quality will meet the demand based on the given materials and resources.
 - **Production control (S2):** organize the production schedules and regulate the process such that the utilization of resources is optimized.
 - **What-if scenarios (S3):** create a hypothetical situation and predict its effect on the production in order to generate variants of the production schedule.
 - **Predictive maintenance (S4):** use the data stream from the plant and physical-based modelling to generate a prognosis of the remaining lifetime of plant components.

In order to assess the services and understand the key actors and stakeholders, it is important to recognize their respective phases in the lifecycle. Figure 3.2 shows a four-phase lifecycle view [16] from an industrial perspective and where each proposed service belongs in the cycle.

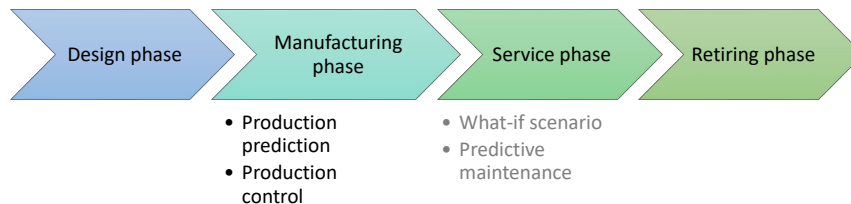


Figure 3.2: Services of microbrewery DT in different product lifecycle phases

The *design phase* encompasses the designs of product, process, and plant, in increasing order of scale. The *manufacturing phase* concerns goods production, and the internal logistical affairs involved. **S1** and **S2** primarily concern the questions of production quantity as well as the strategies to optimize the quantity. Therefore, they are considered to be in the manufacturing phase. The *service phase* includes external logistics, user experiences; the detection of anomalies, and repairs. **S3** and **S4** fall under this category. In the *retiring phase*, decommissioning of the product is dealt with. Valuable data and parts can be obtained from this recycling action so as to improve the future lifecycles.

Due to the stringent timeline, we only implement **S1** and **S2** in this project, reason being that they are considered essential for the brewery operations, and also because **S3** and **S4** are, to a considerable extent, based upon the functionalities of the first two services. Nevertheless, we discuss some preliminary designs of **S3** and **S4** in the Future work section (Section 6.2).

- **Data:** The monitoring data are directed from the PEs, and the control signals are directed toward the PE. In the VE space, the *reaction parameters* depend on the chosen beer flavor. The *heat transfer parameters* are the static function of the physical attributes of the fermenter. The remaining data from the VEs are generally denoted as model states.
- **Connections:** They are managed in different fashions by the selected frameworks. More about them will be discussed in the later Section 3.5 and Chapter 4.

3.2 System context description

Using SysML syntax, the system context is described in Figure 3.3 below. The physical components, such as **Fermenter**, **Sensors**, and **WaterPump** are given as the parts of project context.

- **DigitalBrewery** is the representation of the microbrewery DT. It makes associations with the other blocks which represent the components of the DT.
- **PredictionSystem** and **ControlSystem** are the embodiments of the VE models and the corresponding services.
- **DataManagement** corresponds to the components that manage the storage and the transferring of data.
- **I&O_Framework** corresponds to the selected integration and orchestration framework in this project.

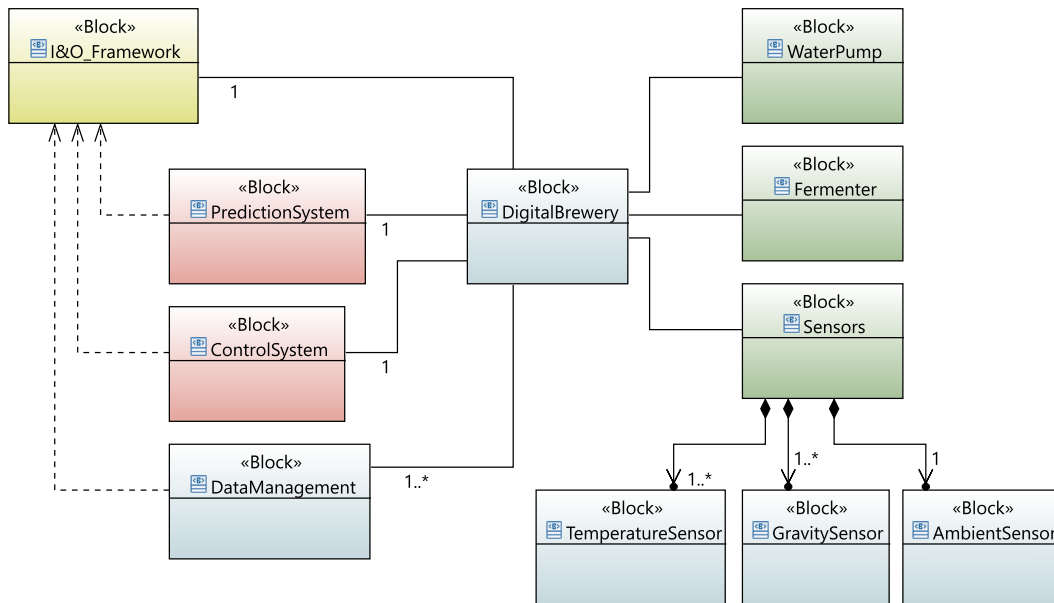


Figure 3.3: System context of the brewery DT. The PE components are denoted in green. The blocks in red color encompass the services and the process models in VEs. The yellow block represents the integration and orchestration framework.

3.3 Requirements identification

Derivation rationale

First we highlight a collection of characteristics aligning with the integration and orchestration theme. This collection is gathered based on the high frequency of occurrences found across the literature of bioprocessing DTs. The findings are elicited in Table 3.1. Later in this section, these characteristics will become the basis for deriving the requirements of the DT services.

| ID | Integration |
|----|--|
| I1 | Configurability of parameters and time advancement |
| I2 | Automated code/data generation for integration |
| I3 | Data exchange consistency |
| I4 | Modularity |
| I5 | Ontology checking |
| | Orchestration |
| O1 | Control workflow and execution sequence |
| O2 | Managing mixed fidelity/granularity |

Table 3.1: Highlighted characteristics of integration and orchestration

I1 can be further broken down to two parts. First is the adjustability of initialization of the subsystems. As Tolksdorf et al. [61] point out, upon the convergence of sub-models into one flowsheet, process engineers often face the challenge of guessing sensible initial values as the sub-models no longer are transparent to them, and a poor guess can easily lead to underperforming models. Therefore, it is argued that the accessibility to critical parameters throughout the whole process is essential to integration. The second part is related to the importance of time step management. In [62], the researchers experiment with varying execution step-sizes for a vehicle DT, showing that to a certain degree, distributed components can be ran with different clock rates and still produce matching results. The ability to parameterize time advancement extends flexibility in the platform under design.

I2 refers to the required automation to combine models in order to reduce human errors. As chemical process optimizations are rarely accomplished by one single program, manually interfacing multiple simulation packages becomes impractical as soon as the system grows large [63]. A systematic method to generate “glue code” including data adaptation is an ideal solution. In practice, we tend to consider a hybrid scheme with varying levels of automation as realistic, as full automation could be too difficult to achieve.

I3 suggests in the case when a variable is transferred from one model to another, it shall retain its structure and its dependency relation with other objects. An important implication of this property occurs when several solvers are working on shared data. If information about algebraic dependencies between outputs and inputs is supplied, the importing tool is able to detect and handle algebraic loops automatically [43].

In **I4** the modularity property implies not only the plug-and-play accessibility of the models, but also the support for templates—static reuse of resources—and instances—dynamic reuse of resources. Actor-oriented design orthogonalizes component definition and component composition, allowing them to be considered independently [64]. It promotes modularity which in turn reduces the associated design costs.

I5 considers ontologies. An ontology is the explicit organization of constituent concepts and the relations between those concepts. More specifically, an ontology can help to express the intended use of a model. In [28], it is indicated that errors may arise from ontology inconsistencies, as illustrated in Figure 3.4.

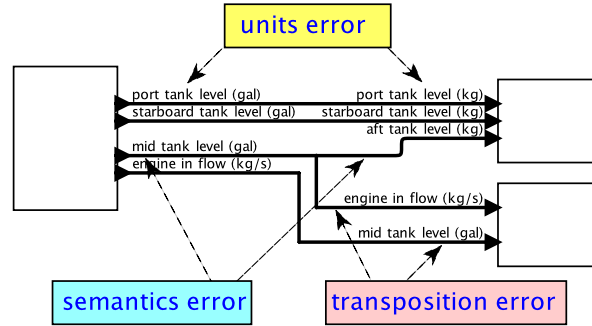


Figure 3.4: Errors that can be detected by ontology checking [28]

O1 reinforces the importance of effective scheduling. One of the key factors that influences the workflow between models is coupling [65], that is, how tightly are models intertwined with one another. A loose coupling relation generally requires less complex schedules. One may use strategies such as using an external framework to de-couple a pair of inherently coupled models. Other considerations, such as atomicity, might be required should there be concurrent processes.

O2 concerns time synchronization and convergence across different models. For instance, merging discrete and continuous time granularities by techniques of sampling and quantization; another example, signaling between time and untimed models. In the case of FMI, an event instant may be driven by a predefined time event or at the transition of state event indicators [43]. This is to allow numerical robustness, and is a part of the FMI feature which is known as “hybrid ODE”.

The mixed fidelity of data implies varying levels of accuracy in which models’ data to quantify the system. This property is often correlated to the computational intensity used by the models. The discrepancy may be solved by data manipulation techniques such as interpolating missing data or sampling out redundant data.

Production Prediction (S1) Requirements

The requirements for Production Prediction service are shown in Table 3.2. The rightmost column indicates the relevant integration and orchestration characteristics.

In addition to requirements, we elicit a set of KPIs in Table 3.3, whose purpose is to verify the effectiveness of the DT regarding how it satisfies the requirements. In principle, each KPI should be a measurable metric that reflects a certain aspect of system performance or non-functional quality. The KPIs are defined in relation to the specific requirements they address.

| Requirements | | |
|--------------|--|--------------|
| ID | Requirement description | Derived from |
| R1 | The model state variables shall be validated with real-time empirical data. | I3, I5, O2 |
| R2 | The operator shall be able to configure the initial model parameters and the model execution scheme. | I1 |
| R3 | The model states shall update automatically based on the latest real-time data. | I2, I3, O1 |
| R4 | The model may be queried for its past states in order to optimize its present parameters. | I1, I4, O1 |

Table 3.2: S1 requirements

| KPIs | | | |
|------|---------------------|---|----------|
| ID | Name | Description | Verifies |
| KPI1 | Level of automation | What are the manual steps required to support the automation of the validation workflow. | R1, R4 |
| KPI2 | Updating delay | How long does it take to perform model state updates. | R3 |
| KPI3 | Data history | What is the amount of historical data that is available for performing predictions. | R4 |
| KPI4 | Modifiability | What configuration options are accessible for model executions and initial values after startup | R2 |

Table 3.3: S1 KPIs

Production Control (S2) Requirements

Table 3.4 and Table 3.5 shows the requirements and KPIs respectively for the Production Control service.

| Requirements | | |
|--------------|---|----------------|
| ID | Requirement description | Derived from |
| R1 | Model optimization may be triggered on the detected disturbance, time, or plant-wide performance. | I5, O1, O2 |
| R2 | Controller calibration may be triggered on time, or state variables deviation. | I2, I5, O1, O2 |
| R3 | Service shall ensure that the calibrations comply with the controller's desired operating range. | I1, I5, O1 |

Table 3.4: S2 requirements

| KPIs | | | |
|------|--------------|--|----------|
| ID | Name | Description | Verifies |
| KPI1 | Traceability | How accessible is the information which concerns the control-induced transitory behaviors. | R3 |
| KPI2 | Latency | What is the response time of actuator triggering. | R1, R2 |

Table 3.5: S2 KPIs

3.4 Use case description

This section covers the use case description for **S1** and **S2**.

Figure 3.5 illustrates the use case diagram for the **PredictionSystem**. This is corresponding to S1, Production Prediction service.

- In the **InitiateFramework** use case, **Operator** accesses the user interface and configures the framework in order to start up the DT.
- **Integrate** manages the data from different sources. It includes **MergeData** which merges data of the sensors and the model states.
- **Orchestrate** manages the scheduling among the models, as well as the validation and updates of the model states.

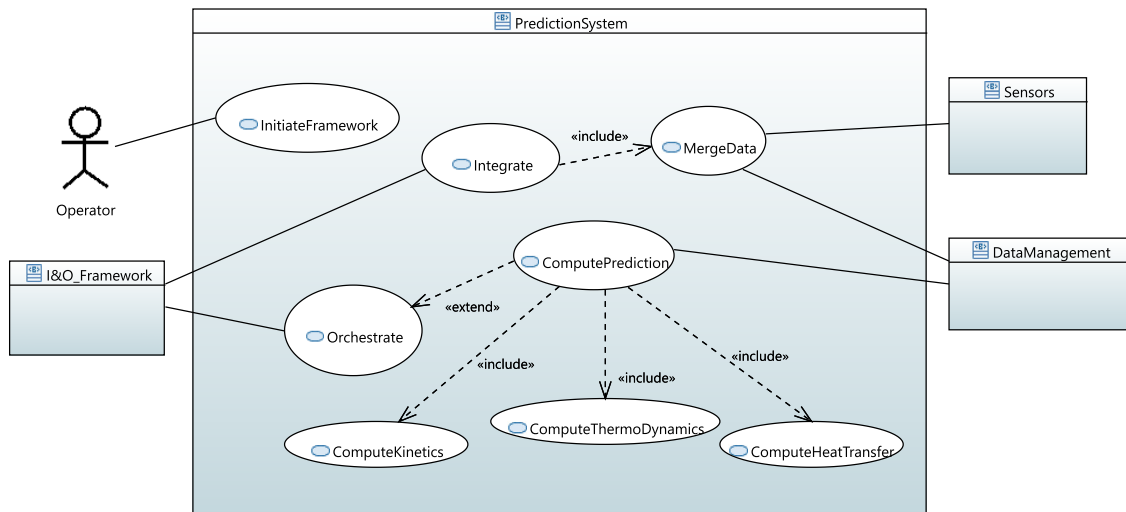


Figure 3.5: Use cases for S1 scenario

- **ComputePrediction** includes all the calculations in order to generate a prediction value. It includes three use cases that correspond to the VE models of the same name.

Figure 3.6 illustrates the use case diagram for the **ControlSystem**. This is corresponding to S2, Production Control service.

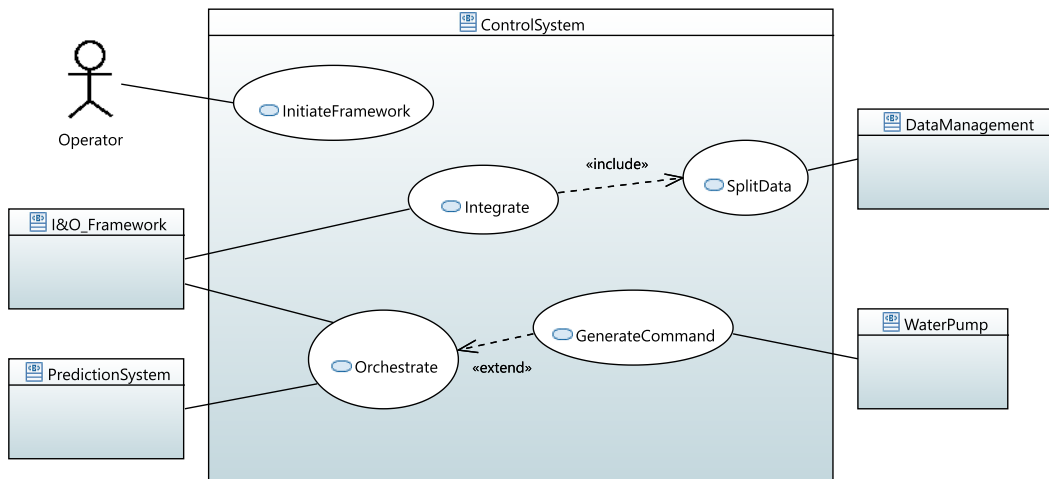


Figure 3.6: Use cases for S2 scenario

- **InitiateFramework**, **Integrate**, and **Orchestrate** use cases have the same behaviors as in the S1 scenario.
- **SplitData** splits the raw data of control to the right destinations in **DataManagement**, with the necessary formatting modifications.
- **GenerateCommand** uses the control model in VE to generate the control signals to the **WaterPump**.

3.5 Selected frameworks

This section covers the architecture and workflow of three frameworks, namely, TwinOps, Thingsboard, and Ptolemy II. They are intended for applying integration and orchestration techniques.

These three frameworks are chosen to reflect on the broad spectrum of the software technology sectors, and to highlight the similarity and difference of approaches under the DT context. TwinOps inherits the CI/CD paradigm from DevOps which aims to eliminate the boundary of developments and operations in order to accelerate the software delivery. We argue that this concept can improve the delivery of DT services as well. Thingsboard has its root in the IoT field, which has always regarded the connectivity between “things” as its core focus. We think this focus can also benefit the connectivity of DT entities. Ptolemy II provides an experimentation platform for CPSes. The management of cyber-objects and physical objects in Ptolemy II is highly applicable to the orchestration and integration of DTs.

3.5.1 TwinOps

The workflow of TwinOps stresses automated transformations, starting at the models in VEs, all the way to the programs running in the PEs. This is also referred as *Model-to-Code-to-Target CI/CD* in the original paper [51]. For the feedback direction, the execution traces from PEs are used to validate and update the models, hence closing the pipeline loop. Figure 3.7 illustrates:

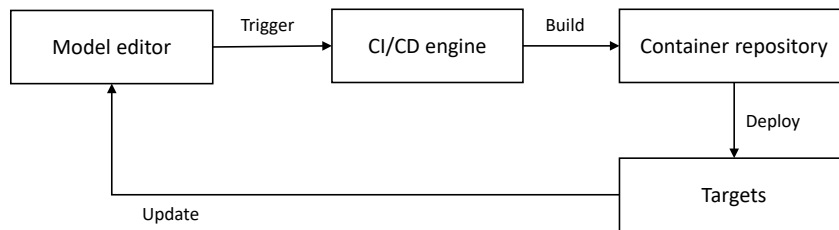


Figure 3.7: TwinOps workflow

- **Model editor:** responsible for building models, drawing the ports topology, and configuring the initial states.
- **CI/CD engine:** integrating the models committed by the previous pipeline stage; validating the models with real-world data; building the containerized image upon passing of all integration tests.
- **Container repository:** a hub which stores numerous iterations of models’ artifacts—such as the generated code—for the targets. This allows the targets to use the latest codes or rollback to previous versions. It is presumed that the targets in DTs run on diverse platforms (e.g., x86, x64, or ARM), hence the container infrastructure helps to distribute the artifacts efficiently.
- **Targets:** a collection of PEs, which can pull the updated program from the repository, as well as produce data analytics which are used for the services.

It is important to recognize that the TwinOps framework is not bound to specific technologies or tools. The DT operator has the liberty to choose the collection of building blocks which fits best to the use case, as long as this choice is able to fulfill the workflow outlined in Figure 3.7. The toolset implemented in the microbrewery DT will be described in Subsection 4.1.1.

3.5.2 ThingsBoard

ThingsBoard [66] project supports M2M-style network. Essentially, it is an open-source platform that enables rapid development, management, and scaling of IoT projects. The key concepts of ThingsBoard are that the user can provision devices and assets; define relations between them and build rule chains that handle the event processing across the relations. From these features, we see the potentials in supporting the integration and orchestration of our DT. Figure 3.8 illustrates the workflow.

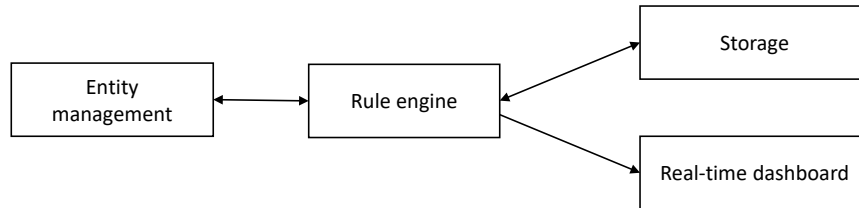


Figure 3.8: ThingsBoard workflow. The arrows represent the direction in which each stage could make modifications to the other stages

- **Entity management:** this is the first step where the user should define all the involved entities and their relations. There are two main entity types. *Device* entity type refers to the basic unit that transmit and receive messages. The *Asset* entity is an abstract type that can form logical grouping of *devices* or other *assets* via *relations*. Commonly used *relations* are “contains”, “manages”, etc.
- **Rule engine:** refers to the event-based *rule chains* which are customized by the user. A rule chain consists of connected *rule nodes* which represent functions or conditions. There are built-in node type such as “filter” or “transform”, etc. The user can also define their own nodes using JavaScript. When a message arrives from the entities the rule engine will produce the corresponding actions based on the *rule chains* and the *relations* which have been given in the entity management stage. As an example, a “smart building” could impose crowd traffic monitoring using *rule chains* to broadcast the readings of the passengers counter—a *device*—in one elevator to all the other elevators which the building—an *asset*—“contains”.
- **Storage:** ThingsBoard supports in-memory storage itself, or one can use the data the supported external space, e.g., cloud-based.
- **Real-time dashboard:** this is related to the user experience and the user interface aspects. The operator can add *widgets* in the dashboard to reflect the events happening in the rule engine in the forms of chart, table or various other visualizations.

There are a couple of underlying services called *ThingsBoard Core* and *ThingsBoard Transports* that basically handle the M2M networking on the platform. They provide communication protocol stacks and account for maintaining the connectivity states. Consequently, they greatly influence system performance. Because these two are background processes, they are omitted in the workflow shown in Figure 3.8.

3.5.3 Ptolemy II

In Subsection 2.2.2 we have briefly described Ptolemy II from the perspective of actor-oriented model. Here we look at Ptolemy II from the software architecture angle. We aim to utilize it as a DT framework.

The official documentation [28] presents a meta model that describes the Ptolemy II software architecture, as seen in Figure 3.9. Below are some important highlights of the figure:

- Every component in a Ptolemy II model is an instance of the NamedObj class. There are four subclasses of NamedObj. These are Attribute, Entity, Port, and Relation respectively.
- Relation class represents communication path between the instances of Entity.
- Port class has many-to-many links to Relation, and it also hosts the Receiver interface that implements methods relating to data communication. Together they can be used to manage the models integration.
- Implemented by Director, the Executable interface contains methods that orchestrate the modeling progressions. In other words, the interface coordinates the iterations among actors based on the rules of the selected MoC.

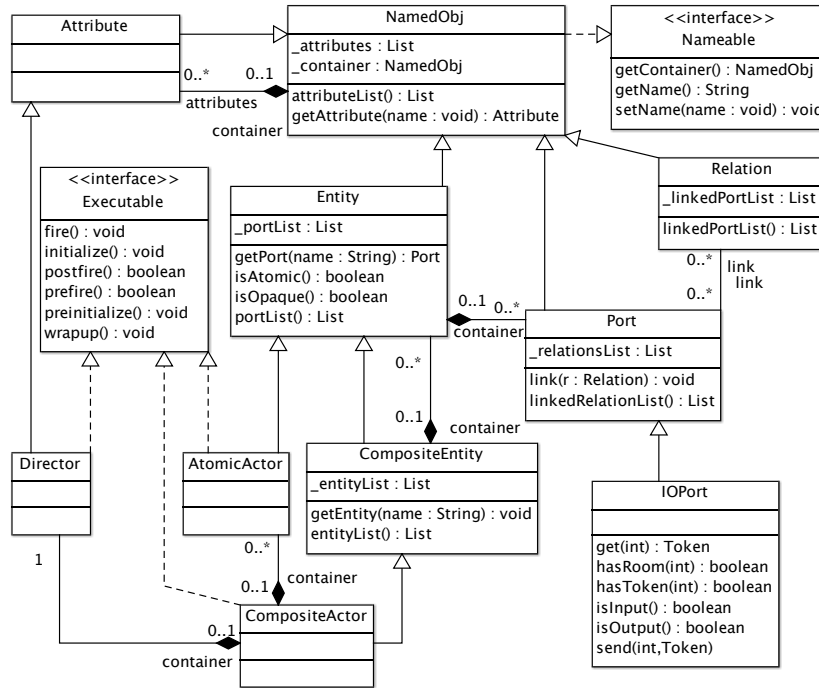


Figure 3.9: Meta model of Ptolemy II

The abstract syntax are implemented as a collection of Java classes and packages. Ptolemy II enables the user to interact with these abstract syntax by the mean of a graphical editing interface, utilizing drag-and-drop actions to accomplish designs.

Chapter 4

Implementation

This chapter presents the implementations¹ of **S1** (Production Prediction) and **S2** (Production Control) mentioned in Chapter 3, using each of the three selected frameworks in order to showcase the possibilities of integration and orchestration approaches. As explained in Section 3.1, **S3** and **S4** are not implemented in this project.

Despite the influence of bio-chemistry theme in our case study, we choose to convert all VE models to FMUs over the CAPE-OPEN standard for the sake of generalizing the approach so they can extend to other domains. As FMI is more inclusive to a wider range of models for integration.

4.1 Production Prediction (S1) demonstrations

In **S1**, the DT collects readings from the three sensor devices of the PE. The scanning period is set to five minutes; we consider this frequency adequate for monitoring the slow variations of the process. After merging the monitoring data, the models of the VE will be invoked to compute the final predicted wort temperature using the given data. Besides, the three chemistry models (see also Section 3.1) have data dependency to one another.

4.1.1 S1 in TwinOps

Figure 4.1 shows the technologies that compose the building blocks of the TwinOps workflow:

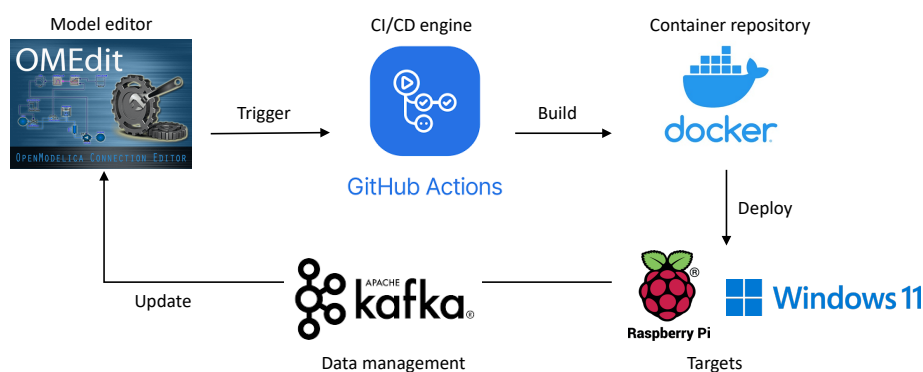


Figure 4.1: TwinOps implementation workflow for S1

¹This is an open source project. One can find the essential software implementations on our GitLab page at https://gitlab.tue.nl/20181234/brewery_dt, as well as the overview of equipment in Appendix B.

For the *model editor* stage, OMEdit [67] is chosen to achieve the task. It is a graphical editing tool that allows user to connect individual FMUs to each other, and then export the topology to the System Structure and Parameterization (SSP) format [68]. SSP is an open standard used to describe the port connections of a set of FMUs. It was created by the same organization which initiated the FMI standard, the Modelica Association. It is supported by an increasing number of major simulation environments which also support FMI. Figure 4.2 shows the layout of our use case in OMEdit.

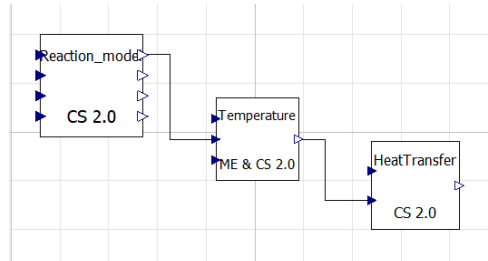


Figure 4.2: The SSP file rendered in OMEdit. Due to their different sources of origin, each FMU has varying supports for simulation modes, such as co-simulation version 2.0 (CS 2.0) or model exchange (ME). The FMUs also have different numbers of I/O ports. Since the models are designed to be “future-proof” for subsequent services, this is the reason why many I/O ports appear unused as they are irrelevant to this specific service

We choose GitHub Actions [69] as our *CI/CD engine* for its native support for a vast variety of operating systems and programming frameworks. The operator can configure the pipeline using YAML—a human-readable data-serialization language that is commonly used for writing configuration files.

As shown in Figure 4.3, two phases (referred as jobs) are created in the pipeline, the left job represents CI, and the right job represents CD which will only be invoked if the CI job has succeeded. In the CI job, the SSP file is validated with the monitoring data which has been retrieved from a user-created trigger which activates every time new data are detected. The results will be saved as a GitHub Actions artifact. In the CD job, the task is to build and push the container image in Docker format [70].

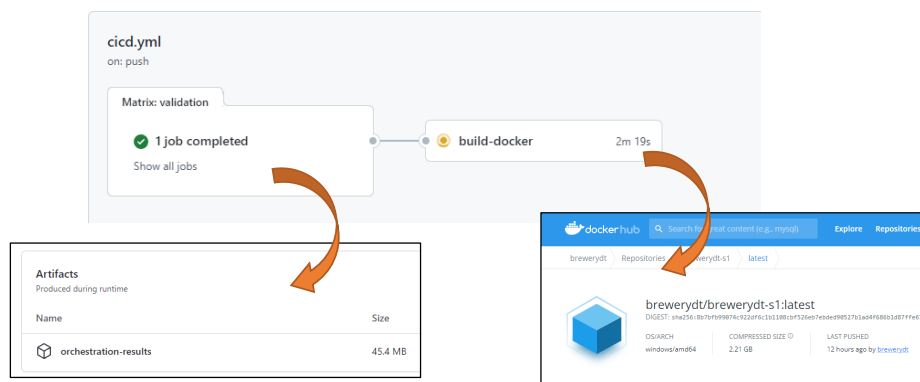


Figure 4.3: The CI (left) and CD (right) jobs and their outputs

The *targets* for our use case are Raspberry Pi and Windows PC. The former is used as a data aggregation hub for all the sensor controls, and the latter is where we host our VE models.

One might notice the Kafka [71] block was not originally present in Subsection 3.5.1 as a workflow stage. The reason being, the update path from the *targets* to the *model editor* stage

could range from simplistic to rather complex, depending on the actual building blocks applied. In our use case, the *targets* involve two different platforms—a microprocessor and a PC, leading us to introduce Kafka as an auxiliary tool to simplify the data management. In the DT context, models may access and store data in different formats and frequency. We adopt Kafka over the more traditional database due to its advantages of compartmentalized data queue and its active data streaming API. One can read more about the principles of Kafka in Appendix C.

4.1.2 S1 in ThingsBoard

Figure 4.4 shows all the *device* entities (sensors) and the *asset* entities used in **S1**. The entities can be added by the operator manually in ThingsBoard UI, or be provisioned in bulk using CSV file. If multiple devices share the common settings, then one might consider creating “device profiles” to improve entity management, like the “Sensors” profile or the “Models” profile shown in the figure.

The asset named “DT-PE” has a “contain” *relation* to all devices of the “Sensors” profile. In other words, “DT-PE” is an amalgamation of devices of the sensor type. The *relation* defines a namespace which the *rule chain* will adhere to.

The image shows two overlapping screenshots from the ThingsBoard UI. The top screenshot is titled 'Devices' and shows a table of device entities. The bottom screenshot is titled 'Assets' and shows a table of asset entities.

| Created time ↓ | Name | Device profile |
|---------------------|----------------|----------------|
| 2022-08-09 10:58:49 | Plaato | Sensors |
| 2022-08-09 10:58:36 | Tilt | Sensors |
| 2022-08-09 10:58:24 | Inkbird | Sensors |
| 2022-08-09 10:57:31 | ReactionMod | Models |
| 2022-08-09 10:57:14 | TemperatureMod | Models |
| 2022-08-09 10:56:28 | HeatTransfer | Models |

| Created time ↓ | Name | Asset type |
|---------------------|-------------------------------------|----------------|
| 2022-08-09 16:45:54 | DT-PE | PhysicalEntity |
| 2022-08-09 10:56:45 | SequentialByOriginator_3a782d3ce737 | TbServiceQueue |
| 2022-08-08 10:28:09 | Main_3a782d3ce737 | TbServiceQueue |

Figure 4.4: S1 entity management in ThingsBoard. `Plaato` is the sensor used for collecting multitude of fermenter statuses; `Tilt` is a gravity detector; `Inkbird` is an ambient thermostat. The asset type `TbServiceQueue` is auto-generated by the core process to handle network traffics.

Figure 4.5 presents the *rule chain*, which has three major parts:

- The top branch (red dashed border) is the event-based logic for inter-models procedure calls. Its primary function is to orchestrate by signaling from the originator model to the destination model, such that a correct execution sequence could be maintained.
- The bottom branch (green dashed border) is for updating the monitoring data from the sensors to the model states. The “DT-PE” asset plays an important role here for transforming the local namespace of the sensor data into the global namespace of the DT, such that the models can use the data without binding to specific sensor devices.
- The nodes in the middle are responsible for recording and saving data with timestamps for future usage by the real-time dashboard.

As introduced in Subsection 3.5.2, the real-time dashboard offers a wide range of *widgets* for the operator to incorporate the human-in-the-loop concept to DT operations. In Figure 4.6 we

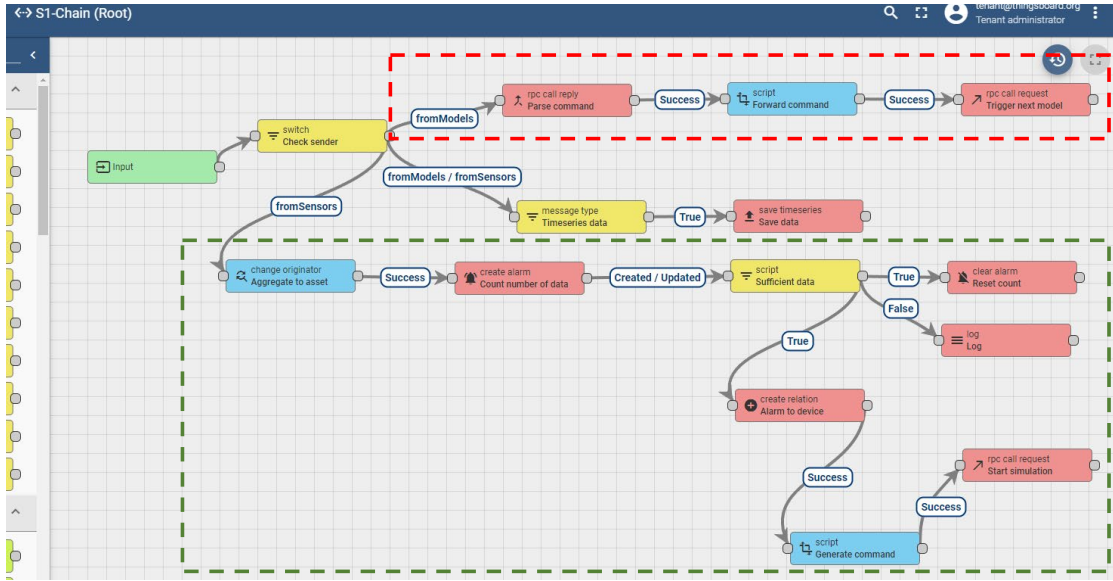


Figure 4.5: S1 rule chain in ThingsbBoard

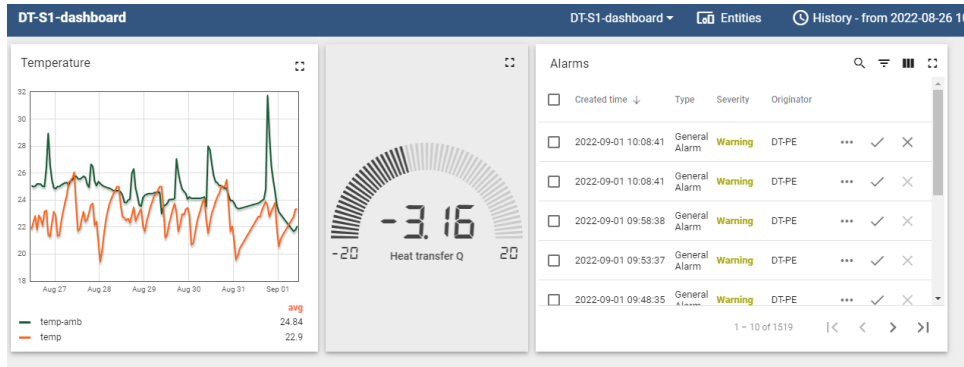


Figure 4.6: A snapshot of the real-time dashboard in ThingsBoard for the *heat transfer model*

demonstrate the usage of built-in *widgets* that visualize the system information. The plot on the left records the temperatures monitored from the PEs. The middle gauge displays the heat loss prediction at a given future moment. The alarm widget on the right side notifies the operator every time the *heat transfer model* has been updated.

4.1.3 S1 in Ptolemy II

Figure 4.7 displays the overall design of **S1**. The Discrete Event (DE) Director is used as the orchestrating instructions are treated as individual “events”. There are two clock sources, `Monitor clock` is synchronized against real-world time such that it takes up the task to trigger the models based on the sensor events of PEs. On the other hand, `FMU clock` dictates the time advancement of the FMU models. It is initialized—by `Monitor clock`—when the monitoring data are ready, and is terminated when all models have completed simulations—notified via a dedicated UDP listener. Each `FMU Proxy` represents a model of the VEs.

Since Ptolemy II does not possess data management actors suitable for our DT, we utilize Kafka to fetch the monitoring data in order to update the models and to store the computed results.

Although Ptolemy II has built-in support for FMI by its `FMUImporter` actor, however, from our experiments we discover it has several weaknesses which lead to undesirable crashes (see

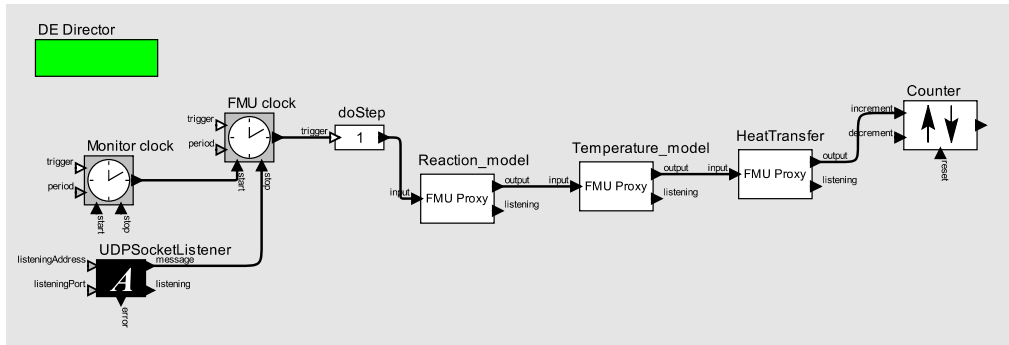
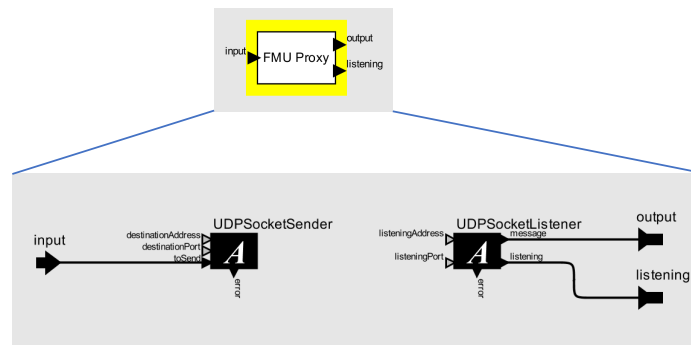


Figure 4.7: Ptolemy II design for S1

Appendix D for details). As a circumvention, a user defined actor named `FMU Proxy` is created as shown in Figure 4.8. The actor serves as a middle-man who forwards the orchestrating instructions from Ptolemy II to the solver outside of the environment via UDP sockets. Using this approach, the operator can orchestrate the FMU models as if they are inside the Ptolemy II environment.

Figure 4.8: The internal implementation of `FMU Proxy`

4.2 Production Control (S2) demonstrations

In **S2**, the control model generates a command based on the fed data. Then the commands are sent from the VE to the actuator—a water pump controller—of the PE. Most of the framework infrastructures are inherited from **S1**. As more services are introduced, we start to benefit from the dividends of frameworks in term of a significant reduction in developer’s workload.

An alternative implementation that uses human-in-the-loop control scheme instead of fully automated decisions can be found in Appendix E. The alternative demonstrates how DTs can be used to assist the operators in a scenario where human intervention is necessary.

4.2.1 S2 in TwinOps

On top of the existing **S1** workflow, the operator can simply add the new model to the SSP file as in Figure 4.9. After that, a new task needs to be added to the CI/CD pipeline so as to pass the computed commands to a remote webhook that controls the water pump.

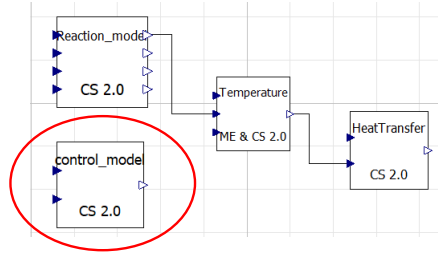


Figure 4.9: The SSP layout of S2

4.2.2 S2 in ThingsBoard

The control model is provisioned as an entity of the “Models” type just as with the models in **S1** previously.

A new *rule chain* shown in Figure 4.10 is added to support the control service. The top branch is responsible for triggering the control model, similarly to the way of **S1**. The bottom branch takes care of sending the command message to the actuator.

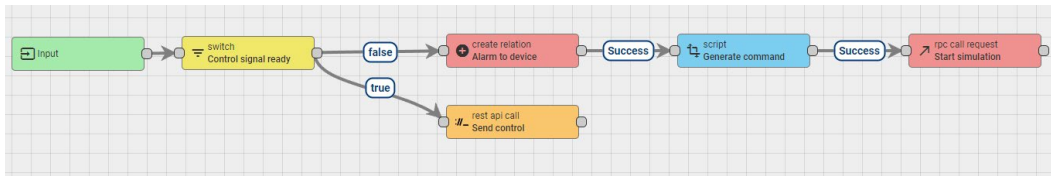


Figure 4.10: S2 rule chain in ThingsBoard

The **S2** *rule chain* is connected to the “root” *rule chain* of **S1** via the “Flow” type nodes (the purple color nodes in Figure 4.11), so the established data routing paths could be reused.

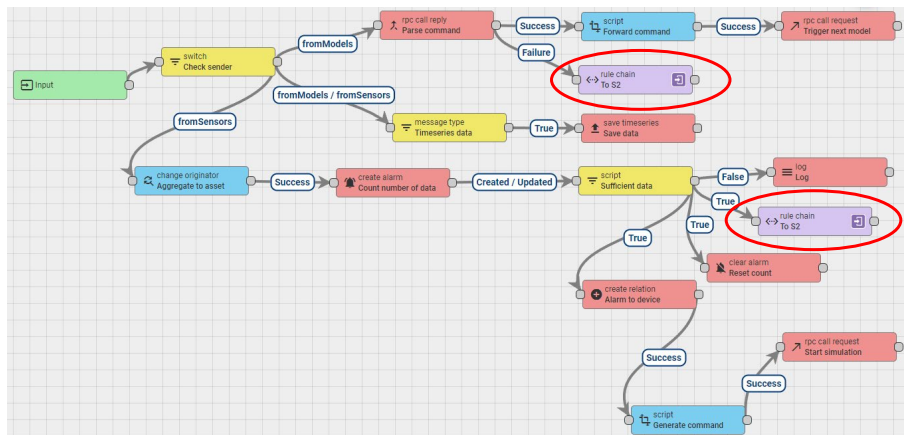


Figure 4.11: Integration of S2 to the root rule chain

In the real-time dashboard, the computed control signal are superimposed onto the timeseries chart of the temperatures, as shown in Figure 4.12.

One can notice a pattern in which the “high” (switch on) intervals of the control commands are followed by the climbing of the internal temperature, and vice versa. This example demonstrates how the framework can offer analytics that enable the operator to trace and assess the effects of the control system.

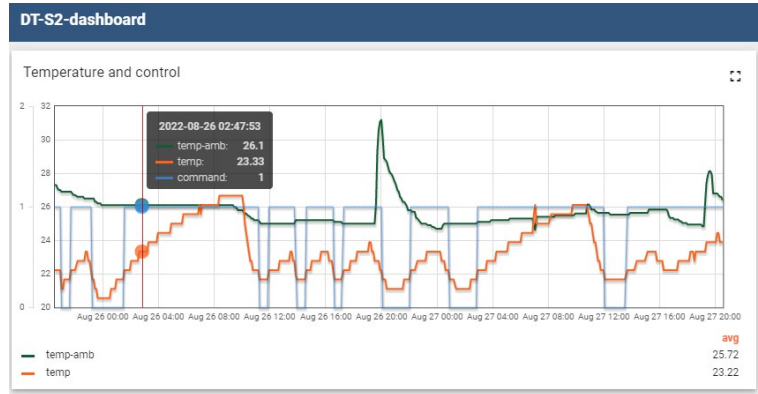


Figure 4.12: A snapshot of S2 real-time dashboard. The chart displays the room temperature (green), the fermenter temperature (orange), and the output of the control model (blue).

4.2.3 S2 in Ptolemy II

Shown in Figure 4.13, we utilize the “composite actor” (circled) in Ptolemy II to encapsulate **S2** and integrate it to the existing design of **S1**, more specifically, reusing the `clock` actors. Using composite actors improves the overall portability as the operator can attach new services to the root service (**S1**).

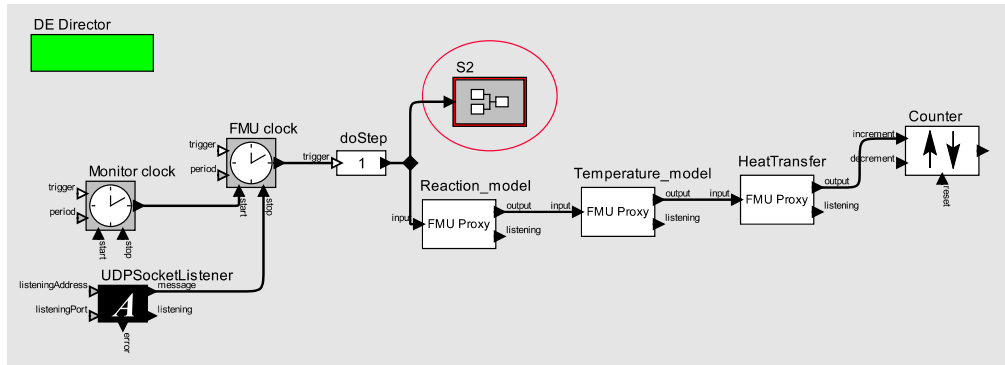


Figure 4.13: The encapsulation of S2

The internal implementation of **S2** is shown in Figure 4.14, where the `FMU Proxy` is triggered by the `FMU clock`—shared by **S1**—from the outside. Then the computed command is sent to a remote actuator webhook using the REST client actor.

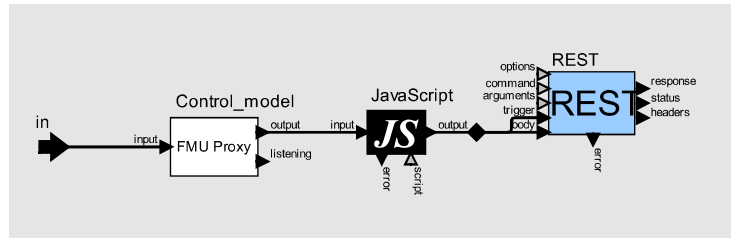


Figure 4.14: S2 design in Ptolemy II

We conclude the implementations for **S1** and **S2** here. In the next chapter we will compare the frameworks’ characteristics, performances, and non-functional properties.

Chapter 5

Evaluation and Discussion

In this chapter, we revisit the proposed KPIs of the two services, and evaluate them in regards to the framework implementations. As explained in Chapter 3, the KPIs are meant to verify the requirements of the services, so we could confirm the effectiveness of the frameworks. After that, a summary will be made to compare the frameworks in overall.

5.1 S1 KPIs evaluation

In **S1** (Production Prediction), four KPIs are highlighted (see also Table 3.3), namely, **level of automation**, **updating delay**, **data history**, and **modifiability**.

KPI1: level of automation

This KPI measures the number of manual steps required in order to support the operations of the framework. Generally, fewer steps are desired as it indicates the framework has a higher level of automation. The steps referred here contribute toward the requirement **R1**—validate the model states with real-time data.

In practice, the notion of “step” depends on the architecture of individual frameworks and the use cases. For the purpose of comparison, we have identified four primary actions that occur in **R1**: data collection, data integration, model scheduling, and status reporting. The manual steps contribute to the automation of these actions in runtime. Note that the actions we have identified here are specific to the microbrewery context. For other use cases, more or fewer actions might be needed to achieve the outcome.

In TwinOps, the collection of monitoring data is handled automatically by Kafka. The openness of the SSP format—which defines the connections between models—allows the internal variables of models to be accessed easily. The core validation process, which includes validating the simulations against real-world data and reporting the new model states, takes place in the CI/CD pipelines. The pipelines are configured in the YAML file by the user ahead of executions. This configuration is the only manual step in the full process cycle. In other words, once the pipelines are pre-configured, the rest of the process should run automatically.

As an IoT-based framework, ThingsBoard uses application-layer protocols that enhance the simplicity in data collection, shielding the user from handling low level communications. However, due to the lack of native model editor supports, the monitoring data and the models are integrated externally to the platform environment during the running of the system. In consequence, a manual step is required for interfacing back to the platform after the integration is complete. Because of that, it poses another disadvantage such that scaling could be hindered as more models are introduced. The model scheduling and its status reporting are automated using *rule chains*. The user is tasked to manually define the *entities* (e.g., sensors) and their *relations* (e.g., sensors are *contained* by the models) before the beginning of *rule chains*. In overall, the two manual steps identified here are interfacing the model editor and entity provisioning.

Ptolemy II comes with an actor type `Accessors` that supports external communication. In term of data integration, both the built-in `FMUImporter` and the `FMU proxy`—which we opted for—bear resemblance to a blackbox. It means that accessing the internal model states is difficult to achieve unless with an external model editor. Hence, facing the same situation as `ThingsBoard`, the user has to implement an additional interface. The model scheduling is automatically maintained by the MoC director and the clock actors. As we mentioned earlier, Ptolemy II lacks a data storage feature that suites our DT service. Therefore the user has to establish data logging externally, Kafka has been manually integrated for this task. All things considered, the two manual steps are interfacing the model editor and interfacing the data manager.

Table 5.1 shows a summary of manual steps in each framework. `TwinOps` is the most automated framework, followed by `ThingsBoard` and Ptolemy II, as both require some patchworks by the user for different stages of the operation. As we can see, `TwinOps` enables the user to manage an extensive range of configuration options prior to runtime, it leads to less inconveniences in term of making further patchworks in various parts of the system, as in the cases with the other two frameworks. Additionally, it could also mean less risks for human errors to occur.

| | TwinOps | ThingsBoard | Ptolemy II |
|---------------------|-------------------------|--|---|
| Manual steps | Pipelines configuration | (1) Interfacing model editor. (2) Entity provisioning. | (1) Interfacing model editor. (2) Interfacing data manager. |

Table 5.1: S1 KPI1 comparison

KPI2: updating delay

We define updating delay as the time between when the framework has collected all required data and when the updated model states have been computed. Therefore, each updating cycle begins from the completion of data collection and ends in the completion of new model states output. We find that one of the main factors influencing this KPI is the communication style of the orchestration, for instance, whether it relies on external protocols or using in-memory communication. Another factor is due to the initialization process. For cloud based platforms this relates to the initiation of the “workers” and to their load balancing algorithm, i.e., the allocation of computational power. For platforms that are hosted in local premises, loading the required resources also contributes to delays.

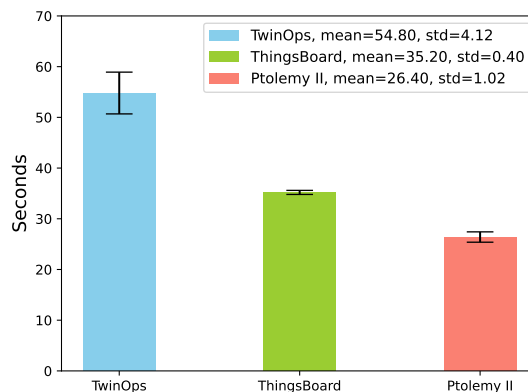


Figure 5.1: S1 KPI2: updating delay mean value and standard deviation

Figure 5.1 shows the statistics of five updating cycles presented in a bar chart. We observe that `TwinOps` has the highest mean delay time as well as the greatest standard variation. This is

due to the CI/CD engine being hosted in the cloud (as with most popular CI/CD engines in the market), such that it is subject to a user’s pricing plan, which results in different load balancing strategies imposed by the cloud server. To keep the measurement consistent, the five updating cycles all use the same free-tier plan.

In ThingsBoard, the orchestration communication utilizes a combination of remote procedure call (RPC) and RESTful protocol. Both are high level protocols that suggest more packet overhead. This results in the moderate delay time we have seen. In most cases, protocol APIs tend to use buffering techniques to ensure QoS, including mitigation of delay time variance. In ThingsBoard we have already seen that being imposed by the background process as `TbServiceQueue` in Chapter 4.

For Ptolemy II, given that we use the `FMU proxy` instead of `FMUImporter`, a communication overhead is expected. Surprisingly, the mean delay time is the lowest among all the tested frameworks. There are a few hypotheses to this. Firstly, `FMU proxy` uses UDP interface for communications, which is a low level connectionless protocol. Therefore it is exempted from delays incurred by the excessive handshakes often found in connection-oriented or high level protocols. Secondly, Ptolemy II holds all of its processes together in an unified software package, meaning it needs few external communications or memory fetching beyond the program scope, in contrast to the modular architecture of TwinsOps or ThingsBoard.

It is important to acknowledge a limitation of this KPI measurement, which is that since the frameworks permit quite flexible building block choices for the user, the technologies adopted in the framework may considerably affect the system timing behaviors. For example, one might use a CI/CD engine that is not cloud-based and possibly achieve a much lower delay time; or one could opt for another M2M protocol that is not built-in to ThingsBoard, thus increase or decrease the overall communication latency. The KPI results presented here should be regarded as a reflection of our particular microbrewery DT setup, serving as a reference instead of an absolute judgment.

KPI3: data history

Data history concerns the amount of historical data ready to be used for the service. For context, the data are being used for computing future heat transfer in **S1**.

In TwinsOp, the user is able to add steps inside the CI pipeline that archive data as artifacts. On the other hand, in the CD pipeline, as we have discussed, data are containerized as images and stored in the online repository. ThingsBoard framework provides native data storage that is integrated with the *rule engine*, such that the historical model states and events are constantly recorded in background. Ptolemy II on its own lacks an efficient data management system. However, we manage to solve that shortcoming by integrating the Kafka service.

In overall, all three frameworks are able to retain all historical data for the full lifecycle of the service.

KPI4: modifiability

This KPI concerns two main aspects: the configuration of model initial values; and the configuration workflow of model executions. Our use case reveals that each framework offers varying degrees of configurability before and after startup, as discussed below.

Due to the modular nature of TwinOps, a user can customize all stages of the framework workflow, including the control of the model executions. In terms of initial value setting for the models, it could be accomplished in the SSP file. Table 5.2 summarizes the initial configuration options accessible by the user. Once all the configuration options are settled, the DT should be able to operate automatically.

Likewise to TwinOps, the ThingsBoard framework is also highly customizable, such that all workflow stages can be configured at the beginning. Furthermore, the *entity management* provides the option to set initial values and attributes to the model entities. The summary of configuration options is shown in Table 5.3.

| | | | |
|----------------|--|-----------------------------------|-------------------------------------|
| Type | SSP | YAML | Dockerfile |
| Format | .ssp | .yaml | No file extension |
| Scope | models editor | CI/CD engine | CI/CD engine & container repository |
| Purpose | Define the execution flow and data dependencies between FMUs | Validate the updated model states | Deploy to the target platforms |

Table 5.2: TwinOps initial configurations overview

| | | | |
|----------------|---|--|---|
| Type | CSV | JSON | Configuration |
| Format | .csv | .json | .conf |
| Scope | Entity management | Profile & rule chain & dashboard | ThingsBoard backend services |
| Purpose | Provision entities and define their relations | Setup device profiles, rule chains, and dashboards | Set parameters to ThingsBoard background processes such as memory allocation, timeout, etc. |

Table 5.3: ThingsBoard initial configurations overview

In contrast, the architecture of the Ptolemy II workflow is relatively monolithic and closed compared to the other two frameworks. Designing is done in the GUI of the software, where the operator performs drag-and-drop on the icons representing directors and actors, though there is the option to import or export the design in XML format. The operator could alter the execution flow by choosing the MoCs, represented by directors. Although the MoCs account for the timing progression of models, they do not address the configurations for the data collection before the model update and the data storage after the model update. It suggests the modifiability of Ptolemy II is inadequate to cover the entire workflow for the service. Many actors in Ptolemy II do not support initialization of model parameters. Therefore, a workaround has to be made by using input variables. Although it does not affect the functionality, potential confusions may arise for the downstream user who does not develop the models in first hand.

5.2 S2 KPIs evaluation

In **S2** (Production Control), two KPIs (see also Table 3.5) will be evaluated, which are **traceability** and **latency**.

KPI1: traceability

In context, the traceability is concerning how much information is available to the operator which they can use to evaluate the effects of the DT's controlling actions.

In TwinOps, the building blocks in each workflow stage are only loosely coupled, they are each an independently deployable module. As a result, the trace of a model's behaviors can be quite scattered. When progressing through stages, the model artifacts, such as connection links, parameters, and states, tend to undergo format transformations that will lose some explicitness of the traces. To illustrate, the models topology is visualized in the model editor stage, but in the CI/CD stage it becomes a part of pipeline instructions that is no longer easily readable by humans. From the CI/CD stage to the containerizing stage, another transformation occurs as the pipeline instructions are brought into a container image which requires another service (e.g.

docker) to run. To sum up, the consistency of trace explicitness is often lost during the transfer of data ownership from one stage to another.

As we have seen in the dashboard demonstration in Section 4.2, the ThingsBoard framework handles the system traceability very well. It is attributed to the highly integrated operability between the *rule engine*, the data storage, and the real-time dashboard. When data ownership is transferred between them, the framework ensures the data properties, e.g., timestamps, data types, etc, remain consistent throughout. Therefore, one is able to easily trace an effect back to its root cause.

In theory, Ptolemy II supports the transfer of data ownership, meaning an actor will gain full controls of data once receiving from another actor. However, this is more true when two actor types are alike. When the types are significantly different, such as between a server actor—which sends and receives internet packets—and a mathematical operation actor, all kinds of errors may occur. The user has to make sure the conversion is done properly. As more kinds of actors are adopted to support the growing DT service, it becomes harder to scale up this kind of pairing conversion. Eventually making the system traces harder to track.

KPI2: latency

This KPI measures the latency between the finishing of model updates and the completion of the control triggering. It indicates the responsiveness of the framework in dealing with VE-PE communications. Figure 5.2 shows the statistics of five measurements.

It can be seen that TwinOps has the highest mean and standard variation. It is again due to the load balancing mechanism managed by the CI/CD host server. ThingsBoard and Ptolemy II have similar mean latency, but the former clearly has the lower of the two variances.

This KPI suffers the same limitation as **S1 KPI2**. The timeliness is sensitive to the high variability of technology choices made by the user. Although we can see a general pattern here that a centralized system such as Ptolemy II has lower latency than distributed systems like TwinOps.

Aiming for low mean latency and low variance are two particularly important properties for a time critical system, even though the microbrewery DT is not one such system. They affect the bound for worst case execution time (WCET), which is a determining factor in schedulability analysis, i.e., the computation deciding whether a set of tasks are able to meet the deadline in a real-time system.

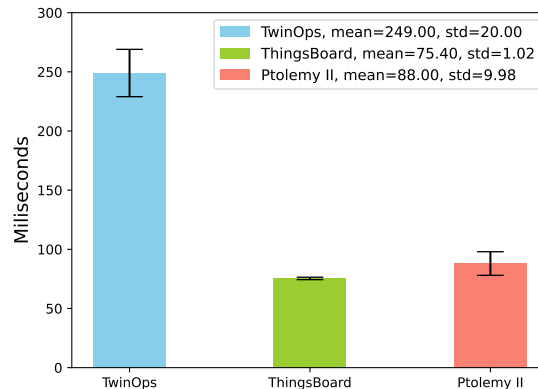


Figure 5.2: S2 KPI2: latency mean value and standard deviation

5.3 Summary of framework comparison

This section summarizes the most significant findings about the frameworks in terms of integration and orchestration. Table 5.4 shows a side-by-side comparison. The rows represent the categories generalized from the KPIs and the requirements to a broader scope; the columns list the three frameworks. Most categories below have been introduced in the earlier chapters. “Data transformation” covers the ability to transform the data to other forms or formats for the subsequent stages within the workflow, or for storing in case of future usages. “Traceability”, as explained in the earlier section, is an aspect of data ownership. We have decided to compare the traceability instead of the greater scope (ownership), as it is a narrower—more focused—aspect, making it easier to compare between the considerably different framework styles in this project.

| | TwinOps | ThingsBoard | Ptolemy II |
|----------------------|----------------|--------------------|--------------------------------------|
| Integration | | | |
| Configurability | high | high | medium |
| Data transformation | high | medium | medium |
| Traceability | complex | simple | simple (but inefficient) |
| Modularity | high | medium | low |
| Orchestration | | | |
| Workflow automation | high | medium | low |
| Timeliness | slow | fast | fast |
| Managing diversity | good | good | good (but requires external add-ons) |

Table 5.4: Framework comparison

Regarding the integration aspects, TwinOps is highly configurable and modular, credited to its multi-stage architecture; its data can be transformed effectively to adapt to different stages of the workflow. However, the same features make it difficult to conduct explicit ownership transfer of model artifacts, meaning the methods for data accessing may vary considerably in different stages. This can hinder the traceability of the DT behaviors. ThingsBoard also has high configurability. The components within the framework are more cohesive, resulting in a more consistent data ownership and medium modular potential. Ptolemy II, being a self-contained software, has limited—under the DT context—configuration options and limited modularity. Some data transformation can be performed by the actors, although it is often not necessary as the processes in the workflow are already tightly coupled. The actor-orientated architecture is conducive to a consistent data ownership although extra patches, such as data format conversions from one actor type to another, are often needed. That makes the tracing of the system become inefficient.

For the orchestration aspects, TwinOps is able to attain the highest automation; we argue it is correlating to the extensive preliminary configurations. It is quite slow, because the cloud-based CI/CD engine focuses on the “continuity” angle instead of the “speed” angle of operation. ThingsBoard takes advantages of its IoT networking backends, resulting in decent automation and decent speed. For Ptolemy II, the workflow requires some manual implementations from the user to automate. It is fast since the framework is relatively more centralized, thus its components suffer less from communications and synchronizations penalty.

Finally, all three frameworks manage to satisfy all requirements of the DT services, even though their KPIs show the weaknesses and caveats that the user has to keep in mind of during the development process. The frameworks are capable of managing diverse types of entities in the microbrewery DT, presuming the requirement for time criticality is not strict, as is the case with fermentation processes.

Chapter 6

Conclusion and Future work

6.1 Conclusion

This project contributes to the identification of the important aspects of integration and orchestration of DT models: (1) We identified the related literature, relevant requirements and KPIs. (2) We have investigated three distinctive frameworks which we believed suitable for DT developments and tested them for a microbrewery DT. In consequence, we have obtained findings that can answer the research questions:

- **RQ1:** What are the key ingredients for integration and orchestration of models in different services for a microbrewery DT?

The microbrewery DT concerns a weeks-long bio-chemical process. Many of its process variables are inferential rather than directly detectable. Taken these characteristics together, the production prediction service (**S1**) has to inter-operate several numbers of entities effectively for a prolonged time frame in order to generate reliable predictions. We argue that a multi-stage or pipelining architecture can resolve the differences across the entities sufficiently well. Since it opens up more ways to combine integration and orchestration techniques by allowing the developer to add or swap individual stages. In this type of design, the emphasis is on the configurability, and modularity of individual stages. Working with flexible building blocks alleviates many constraints for the user when integrating distinctive models.

Another implication of early stage configurations is that it correlates with high level of automation during the operation phases. Our results have shown that extensive initial setups of the DT system reduce the number of orchestration and integration steps that need operator interventions later, thus mitigating the chance of human errors.

In the production control service (**S2**), the operator is most interested in knowing the effect of the control actions. Therefore, the traceability within the system becomes particularly important. The traceability is dependent not only on data availability, even more so the semantics and explicitness of data need to be consistent for the operator to access them easily afterward. Therefore, the trade-off between the degree of data transformations—for the purpose of adapting in different systems/hardwares, etc—and data consistency need to be considered. The trade-off occurs as the data format is transformed in numerous workflow stages, the means of accessing the data may also change repeatedly. That leads to a higher chance of improper data retrieval or negligence by the user, which eventually undermines the data consistency.

Coming back to the research question, the key ingredients are configurability, modularity, and traceability.

- **RQ2:** How can these ingredients be generalized to benefit other application domains?

Most of the previously mentioned points will still hold in the DTs of other fields, as long as their development processes are also guided by MDSE and the 5D model. As these two concepts provide a structured view of a complex system, guiding developers toward the requirements and promoting the usage of recurring ingredients in order to eliminate rework. Nevertheless, in some circumstances, if a certain DT requires real-time processing within a very short cycle time, e.g., seconds, then the system timeliness becomes a more important issue. Our findings indicate that a monolithic orchestrating style has shorter delays than the distributed counterparts, mainly because it has inherently fewer communication points and synchronization barriers.

As a result, when creating DTs for a new application domain, one should identify the unique requirements, and then utilize the MDSE methods and 5D model to incorporate the unique parts into the known patterns based on the past designs.

- **RQ3:** In what circumstances do the selected frameworks best fit these ingredients?

From this project we have learned that the choice of frameworks depends highly on the specific use case. If one's use case stresses on minimizing manual integration and verification by humans, then TwinOps will be best suited as its CI/CD paradigm is made for that very purpose. For the use cases where the models are physically separated on different machines, ThingsBoard will be a wise choice as it is supported by the underlying IoT network. We consider Ptolemy II falls short of several crucial integration aspects as a DT framework, for its lack of data management components and its relatively inflexible user modifiability. However, it serves as a good experimentation framework for testing orchestrations thanks to the variety of MoCs it supports. Apart from that, the monolithic architecture of Ptolemy II allows the fastest overall speed, useful for dealing with time critical applications.

To summarize, in this project we found out the frameworks of distinguishing styles can all accomplish the requirements of our microbrewery DT services. This is an encouraging finding because it allows users from different disciplines to choose what is best fit for their use cases and backgrounds—despite the different shortcomings within the individual framework to be overcome. We learned that multi-stage frameworks have the benefit of accepting more kinds of integration and orchestration techniques, but come with the cost of potentially slower system speed, and more complex system behavior traces, as the data exchanges across stages may be implemented differently. On the contrary, monolithic frameworks are less flexible with especially the integration aspects, but they have the advantage of faster system responsiveness. Ultimately, developers may take these findings as a guidance when designing their own frameworks for DTs of different use cases.

6.2 Future work

For the research on frameworks, we can start looking for extension modules/packages that may compensate for the weaknesses in each framework, just as we have shown that Kafka was adopted as a data management tool to support the frameworks that lack it. The immediate focus will be on the property of time criticality. We are interested in how the framework determines if the operations can meet the deadlines in a real-time scenario, and how can it be guaranteed. Time criticality is prioritized because it is a crucial factor for many systems such as automotives that have safety requirements—unfortunately not the case of a microbrewery.

As for the microbrewery case study, there four services proposed, but only two (**S1** and **S2**) have been implemented, and they both fall within the *manufacturing phase* of the product life cycle. In this phase the main concern is the final production yield.

The other two services, **S3** (what-if scenarios) and **S4** (predictive maintenance) belong to the *service phase* which holds a different kind of objective, especially more focusing on the user-orientated aspects than the *manufacturing phase*. Because of that, we predict a new set of challenges and findings will arise, making them worthy of further investigations. Table 6.1 and 6.2 present the tentative requirements for **S3** and **S4**.

| Requirements | |
|--------------|---|
| ID | Requirement description |
| R1 | The scenario-specific data shall be handled separately from the master data. |
| R2 | The scenarios can be added, reset, and removed without affecting the master schedule. |
| R3 | The iterations of scenario may be stored and managed under version control. |

Table 6.1: S3 requirements

| Requirements | |
|--------------|---|
| ID | Requirement description |
| R1 | Dynamic data acquisition shall be used to update the static attributes in the predictive model. |
| R2 | Real-time measurements and simulated results shall be used in the retrofitting of the predictive model. |
| R3 | Data pruning and refining should be done before applying them to the predictive model |

Table 6.2: S4 requirements

In the requirements for **S4** we can see a large proportion directs to the data processing aspect of the predictive model, it is because we plan to adopt a data-driven modelling approach for this service. It will be unlike the mechanistic models used in **S1** to **S3**, as the data quantity and quality will have greater influences on the final results.

Likewise the KPIs for **S3** and **S4** are also proposed in Table 6.3 and Table 6.4. Note that these are still in early stage of design therefore are subject to substantial refinements.

| KPIs | | | |
|-------------|---------------|---|-----------------|
| ID | Name | Description | Verifies |
| KPI1 | Scalability | To what extent are the additional space and time bounded. | R1, R2, R3 |
| KPI2 | Accessibility | Are the model instances of the scenarios easily accessible to the operator. | R1, R3 |
| KPI3 | Modularity | Is changeover of the setup for each scenario easily adaptable. | R2 |

Table 6.3: S3 KPIs

| KPIs | | | |
|-------------|------------------|---|-----------------|
| ID | Name | Description | Verifies |
| KPI1 | Accuracy | What is the error of the predictive model against real-world measurement. | R1, R2 |
| KPI2 | Data sufficiency | How missing data may affect the overall performance of predictive model. | R1, R2 |
| KPI3 | Data quality | How much volume, variety, and veracity of the data is lost as the result of transferring and merging. | R3 |

Table 6.4: S4 KPIs

Bibliography

- [1] M. W. Grieves, “Virtually intelligent product systems: Digital and physical twins,” *Complex Systems Engineering: Theory and Practice*, pp. 175–200, 1 2019. [Online]. Available: <https://arc.aiaa.org/doi/pdf/10.2514/5.9781624105654.0175.0200> 1
- [2] Boeing, “Key findings on airplane economic life,” 3 2013. [Online]. Available: https://www.boeing.com/assets/pdf/commercial/aircraft_economic_life_whitepaper.pdf 1
- [3] P. Zheng, H. wang, Z. Sang, R. Y. Zhong, Y. Liu, C. Liu, K. Mubarak, S. Yu, and X. Xu, “Smart manufacturing systems for industry 4.0: Conceptual framework, scenarios, and future perspectives,” *Frontiers of Mechanical Engineering 2018 13:2*, vol. 13, pp. 137–150, 1 2018. [Online]. Available: <https://link.springer.com/article/10.1007/s11465-018-0499-5> 1
- [4] H. Kwon, S. An, H.-Y. Lee, W. C. Cha, S. Kim, M. Cho, and H.-J. Kong, “Review of smart hospital services in real healthcare environments,” *Healthcare informatics research*, vol. 28, pp. 3–15, 1 2022. [Online]. Available: <https://pubmed.ncbi.nlm.nih.gov/35172086> 1
- [5] Y. Duan, J. S. Edwards, and Y. K. Dwivedi, “Artificial intelligence for decision making in the era of big data – evolution, challenges and research agenda,” *International Journal of Information Management*, vol. 48, pp. 63–71, 10 2019. 1
- [6] PwC, “Industry 4.0 - publications.” [Online]. Available: <https://www.pwc.nl/en/publicaties/industrie-4-0.html> 2
- [7] IBM, “What is a digital twin?” [Online]. Available: <https://www.ibm.com/topics/what-is-a-digital-twin> 2
- [8] 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. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-58167-1_4 2
- [9] A. Kossiakoff, W. N. Sweet, S. J. Seymour, and S. M. Biemer, *Systems engineering principles and practice*. John Wiley & Sons, 2011, vol. 83. 2
- [10] S. Boschert and R. Rosen, “Digital twin-the simulation aspect,” *Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and Their Designers*, pp. 59–74, 1 2016. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-32156-1_5 2
- [11] M. Macchi, I. Roda, E. Negri, and L. Fumagalli, “Exploring the role of digital twin for asset lifecycle management,” *IFAC-PapersOnLine*, vol. 51, pp. 790–795, 1 2018. 2
- [12] K. Y. H. Lim, P. Zheng, and C. H. Chen, “A state-of-the-art survey of digital twin: techniques, engineering product lifecycle management and business innovation perspectives,” *Journal of Intelligent Manufacturing 2019 31:6*, vol. 31, pp. 1313–1337, 11 2019. [Online]. Available: <https://link.springer.com/article/10.1007/s10845-019-01512-w> 2

-
- [13] F. Ansari, S. Nixdorf, and W. Sihn, “Insurability of cyber physical production systems: How does digital twin improve predictability of failure risk?” *IFAC-PapersOnLine*, vol. 53, pp. 295–300, 1 2020. 2
- [14] H. Cai, J. Zhu, and W. Zhang, “Quality deviation control for aircraft using digital twin,” *Journal of Computing and Information Science in Engineering*, vol. 21, 6 2021. [Online]. Available: <https://asmedigitalcollection.asme.org/computingengineering/article/21/3/031008/1102047/Quality-Deviation-Control-for-Aircraft-Using> 2
- [15] D. G. Broo, M. Bravo-Haro, and J. Schooling, “Design and implementation of a smart infrastructure digital twin,” *Automation in Construction*, vol. 136, p. 104171, 4 2022. 2
- [16] M. Liu, S. Fang, H. Dong, and C. Xu, “Review of digital twin about concepts, technologies, and industrial applications,” *Journal of Manufacturing Systems*, vol. 58, pp. 346–361, 1 2021. 2, 16
- [17] M. Shafto, M. Conroy, R. Doyle, E. Glaessgen, C. Kemp, J. LeMoigne, and L. Wang, “Modeling, simulation, information technology and processing roadmap,” *National Aeronautics and Space Administration*, vol. 32, pp. 1–38, 2012. 2
- [18] M. Schluse, M. Priggemeyer, L. Atorf, and J. Rossmann, “Experimentable digital twins-streamlining simulation-based systems engineering for industry 4.0,” *IEEE Transactions on Industrial Informatics*, vol. 14, pp. 1722–1731, 4 2018. 2
- [19] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, “Digital twin in manufacturing: A categorical literature review and classification,” *IFAC-PapersOnLine*, vol. 51, pp. 1016–1022, 1 2018. 3
- [20] I. A. Udugama, C. L. Gargalo, Y. Yamashita, M. A. Taube, A. Palazoglu, B. R. Young, K. V. Gernaey, M. Kulahci, and C. Bayer, “The role of big data in industrial (bio)chemical process operations,” *Industrial and Engineering Chemistry Research*, vol. 59, pp. 15 283–15 297, 8 2020. [Online]. Available: <https://pubs.acs.org/doi/full/10.1021/acs.iecr.0c01872> 3
- [21] 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, 2021. [Online]. Available: <https://doi.org/10.1007/s43393-021-00024-0> 3, 7, 47
- [22] B. R. Barricelli, E. Casiraghi, J. Gliozzo, A. Petrini, and S. Valtolina, “Human digital twin for fitness management,” *IEEE Access*, vol. 8, pp. 26 637–26 664, 2020. 3
- [23] 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, 9 2017. 4
- [24] Mathworks, “Simulink - simulation and model-based design.” [Online]. Available: <https://nl.mathworks.com/products/simulink.html> 5
- [25] Dassault Systèmes, “Catia.” [Online]. Available: <https://www.3ds.com/products-services/catia/> 5
- [26] M. van den Brand, L. Cleophas, R. Gunasekaran, B. Haverkort, D. A. Negrin, and H. M. Muctadir, “Models meet data: Challenges to create virtual entities for digital twins,” *Companion Proceedings - 24th International Conference on Model-Driven Engineering Languages and Systems, MODELS-C 2021*, pp. 225–228, 2021. 5
- [27] D. A. Negrin, L. Cleophas, and M. van den Brand, “Using Ptolemy II as a framework for virtual entity integration and orchestration in digital twins,” *Companion Proceedings - 24th International Conference on Model-Driven Engineering Languages and Systems, MODELS-C 2021*, pp. 233–236, 2021. 5
-

- [28] C. Ptolemaeus, *System design, modeling, and simulation: using Ptolemy II*. Ptolemy.org, 2014, vol. 1. 6, 12, 13, 18, 19, 24
- [29] C. L. Gargalo, S. C. de Las Heras, M. N. Jones, I. Udugama, S. S. Mansouri, U. Krühne, and K. V. Gernaey, “Towards the development of digital twins for the bio-manufacturing industry,” *Advances in biochemical engineering/biotechnology*, vol. 176, pp. 1–34, 2020. [Online]. Available: <https://link.springer.com/chapter/10.1007/10.2020.142> 7, 8
- [30] A. Rasheed, O. San, and T. Kvamsdal, “Digital twin: Values, challenges and enablers from a modeling perspective,” *IEEE Access*, vol. 8, pp. 21 980–22 012, 2020. 7
- [31] “Tensorflow.” [Online]. Available: <https://www.tensorflow.org/> 8
- [32] “Pytorch.” [Online]. Available: <https://pytorch.org/> 8
- [33] H. Narayanan, M. F. Luna, M. von Stosch, M. N. C. Bournazou, G. Polotti, M. Morbidelli, A. Butté, and M. Sokolov, “Bioprocessing in the digital age: The role of process models,” *Biotechnology Journal*, vol. 15, p. 1900172, 1 2020. [Online]. Available: <https://onlinelibrary.wiley.com/doi/full/10.1002/biot.201900172> 8
- [34] T. Zhou, R. Gani, and K. Sundmacher, “Hybrid data-driven and mechanistic modeling approaches for multiscale material and process design,” *Engineering*, vol. 7, pp. 1231–1238, 9 2021. 8
- [35] H. Jiang, S. Qin, J. Fu, J. Zhang, and G. Ding, “How to model and implement connections between physical and virtual models for digital twin application,” *Journal of Manufacturing Systems*, vol. 58, pp. 36–51, 1 2021. 8
- [36] S. C. Rutan, “Recursive parameter estimation,” *Journal of Chemometrics*, vol. 4, pp. 103–121, 3 1990. [Online]. Available: <https://onlinelibrary.wiley.com/doi/full/10.1002/cem.1180040203> 8
- [37] R. M. Oisioviici and S. L. Cruz, “State estimation of batch distillation columns using an extended kalman filter,” *Chemical Engineering Science*, vol. 55, pp. 4667–4680, 10 2000. 8
- [38] D. Krämer and R. King, “On-line monitoring of substrates and biomass using near-infrared spectroscopy and model-based state estimation for enzyme production by *s. cerevisiae*,” *IFAC-PapersOnLine*, vol. 49, pp. 609–614, 1 2016. 8
- [39] “PID theory explained - NI.” [Online]. Available: <https://www.ni.com/nl-nl/innovations/white-papers/06/pid-theory-explained.html> 8
- [40] M. S. Hong, K. A. Severson, M. Jiang, A. E. Lu, J. C. Love, and R. D. Braatz, “Challenges and opportunities in biopharmaceutical manufacturing control,” *Computers & Chemical Engineering*, vol. 110, pp. 106–114, 2 2018. 8
- [41] “the cape-open laboratories network — expanding process modelling capability through software interoperability standards.” [Online]. Available: <https://www.colan.org/> 9
- [42] “Functional mock-up interface.” [Online]. Available: <https://fmi-standard.org/> 9
- [43] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauss, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf, “The functional mockup interface for tool independent exchange of simulation models,” *Proceedings from the 8th International Modelica Conference, Technical Univeristy, Dresden, Germany*, vol. 63, pp. 105–114, 6 2011. 9, 18, 19
- [44] “Functional mock-up interface specification.” [Online]. Available: <https://fmi-standard.org/docs/3.0-dev/> 9, 52

-
- [45] J. P. Belaud and M. Pons, "Open software architecture for process simulation: The current status of cape-open standard," *Computer Aided Chemical Engineering*, vol. 10, pp. 847–852, 1 2002. 9
- [46] "SysML open source project - what is SysML? who created it?" [Online]. Available: <https://sysml.org/> 10
- [47] "Ptolemy project home page." [Online]. Available: <https://ptolemy.berkeley.edu/> 10
- [48] "OMG system modeling language specification version 1.0." [Online]. Available: <https://www.omg.org/spec/SysML/1.0/About-SysML/> 11
- [49] IBM, "Engineering systems design rhapsody." [Online]. Available: <https://www.ibm.com/products/systems-design-rhapsody> 11
- [50] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *IEEE Software*, vol. 33, pp. 94–100, 5 2016. 11
- [51] 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*, p. 668, 10 2020. [Online]. Available: <https://doi.org/10.1145/3417990.3421446> 11, 12, 22
- [52] D. Preuveneers, W. Joosen, and E. Ilie-Zudor, "Robust digital twin compositions for industry 4.0 smart manufacturing systems," *Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOCW*, vol. 2018-October, pp. 69–78, 11 2018. 12
- [53] A. Borghesi, G. D. Modica, P. Bellavista, V. Gowtham, A. Willner, D. Nehls, F. Kintzler, S. Cejka, S. R. Tisbeni, A. Costantini, M. Galletti, M. Antonacci, and J. C. Ahouangonou, "Iotwins: Design and implementation of a platform for the management of digital twins in industrial scenarios," *Proceedings - 21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2021*, pp. 625–633, 5 2021. 12
- [54] M. H. Hung, Y. C. Lin, H. C. Hsiao, C. C. Chen, K. C. Lai, Y. M. Hsieh, H. Tieng, T. H. Tsai, H. C. Huang, H. C. Yang, and F. T. Cheng, "A novel implementation framework of digital twins for intelligent manufacturing based on container technology and cloud manufacturing services," *IEEE Transactions on Automation Science and Engineering*, 2022. 12
- [55] "Kubernetes." [Online]. Available: <https://kubernetes.io/> 12
- [56] S. Tripakis and E. A. Lee, "Fundamental algorithms for system modeling, analysis, and optimization," 2014. 13
- [57] L. Mears, S. M. Stocks, M. O. Albaek, B. Cassells, G. Sin, and K. V. Gernaey, "A novel model-based control strategy for aerobic filamentous fungal fed-batch fermentation processes," *Biotechnology and Bioengineering*, vol. 114, pp. 1459–1468, 7 2017. [Online]. Available: <https://onlinelibrary.wiley.com/doi/full/10.1002/bit.26274> 13
- [58] P. C. Lopez, I. A. Udugama, S. T. Thomsen, C. Bayer, H. Junicke, and K. V. Gernaey, "Promoting the co-utilisation of glucose and xylose in lignocellulosic ethanol fermentations using a data-driven feed-back controller," *Biotechnology for Biofuels*, vol. 13, pp. 1–14, 12 2020. [Online]. Available: <https://biotechnologyforbiofuels.biomedcentral.com/articles/10.1186/s13068-020-01829-2> 13
- [59] F. Feidl, S. Vogg, M. Wolf, M. Podobnik, C. Ruggeri, N. Ulmer, R. Wälchli, J. Souquet, H. Broly, A. Butté, and M. Morbidelli, "Process-wide control and automation of an integrated continuous manufacturing platform for antibodies," *Biotechnology and Bioengineering*, vol. 117, pp. 1367–1380, 5 2020. [Online]. Available: <https://onlinelibrary.wiley.com/doi/full/10.1002/bit.27296> 13
-

- [60] T. Eppinger, G. Longwell, P. Mas, K. Goodheart, U. Badiali, and R. Aglave, "Increase food production efficiency using the executable digital twin (xdt)," *Chemical Engineering Transactions*, vol. 87, pp. 37–42, 7 2021. [Online]. Available: <https://www.cetjournal.it/index.php/cet/article/view/CET2187007> 14
- [61] G. Tolksdorf, E. Esche, J. van Baten, and G. Wozny, "Taylor-made modeling and solution of novel process units by modular cape-open-based flowsheeting," *Computer Aided Chemical Engineering*, vol. 38, pp. 787–792, 2016. [Online]. Available: <http://dx.doi.org/10.1016/B978-0-444-63428-3.50136-3> 18
- [62] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Bateh, H. Tummescheit, and C. Sureshkumar, "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, 3 2014. 18
- [63] D. Krone, E. Esche, N. Asprien, M. Skiborowski, and J. U. Repke, "Conceptual design based on superstructure optimization in gams with accurate thermodynamic models," *Computer Aided Chemical Engineering*, vol. 48, pp. 15–20, 1 2020. 18
- [64] E. A. Lee and S. Neuendorffer, "Actor-oriented models for codesign," *Formal Methods and Models for System Design*, pp. 33–56, 2004. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4020-8052-4_2 18
- [65] 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, 11 2018. 19
- [66] ThingsBoard, "ThingsBoard - open-source IoT platform." [Online]. Available: <https://thingsboard.io/> 23
- [67] OpenModelica, "OMEdit – OpenModelica connection editor." [Online]. Available: <https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omedit.html> 26
- [68] Modelica Association, "System structure and parameterization," 2019. [Online]. Available: <http://www.opensource.org/licenses/bsd-license.html> 26
- [69] GitHub, "GitHub Actions documentation." [Online]. Available: <https://docs.github.com/en/actions> 26
- [70] Docker Inc, "Docker documentation." [Online]. Available: <https://docs.docker.com/> 26
- [71] Apache, "Apache Kafka." [Online]. Available: <https://kafka.apache.org/documentation/> 26, 51
- [72] H. Helgers, A. Schmidt, and J. Strube, "Towards autonomous process control—digital twin for cho cell-based antibody manufacturing using a dynamic metabolic model," *Processes*, vol. 10, 2022. [Online]. Available: <https://www.mdpi.com/2227-9717/10/2/316> 47

Appendix A

DT workflow in bioprocessing domains

The 5D model introduced in Section 1.2 provides a reference for entity management in DTs. Nonetheless, in the field of bioprocessing, there are other aspects at which one often finds important to inspect. One of them is the maturity of the DT models. For this reason, Udugama et al. [21] propose a five-step workflow (illustrated in Figure A.1) to implement a DT of bioprocessing domain. The workflow progresses in increasing order of mathematical complexity and functional requirements.

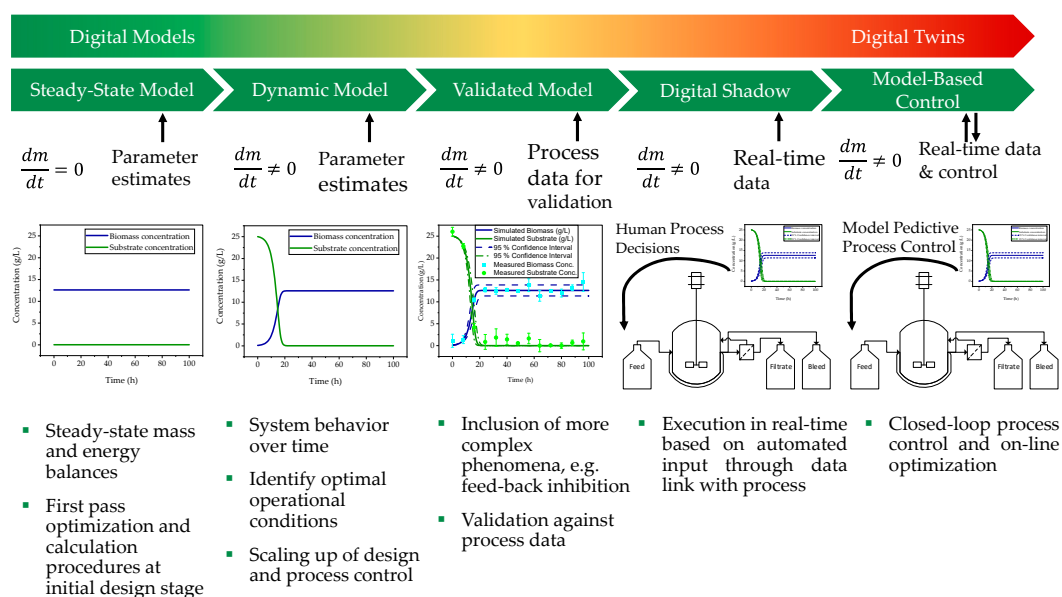


Figure A.1: The five-step implementation. This workflow has been adopted in a design of DT for monoclonal antibodies manufacturing [72].

1. **Steady-state model:** consisted of a mass and energy balance of the different key compounds in a reaction. These models are mathematical expressions of a process that are not time-dependent and hence carry no accumulation term.
2. **Dynamic model:** contains time-based derivative terms on all variables of interest.

3. **Validated model:** extends the capabilities of dynamic process models, such that it needs to be validated against process data obtained from an actual physical process.
4. **Digital Shadow:** consistent with the definition of Section 1.2; An one-way real-time monitoring model.
5. **Digital Twin:** consistent with the definition of Section 1.2; A two-way real-time monitoring and control model.

Appendix B

Microbrewery physical workbench

This section presents the workbench setup of the microbrewery DT. All the equipment are settled in a typical living room. A Raspberry Pi is used for aggregating all sensor data (except for those come from Airlock, they use a dedicated webhook server which is included as a part of product supports) before further processing in the DT. Figure B.1 shows the fermenter setup.



Figure B.1: The fermenter before (left) and after (right) being filled up with wort.

The other PEs that do not directly attach to the fermenter are installed in the same room, shown in Figure B.2.

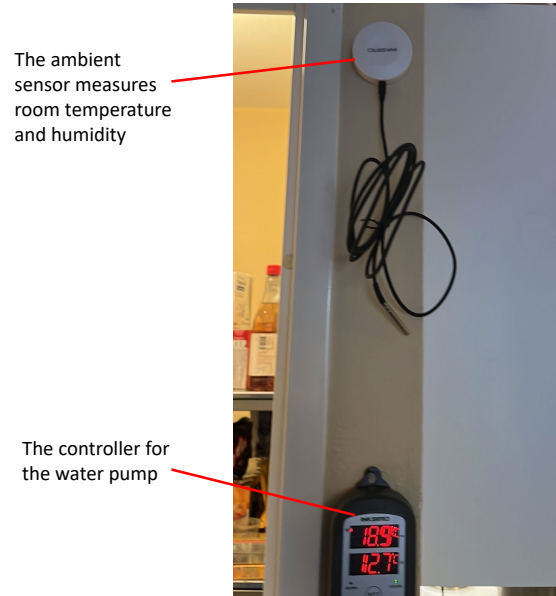


Figure B.2: More devices at the workbench

Figure B.3 shows the overall network topology of the workbench.

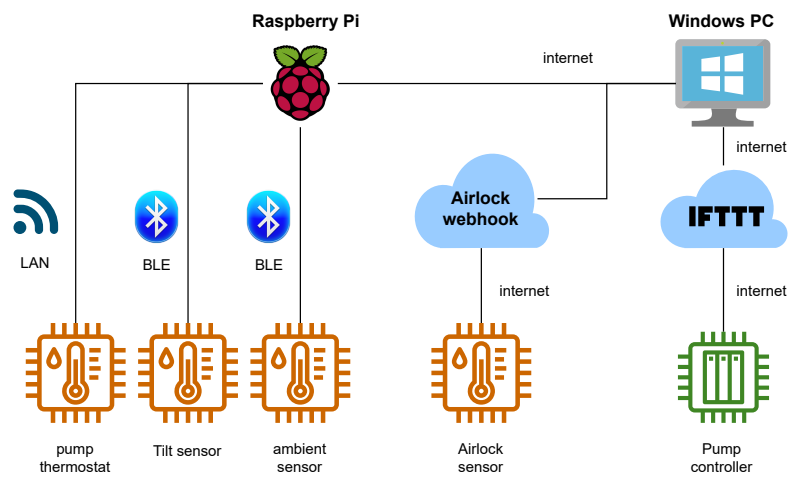


Figure B.3: Network topology of the workbench. Airlock sensor has an officially supported cloud server. The PC can send the control signals to the pump controller via IFTTT, which is a popular commercial service for home/smart phone automation.

Appendix C

Kafka primer

Kafka has been used extensively alongside our frameworks as a data management tool. This section will briefly introduce the main concepts of Kafka, and the advantages that make it suitable under DT contexts.

Kafka is considered an event streaming platform [71] that is based on the publish/subscribe design pattern (illustrated in Figure C.1). It supports three fundamental aspects:

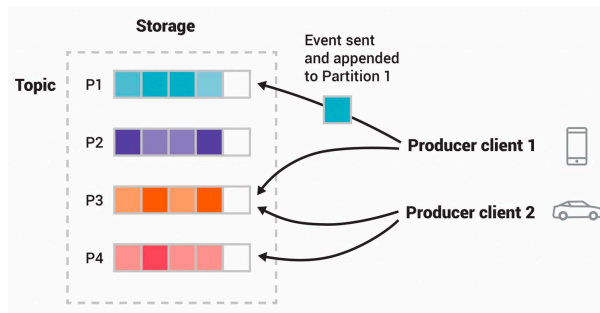


Figure C.1: An example Kafka architecture [71]

- **Publish** and **Subscribe**: the “producer” publishes events which can be subscribed by the “consumer”. The event streams are maintained by the Kafka server such that the producer and consumer can be agnostic and decoupled to each other.
- **Process**: referred to the enrichment and transformations of the event data. Processing can happen in real-time or in retrospect.
- **Storage**: The event data can be stored for a duration chosen by the user.

What makes Kafka stand out from alternative solutions of similar style (e.g., MQTT) is its emphasis on durable and persistent messages, which allows data manipulation being done either when the messages occur or retrospectively. This is important as DTs often deal with both real-time and historical data in a hybrid fashion. In Kafka the persistent storage can be accessed by “offset tracking”, a mechanism analogous to the pointer of a data structure.

Another DT aspect Kafka manages to address is the compartmentalization of data semantics through “topic” assignments. Conventional SQL-based approach depends on active query to data tables which sits passively in the binary objects. In contrast, Kafka turns the situation in which the data are proactively partitioned to distinct topics, while the users are ensured that different data compartments will be independent and readily accessible.

Appendix D

FMI revisit

This section explains why the `FMUImporter` of Ptolemy II has failed to perform the FMUs in our case study. Our troubleshooting results indicate the root cause is due to the improper calling sequence of the FMI functions.

It begins with the FMUs used in the microbrewery DT having the property of nonlinear algebraic loop, that is, when the outputs and inputs of the model have mutual dependencies. According to the official FMI specification [44], it states that such algebraic loops may be solved in the *Initialization Mode* by setting initial values to the involved inputs/outputs, which allows the solver to iteratively compute them to a convergence—when the interdependent input-output pair reaches an arbitrarily small difference. Listing D.1 shows the pseudocode of what a valid calling sequence should look like.

```
instantiate()
setupExperiment(startTime=startTime, stopTime=stopTime)
enterInitializationMode()
setReal(inputs, values) # setting values to model variables
exitInitializationMode()
```

Listing D.1: Resolving algebraic loop in the Initialization Mode

However, the internal implementation of `FMUImporter` does not follow this strictly, as one can see in Figure D.1, in there the initialization function is terminated before setting the start values. Consequentially, the output value returns NaN since the FMU could not resolve the algebraic loop.

```
FMUImporter._fmiInitialize(): about to invoke the fmi setup experiment function
FMUImporter._fmiInitialize(): about to invoke the fmi enter initialization function
FMUImporter._fmiInitialize(): about to invoke the fmi exit initialization function
FMUImporter._fmiInitialize(): about to request refiring if necessary.
FMUImporter._fmiInitialize(): about to record FMU state.
Initialized FMU.
FMUImporter.initialize() call completed.
Called prefire()
Called fire()
FMUImporter.fire() at time 0.0 and microstep 0
Setting start value of input Temperature to 0.0
Setting start value of input sg_0 to 0.0
Setting start value of input grams_yeast to 0.0
Setting start value of input batch_volume to 0.0
Setting start value of input time to NaN
FMUImporter.fire(): set input variable Temperature to 299.15
FMUImporter.fire(): set input variable sg_0 to 1.054
FMUImporter.fire(): set input variable grams_yeast to 11.5
FMUImporter.fire(): set input variable batch_volume to 20.0
⋮
FMUImporter.fire(): Output c et oh sends value NaN at time 5.0 and microstep 0
```

Figure D.1: Incorrect function calling sequence by Ptolemy II

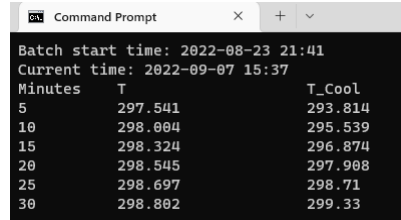
Due to this reason, along with several other limitations of Ptolemy II regarding its supports for FMI, we decide not to adopt the `FMUImporter` in the microbrewery DT implementation.

Appendix E

Human-in-the-loop control scheme

This section dedicates to showcase the human-in-the-loop concept for S2—Production Control—of the microbrewery DT. For the context, a water pump is used regulate the fermentation temperature by circulating the water from a ice-cooled bucket to the fermenter and back, forming a closed water-cooling circuit.

In the original S2, The models in VEs generate control signals that switch the power of the water pump in PEs. This operation is fully automated. Alternatively, in the human-in-the-loop version, the projection of water-cooling heat transfer is informed (via a client terminal as shown in Figure E.1) to the operator by the models, enabling the operator to make human judgments about replacing the ice in the bucket in order to alter the effect time of the temperature regulation. The two control schemes are illustrated Figure E.2.



```
Command Prompt
Batch start time: 2022-08-23 21:41
Current time: 2022-09-07 15:37
Minutes  T          T_Cool
5        297.541    293.814
10       298.004    295.539
15       298.324    296.874
20       298.545    297.908
25       298.697    298.71
30       298.802    299.33
```

Figure E.1: The client interface displaying a 30-minute heat predictions of the water-cooling setup

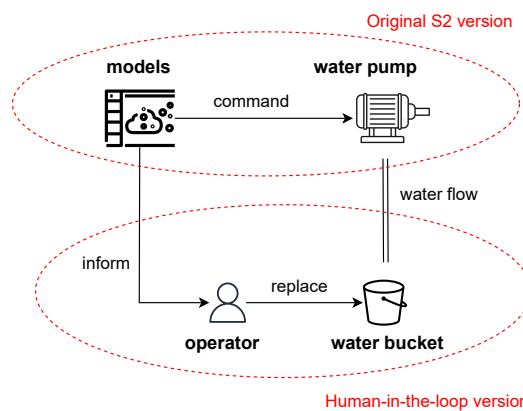


Figure E.2: The original S2 (top) and the human-in-the-loop control scheme (bottom)

The real water circuit setup at the workbench is shown in Figure E.3.

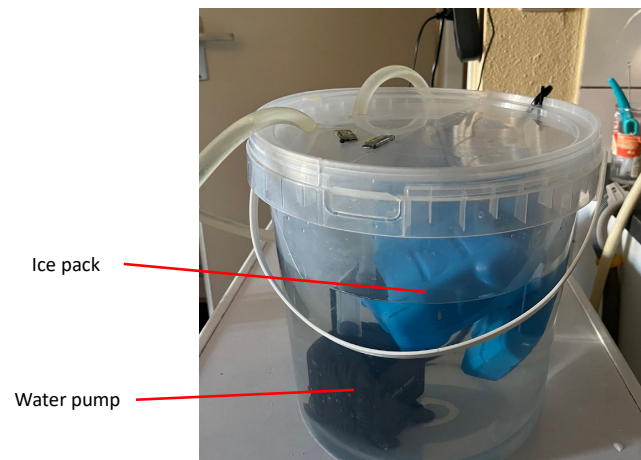


Figure E.3: Water circuit setup

This demonstration describes a proof-of-concept for human-machine integration in a DT, although not being the primary research objective of this study, we recognize its importance for especially DTs in the bioprocessing domains.