

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Virtual Factory Simulator

Nelson Filipe Barreto Almeida



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. António Lucas Soares

Second Supervisor: Eng. João Pedro Tavares Vieira Basto

Company Supervisor: Inês Novaes Bastos

July 21, 2020

Virtual Factory Simulator

Nelson Filipe Barreto Almeida

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. José Magalhães Cruz

External Examiner: Prof. Samuel Moniz

Supervisor: Prof. António Lucas Soares

July 21, 2020

Abstract

With the growth of Industry 4.0, companies are becoming aware of the power of technology and its beneficial tools capable of improving business processes' results, creating business value. Computer simulation is one of those powerful tools that allows the virtual observation of a process outcome without the need for a real physical environment.

This ability to reproduce a certain scenario can help companies improve in many areas, namely manufacturing. Simulation can enhance the manufacturing system by allowing the company to virtually experiment and validate different process designs and configurations, without requiring actual manufacturing equipment or any physical resources. The opportunity to test several system arrangement possibilities, on a simple screen, can have an important role in planning manufacturing processes and predicting their outcome while avoiding any financial loss.

The goal of this dissertation is to develop a simulation software capable of properly representing and simulating a manufacturing system by starting and cycling through the execution of a sequence of transactions that must consider some stochastic and probabilistic nature to try to properly replicate the manufacturing processes of a real factory.

The Virtual Factory Simulator is expected to positively contribute to other companies' decision making ability by allowing them to see a live production state 3D model along with different KPIs, dashboards and production reports. They can visualize how the system would behave and how production would be tracked, monitored and controlled using the *Critical Manufacturing's* manufacturing execution system.

Keywords: Simulation, Manufacturing, Industry 4.0

Resumo

Com o crescimento da Indústria 4.0, as empresas tornaram-se cientes do poder da tecnologia e das ferramentas informáticas capazes de tornar os processos de negócio mais eficientes, o que permite criar vantagens competitivas. Uma ferramenta informática importante é a simulação computacional, que permite observar, virtualmente, o resultado de um dado processo sem necessitar de um ambiente físico real.

Esta possibilidade de reproduzir um cenário pode trazer vantagens em várias áreas de uma empresa, nomeadamente, na área da produção. A simulação pode melhorar o sistema de manufactura pois permite a uma empresa experimentar e validar vários tipos de configurações sem necessitar de equipamento fabril, nem quaisquer recursos físicos. A possibilidade de testar sistemas de produção com diferentes arranjos, através de um ecrã, pode ter um papel importante no planeamento da produção e na previsão dos seus resultados, sem perda financeira.

O objetivo desta dissertação é desenvolver um *software* de simulação capaz de representar e simular um sistema de produção através da execução uma sequência de transações, que poderão apresentar natureza estocástica, de modo a replicar, de forma fidedigna, a manufactura de uma fábrica real.

É esperado que o *Virtual Factory Simulator* contribua para a tomada de decisão das empresas, através da visualização, em tempo real, dum modelo 3D do estado da fábrica virtual, juntamente com *KPIs*, *dashboards* e relatórios de produção. Também permite às empresas perceberem como é que o sistema de manufactura seria monitorizado e controlado através do *MES* da *Critical Manufacturing*.

Keywords: Simulação, Produção, Indústria 4.0

Acknowledgements

Credit should be given to those who, in any way, gave an important contribution to the dissertation development.

I would like to start by thanking my supervisors Eng. João Pedro Basto and Prof. António Lucas Soares for the help and guidance in this dissertation.

A special thanks goes to the entire *Critical Manufacturing* organization, especially to the facilitator Inês Bastos, for providing the right environment and tools to develop this project during the last 6 months.

I can't forgive to thank Rafael Marques for the convenient carpools.

Finally, a huge thanks to my family for always believing in me and supporting my decisions. Without you I wouldn't be capable of achieving what I did.

Nelson Filipe Barreto Almeida

*“In theory, there is no difference between theory and practice,
while in practice, there is.”*

Benjamin Brewster

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Objectives	2
1.4	Document Structure	2
2	State of The Art	5
2.1	Simulation	5
2.1.1	Concepts	6
2.1.2	One Model vs Multiple Models	8
2.1.3	Modeling Methods	9
2.1.4	Common Shop Floor Simulator Architecture	11
2.2	Manufacturing Execution Systems	12
2.2.1	Enterprise Resource Planning	12
2.2.2	MES, ERP and Shop Floor	13
2.2.3	MES Functions and Benefits	14
3	Methodological Approach	17
3.1	Problem Definition	17
3.1.1	Dates Mismatch	18
3.1.2	Simulator Functionalities	19
3.2	Interface between Simulator and MES	19
3.3	MES Database Architecture	20
3.4	Proposed Solution	21
3.4.1	Dates Mismatch	22
3.5	Working Methodology	23
4	Solution Implementation	25
4.1	Vocabulary	25
4.2	Events	27
4.3	Queries	29
4.4	Entity Relations	31
4.5	Locks & Multithreading	31
4.6	Classes & Methods	31
4.6.1	Simulator	32
4.6.2	EntityManager	33
4.6.3	EventSchedule	36
4.6.4	ProbabilityDistribution	36

4.6.5	Clock	37
4.6.6	AccelerationChange	38
4.6.7	Event	38
4.6.8	Query	40
4.6.9	Utils	40
4.6.10	Class Diagrams	41
4.7	Database Replication & Date Modification	42
4.7.1	ReplicationTables	42
4.7.2	Stored Procedures	42
4.7.3	Date Attributes	43
4.7.4	Date Modification Mathematical Operations	43
4.7.5	Service History	44
4.7.6	Date Modification	44
4.8	Executed Workplan	46
5	Validation Tests	49
5.1	Testing Methodology	49
5.2	Testing Scenarios	49
5.2.1	Case Study Basis	49
5.2.2	Sprint 2 - Case Study and Results	51
5.2.3	Sprint 3 - Case Study and Results	51
5.2.4	Sprint 4 - Case Study and Results	52
5.2.5	Sprint 5 - Case Study and Results	54
5.2.6	Sprints 6, 7 and 8 - Case Study and Results	55
6	Conclusions and Future Work	59
6.1	Conclusions	59
6.2	Future Work	60
	References	61

List of Figures

2.1	One model approach based on a supply chain scenario [1]	8
2.2	Multiple model approach based on a supply chain scenario [1]	9
2.3	Abstraction levels of simulation modeling methods [2]	9
2.4	Common architecture of a shop floor simulator	11
2.5	ISA S95 enterprise layers [3]	13
3.1	MES dashboard with fabLIVE view (on the left) and KPIs (on the right)	18
3.2	Data flow between MES and simulation software	20
3.3	MES database architecture	21
3.4	Scrum sprint process overview [4]	23
4.1	UML class diagram covering the entire software implementation	41
4.2	UML class diagram covering the events package and its associates	41
4.3	UML class diagram covering the queries package	42
5.1	Diagram explaining the manufacturing setup used as the basis case study	50
5.2	Diagram explaining the manufacturing setup used as sprint 3 case study	52
5.3	Diagram explaining the manufacturing setup used as sprint 4 case study	53
5.4	Diagram explaining the manufacturing setup used as sprint 5 case study	55
5.5	Diagram explaining the manufacturing setup used as Sprints 6 through 8 case study	57

List of Tables

5.1	Basis case study entities	50
5.2	Basis case study variables and values	50
5.3	Sprint 3 case study entities	51
5.4	Sprint 3 case study variables and values	51
5.5	Sprint 4 case study entities	53
5.6	Sprint 4 case study variables and values	53
5.7	Sprint 5 case study entities	54
5.8	Sprint 5 case study variables and values	55
5.9	Sprints 6 through 8 case study entities	56
5.10	Sprints 6 through 8 case study variables and values	56
5.11	Simulation information used to validate date modification	58

Abbreviations

MES	Manufacturing Execution System
KPI	Key Performance Indicator
FIFO	First-In-First-Out
ERP	Enterprise Resource Planning
MESA	Manufacturing Enterprise Solutions Association

Chapter 1

Introduction

This preliminary chapter presents the base project to be developed for the dissertation. Sections 1.1 and 1.2 explain how the problem was born and what area it falls into. Section 1.3 approaches the main goals to be achieved by the Virtual Factory Simulator. Lastly, the main structure of this document is presented in section 1.4, highlighting the topics discussed by subsequent chapters to improve the readability of the document.

1.1 Context

With the growth of Industry 4.0, companies are becoming aware of the power of technology and its beneficial tools that can be used to enhance productivity and further innovation in order to improve business results and gain more competitive advantages [5].

Business oriented tools and technologies have been invented to draw useful information that helps organizations gain valuable insights on past and current behaviors in order to predict future needs by producing innovative business models, projects, products, or services [5].

Computer simulation is a powerful tool that allows the virtual observation of a real-world process outcome over time without the need for a real physical environment. Simulation can be seen as a strong methodology to solve many real-world problems. It is used to describe and analyze the behavior of a system, create hypothetical scenarios, and aid in the design of real systems [6].

A company may have many sectors that can greatly benefit from simulation. One of them being manufacturing. In the current highly competitive business environment, the manufacturing industry is facing constant challenges to produce innovative products in a short time period. Consequently, production processes are becoming increasingly more complex and products become more versatile, intricate and inherently complicated [7].

Simulation modelling and analysis can combine with manufacturing in order to gain insight into complex production systems, allowing the development and testing of new operating policies and new systems, before implementing them, and gather information and knowledge without disturbing the actual system [7].

1.2 Motivation

The idea for this project came from *Critical Manufacturing*. A company focused on developing software for other companies. The vast majority of *Critical*'s clients are manufacturing enterprises, which inspired the creation of a demo showroom, at *Critical Manufacturing* offices, for customers to visually track their production processes.

Since it's impossible to achieve this goal with actual manufacturing equipment in its facilities, *Critical Manufacturing* intends to create a 3D virtual factory simulator, allowing companies to easily evaluate, graphically and statistically, how a given production system would behave on their factories.

Critical's main selling product is its MES software to aid companies in their manufacturing processes. A simulation module would greatly improve such system by replicating the manufacturing processes of factory models created through the MES interface. This way, enterprises can easily create and manipulate the production models and obtain their simulated outcome.

1.3 Objectives

The goal of this dissertation is to develop a simulation software capable of properly representing and simulating a shop floor's manufacturing system. The software should start by reading a model directly from MES, and then cycle through the execution of a sequence of material and equipment transactions that must consider some stochastic nature to try to properly replicate the manufacturing processes of a real factory.

The simulator will generate data reflecting the current state of the simulation to be used by MES rendering software modules. Thus, the customer can see a live 3D demonstration of production state, along with dashboards containing production reports and different KPIs like cycle time, queue and process times, resource utilization, etc.

An important aspect of the Virtual Factory Simulator is a time scale acceleration factor. Since some production processes may take hours to complete, it must be possible to configure an acceleration factor to fast forward the simulation.

1.4 Document Structure

The document is composed of six chapters, including the current one.

Chapter 2, analyzes the current state of the art regarding simulation and Manufacturing Execution Systems. There is an emphasis in simulation applied to manufacturing.

Chapter 3, describes the methodological approach to the simulation problem. A detailed problem description can be found.

Chapter 4, thoroughly explains all the important aspects of the proposed solution.

Chapter 5, presents all the testing scenarios used to validate the implemented solution.

Lastly, in chapter 6, conclusions drawn from the dissertation work are presented along with ideas for future work on this project.

Chapter 2

State of The Art

This chapter focuses on gathering the important aspects related to simulation, with an emphasis on manufacturing, used as a support base for developing the dissertation project. Additionally, an introduction to Manufacturing Execution Systems is also presented.

Section 2.1 starts with the definition of simulation and explains its advantages and disadvantages. This section contains four subsections. Subsection 2.1.1 explains the main concepts that revolve around simulation. Subsection 2.1.2 refers the possibility of using one or multiple models to simulate a system. In subsection 2.1.3 the three main methods of simulation modeling can be found. Lastly, the most common architecture for shop floor simulators is presented in subsection 2.1.4, using discrete event simulation.

The second section, 2.2, briefly explains the MES concept and contains three subsections. Subsection 2.2.1 is an introduction to ERP systems. Subsection 2.2.2 refers the interaction between ERP, MES and shop floor. Lastly, in subsection 2.2.3, MES main functionalities and benefits are presented.

2.1 Simulation

Robert E. Shannon (1975) [8] defines simulation as the process of designing a model of a real system and using it to conduct experiments for the purpose of studying the behavior of the system and evaluating various operating strategies. Later, Jerry Banks (1999) [6] explains it as the imitation of how a real-world process or system operates over time.

Simulation generates an artificial history of the system evolution over time, and, afterwards, explores that history to draw inferences concerning the operating characteristics of the real system that is represented. It is a strong methodology to solve many real-world problems. Simulation is used to describe and analyze the behavior of a system, create hypothetical scenarios, and aid in the design of real systems. It can model both existing and conceptual systems [6].

Thus, simulation can be used to study systems during the design planning phase, before they are implemented. Simulation is useful to predict how changes affect preexisting systems and estimate performance of new systems [9].

Simulation can be a powerful tool. However, it can also present certain setbacks. Pros and cons of simulation are listed below [6] [2] [10] [11].

Advantages:

- Test a proposed change or addition without expending resources;
- Change simulation speed, allowing to speed up uninteresting processes or slow them down to be thoroughly investigated;
- Explore hypothetical scenarios without disrupting the real system;
- Better understand the interactions among components of complex systems;
- Identify elements responsible for bottlenecking or restricting the system;
- Help to understand about how a system really operates;
- Detect flaws in the system design;
- Low investment associated;
- Analyze systems where other methods such as analytic calculations and linear programming fail;
- Provide training with less expenses when compared to on-the-job learning;
- Specify requirements for system design.

Disadvantages:

- Specialized training is required to build a simulation model;
- Obtained results may be difficult to interpret due to the possible randomness of results;
- Simulation modeling and analysis can be time consuming;
- May not be possible to adapt a simulation model to a different case scenario.

2.1.1 Concepts

There are several core concepts common to most simulation projects that must be well understood. They help to perceive how a simulation is executed.

A dictionary defining the main simulation concepts can be found below [6] [9] [10] [12] [13].

System

A system can be defined as a group of elements that interact with each other and collaborate towards a common goal. An automobile assembly line can be seen as a system, where machines, car parts and workers collaborate to build a functional vehicle.

Model

A model is an abstract representation of an actual system. It should have the minimum level of complexity necessary to solve the proposed problem.

There are several types of models: mathematical, descriptive, statistical, input-output and discrete-event. Discrete-event models represent the components of a system and their interactions, and are dynamic, i.e., the passage of time is an important factor. Most mathematical and statistical models are static, i.e., represent a system at a fixed point in time.

Dynamic models can be discrete or continuous. In discrete simulation the system only changes when an event occurs, ignoring what happens between two consecutive events. In continuous simulation the system is analyzed at every time unit that is spent.

Event

An event is an occurrence that changes the state of the system. Events can be internal or external, also known as endogenous or exogenous, respectively. Example of an endogenous event can be registering the order of a client in a store, which involves an element already present in the system. Example of an exogenous event can be a new client coming into the store, which involves an element not previously present in the system.

System State Variables

System state variables represent a collection of data used to define the state of the system at a given point in time.

Entity and Attributes

An entity represents an object of interest to the system and requires explicit definition. Entities cause changes in the system state and, without them, nothing would happen in a simulation. An entity can be dynamic if it moves through the system, or it can be static if it serves other entities.

Each entity has its set of attributes, which are characteristics used to describe it.

Resource

A resource is an entity that provides service to dynamic entities. Resources have restricted capacity and can serve one or multiple dynamic entities at the same time.

A dynamic entity can request one or more units of a resource. If the resource is not available at that time, the requesting entity joins a queue, or takes some other action. If permitted to capture the resource, the entity waits for a certain time and then releases the resource.

Typically, resources have two states: available and busy. But other possibilities exist including

failed, blocked or starved.

Queue

Entities are managed by allocating them to resources that provide service. This can result in an entity being suspended into the future, or joining a waiting queue.

Generally, queues are processed in FIFO order, but there are other possibilities.

Activity and Delay

An activity is the duration of time it takes for a task to be executed. The activity duration is known prior to its start, allowing to scheduled its completion. The duration can be deterministic or a random value based on a probability distribution.

A delay is an indefinite duration that is caused by some combination of system conditions. E.g., when an entity joins a queue for a resource, the time that it will remain in the queue may be unknown, leading to a delay.

The beginning and ending of an activity or delay are considered events.

2.1.2 One Model vs Multiple Models

S. Terzi et al. [14], using a supply chain scenario as example, refer two different structural paradigms to model a system. Using a single model or creating multiple small models.

In the single model approach (Fig. 2.1), only one model is used to represent and simulate the system as a whole. This is the most common procedure. For simple systems, this is the wisest route to take.

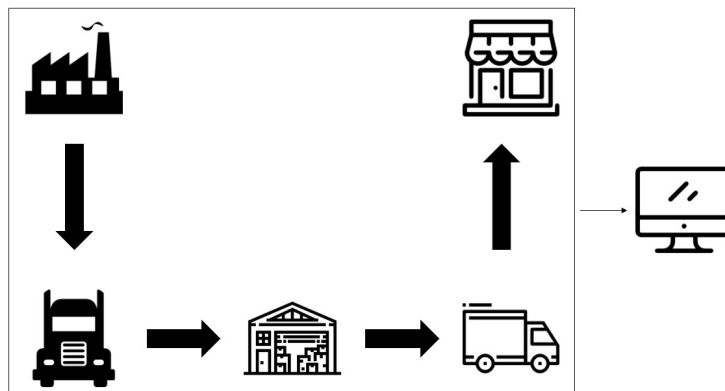


Figure 2.1: One model approach based on a supply chain scenario [1]

In the multiple model approach (Fig. 2.2), multiple smaller and simpler models are created based on the component of the system. The simulation of the whole system requires the execution of all the models, which may happen in parallel. Using multiple models increases re-utilization and versatility [1]. This procedure might be useful in systems with multiple complex components.

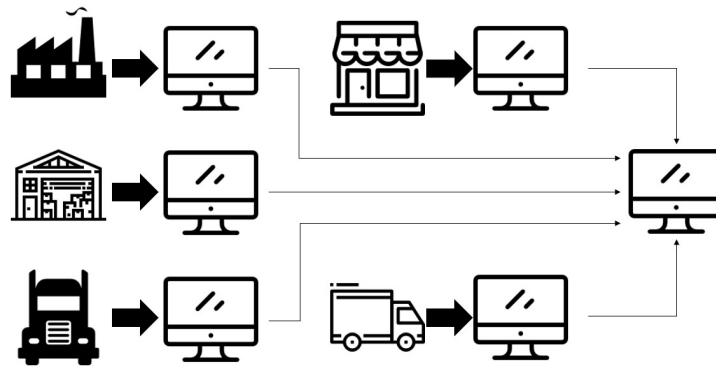


Figure 2.2: Multiple model approach based on a supply chain scenario [1]

2.1.3 Modeling Methods

A simulation modeling method allows to map the system under analysis into an abstract model. Ilya Grigoryev [2] refers three modeling methods:

- System Dynamics
- Agent Based Modeling
- Discrete Event Modeling

Each method presents different abstraction levels, as shown in Fig. 2.3.

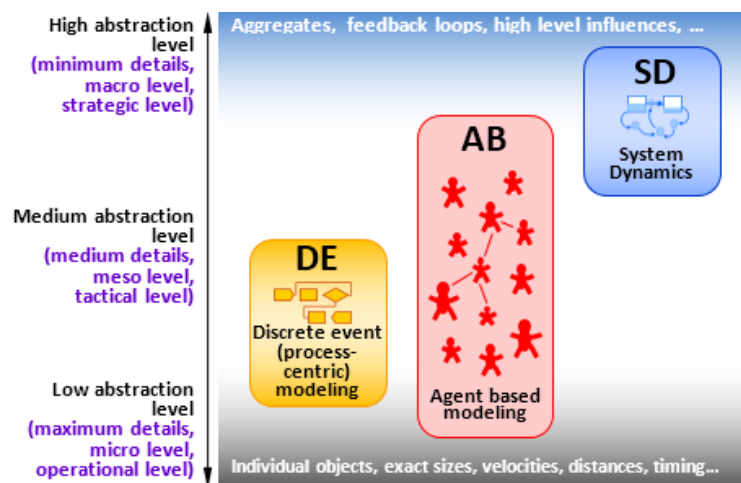


Figure 2.3: Abstraction levels of simulation modeling methods [2]

Each one of this methods suits different purposes, since each method fits different abstraction levels. In some occasions it may be useful to model different parts of the same system with different methods. This is called a multi-method approach [15].

• System Dynamics

System dynamics fits the higher abstraction levels, being typically used for strategic modeling. It assumes high levels of object aggregation, being used to represent discrete entities like people, products, events and others, by their quantities.

System dynamics is a methodology to better understand how the behaviour of dynamic systems evolves over time and how they perform. This method is based on continuous simulation [16].

System dynamics requires:

- Modeling a system as a closed structure that defines its own behaviour;
- Discovering the system's feedback loops (circular causality relationships);
- Identify stocks and flows that affect them.

Stocks are accumulations and characterize the system state, being considered the memory of the system and the source of unbalance. System dynamics only works with aggregates, meaning there are no individual entities. Flows are the rates at which a system state changes. Usually, stocks are expressed in quantities such as people, money, etc. Whereas flows are typically measured in quantities per time period such as clients per month or dollars per year [2].

• Agent Based Modeling

Agent based modeling covers a wide range of abstract levels, from low to medium-high, which means the detail level of its implementation can vary. The focus of this methodology is not on understanding the behaviour of the system. Its objective is to study the behaviour of individual entities (agents). When building a model, all the agents must be defined, as well as their behaviours. Agents can interact with each other and be inserted in different environments. The behaviour of the system is a consequence of how its agents behave and interact [2] [17].

There is no concept of waiting queues nor flows in agent based modeling. Modeled agents can execute actions and operations on their own initiative [18].

• Discrete Event Modeling

Discrete event modeling is useful in medium and low-medium abstraction contexts. Thus, it is used to simulate operational and tactical systems. This methodology is used to understand the behaviour of dynamic systems and how they perform over time, in different scenarios. Discrete event modeling focuses on processes to minutely represent a system [1].

Waiting queues are an important aspect of discrete event modeling. This methodology tries to model how entities move through. Entities are taken as individual elements. Discrete event models can be used in scenarios that require some variability, since it allows stochastic behaviour [1].

Antuela A. Tako et al. [16] performed a literature review to investigate the use of system dynamics and discrete event modeling in logistics and supply chain management (LSCM). They concluded that discrete event simulation is often used in production planning and scheduling, which is the LSCM area this dissertation project fits into.

2.1.4 Common Shop Floor Simulator Architecture

The project of this dissertation is focused in simulating shop floor production processes. Thus, it is useful to analyze the software architecture used in projects addressing shop floor simulation.

Charles McLean et al. [19] developed a simulation project to replicate the job shop operations of Kurt J. Lesker company, an international manufacturer and distributor of vacuum products.

Tiago Constante [20] developed, as a master's dissertation project, a simulation framework to simulate manufacturing systems encountered in shop floors.

K. Worapradya et al. [21] developed a shop floor simulator integrated with an optimization unit to support the decision of a regular MES software, using a conceptual steel making facility as case study.

All these projects were somehow focused on simulating shop floor processes. They all used discrete event simulation, with an architecture similar to the one presented in Fig. 2.4.

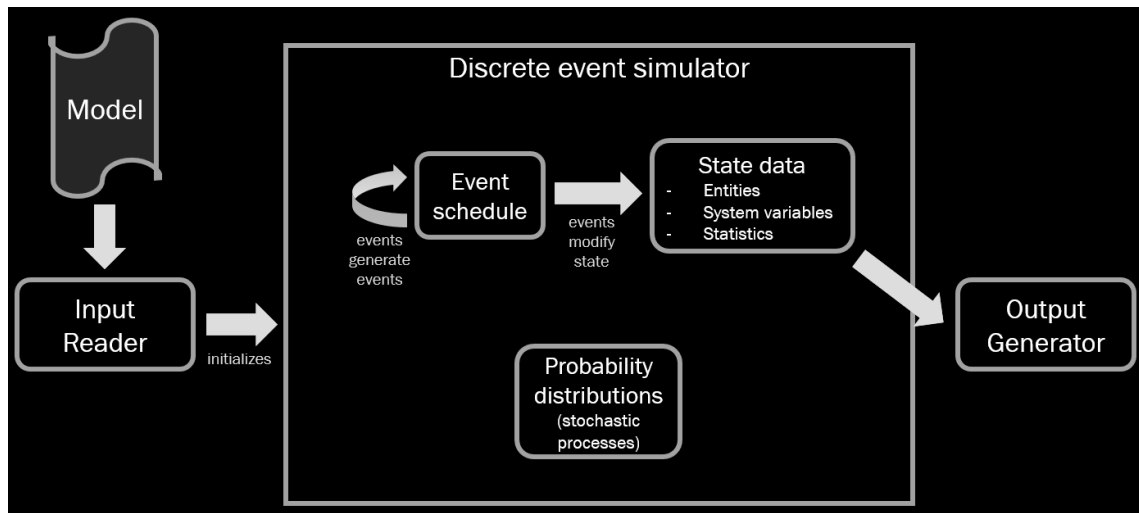


Figure 2.4: Common architecture of a shop floor simulator

The architecture contains several components responsible for maintaining the discrete event simulation workflow, which are described below.

Input Reader

The simulation starts with this component. It is responsible for reading the model (can originate from a file, response to a request, etc.), initializing the state and scheduling the first events of the simulation. It can also create the probability distributions if they are referred in the model.

State data

This component contains information describing the current state of the simulation. Contains the entities currently present in the system, system variables, e.g., the current simulation time and statistical data useful to evaluate production processes.

Event schedule

The event schedule contains a priority queue with the events to be processed in the future. The simulation advances by consuming the next event present in the schedule and performing all the simulation logic deriving from it.

The simulation logic may include modifying state data or creating new events to be added to the schedule. This component can be seen as the discrete event simulation engine.

Probability distributions

The probability distributions component is useful to simulate stochastic processes. Whenever a random number needs to be assigned to an activity duration or any other variable, this component is responsible for generating it using the proper probability distribution.

This module can be ignored in deterministic simulators.

Output Generator

This component is responsible for exporting the state data in a certain format that will be used by another software/component focused on displaying the information generated by the simulation.

2.2 Manufacturing Execution Systems

As stated by the *Manufacturing Enterprise Solutions Association* (MESA) [22], a Manufacturing Execution System (MES) provides information to optimize production activities from order launch to finished goods. Using real-time accurate data, an MES initiates, guides, reports and responds to plant activities as they occur. The rapid response to changing conditions, coupled with a focus on reducing counterproductive activities, leads to effective plant operations and processes. An MES can improve on-time delivery, inventory turns, gross margin, cash-flow performance and return on operational assets.

The MES concept was born from the manufacturing enterprises' need to fulfil market requirements from a point of view of reactivity, quality standards, cost reduction and deadlines [23].

To better understand the role of a Manufacturing Execution System, we first need to comprehend the Enterprise Resource Planning (ERP) concept.

2.2.1 Enterprise Resource Planning

An ERP system manages the main business functions within an enterprise, such as financial and cost accounting, sales and distribution, inventory management, human resources, production planning and manufacturing, supply chain, and customer information [24].

ERP gives an integrated real-time view of a company's core business processes, tied together by software applications and a common database transversal to the entire business, maintained by a database management system. This facilitates information flow between all business functions inside a company, and manages connections to outside stakeholders.

These systems track resources, such as cash, raw materials, and production capacity, along with the status of commitments made by the business, namely, customer orders, purchase orders, and employee payroll, no matter which department of the organization has entered the data into the system [25].

2.2.2 MES, ERP and Shop Floor

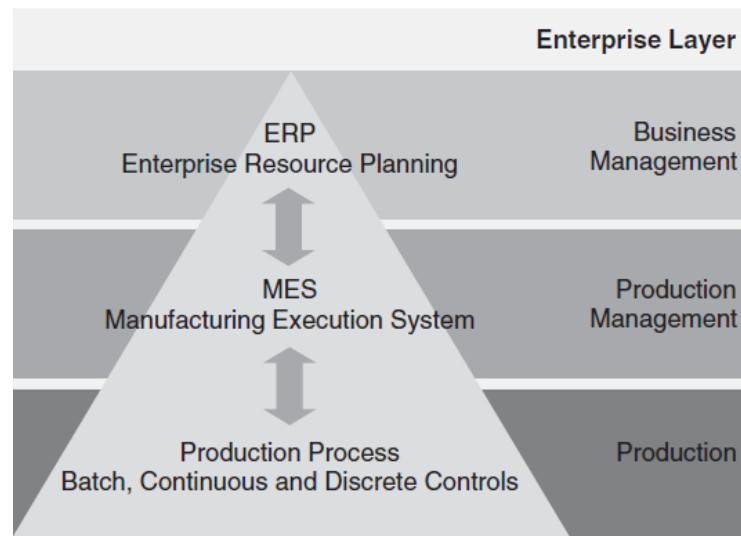


Figure 2.5: ISA S95 enterprise layers [3]

ISA S95, an international standard from the *Instrumentation, Systems, and Automation Society*, defines 3 distinct enterprise layers involved in the manufacturing environment (Fig. 2.5).

The ERP lies on the business management layer and is used largely for administration and accounting with the core functions of financial management and material management [3]. It can indeed be used as a production planning system that communicates directly with the production layer, which represents the shop floor where the production processes are executed, and provides production employees with relevant information. However, the information is often limited and requires too much time to travel from one layer to another, leading to inaccurate information [26]. Therefore, regulating production processes in real time with regard to target values is not possible by using only an ERP system [3].

An MES can be used as an intermediate layer for production management, to efficiently respond to minute-to-minute changes occurring at the shop floor. Different from an ERP system, which generally responds to daily changes [26]. MES closes this gap between the ERP level and actual production, acting as a link between ERP and shop floor while also providing a range of additional significant functions that cannot be performed by the systems of the other layers [3].

A manufacturing execution system is an on-line, integrated, computerized system that provides a set of tools and methods to effectively carry out production processes [26].

2.2.3 MES Functions and Benefits

MESA declared the following as essential functions to be provided by any MES solution [27]:

- Workflow sequence planning based on relevant basic conditions, such as setup time, processing time, available resources, etc.;
- Management and monitoring of relevant resources like staff, machines, tools, etc.;
- Control the production units' flow and immediately respond to events during ongoing production, adjusting the plan if necessary;
- Access to relevant information regarding the production processes (CAD, designs, test specifications, environmental compliance requirements, safety instructions, etc.) whenever required by staff members, with the possibility to record deviations;
- Log production-related operating data linked with the production unit;
- Record staff working hours, while handling cases of absence, holidays, etc.;
- Analyze measurement data in real time in order to ensure product quality and rapidly identify problems;
- Production process monitoring, including alarm management functions;
- Track operating material usage in order to initiate periodic and preventive maintenance tasks;
- Data recording across the entire production chain to ensure product traceability;
- Produce managerial key figures, in real time whenever possible, to facilitate performance analysis. Various display formats should be available to the user.

Due to the functions available on a manufacturing execution system, it can offer the following benefits [28]:

- Reduced manufacturing cycle time;
- Reduced data entry time;
- Reduced work-in-process inventory;
- Reduced lead time;
- Reduced paperwork;
- Improved product quality;
- Avoid loss of documents by keeping a digital version;

- Increased plant operations people productivity;
- Improved customer service;
- Faster response to unexpected events.

Chapter 3

Methodological Approach

This chapter thoroughly explains the simulation problem to be resolved along with the proposed solution to be implemented. The adopted working methodology is also presented.

Section 3.1 contains a detailed description of the simulation problem considering all the important MES aspects. Section 3.2 explains how the MES instance and the simulator integrate with each other, emphasizing the *fabLIVE* and KPI modules.

The MES database architecture is explained in section 3.3 to facilitate the understanding of the proposed solution presented in section 3.4. Finally, section 3.5 describes the working methodology employed throughout the dissertation development, adopted as a regular policy at *Critical's* offices.

3.1 Problem Definition

Critical Manufacturing's MES is the main selling product of the company. It's a software product that provides functionalities for customers to empower their factories' manufacturing processes. However, all those functionalities are focused on real-time production, without the possibility to explore hypothetical scenarios and their outcome. Hence, a simulation module capable of properly representing and simulating a shop floor's manufacturing system would represent a valuable addition to the MES solution.

The simulator should start by creating a factory model based on the shop floor data regarding production elements, such as resources, products, materials, etc., present on a given MES instance. The obtained model will serve as a basis for the simulation whose objective is to cycle through the execution of a sequence of material and equipment transactions, that must consider some stochastic nature, to replicate the manufacturing processes of the factory being represented in the MES instance.

As the actions are being performed by the simulator, data will be generated and introduced into the MES database. This data will fuel two important MES modules: *fabLIVE* and KPIs.

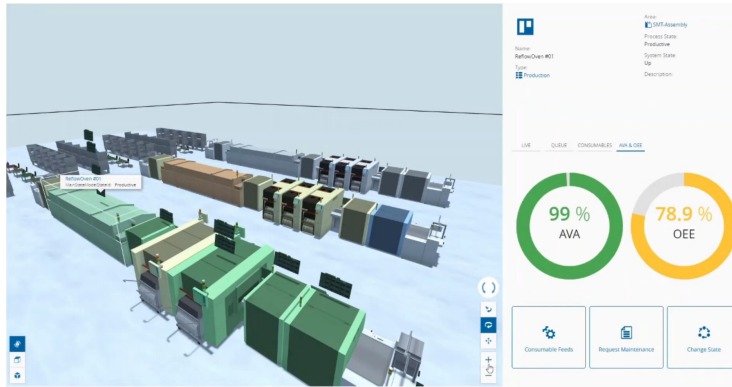


Figure 3.1: MES dashboard with fabLIVE view (on the left) and KPIs (on the right)

fabLIVE presents a graphical 3D view of the current MES factory state. A graphical environment with dynamic 3D model representations of resources and materials can be seen along with their production state.

Key Performance Indicators can be presented on a dashboard alongside the *fabLIVE* screen view (Fig. 3.1). KPI values provide real-time performance analysis of the manufacturing processes taking place at the MES instance.

As production transactions occur, KPI values and *fabLIVE*'s entities change to reflect the current production state, based on the information being retrieved from MES database.

The data generated from a simulation can also be further used for demonstrations to potential customers performed by presales staff.

3.1.1 Dates Mismatch

One mandatory feature that should be present in the simulator is a time scale acceleration factor. Since production processes might require several minutes or even hours to complete, the possibility to accelerate time allows the user to visualize them much faster.

The implementation of the time acceleration feature implies some level of date manipulation, since the simulated actions being accelerated must be executed at an earlier date. This simulation aspect presents an obstacle due to the way data is stored in MES database. When an action is performed, records are added to the database with date attributes automatically generated containing the current system date. Since the simulation is being fast forwarded, the true simulation date should be located in the future. This leads to generated dates not being in accordance with the simulation timetable.

The stored dates will later be used for KPI calculation purposes since certain KPIs are calculated for a given time interval. Therefore, a solution needs to be found to guarantee that KPIs reference the correct dates and time frames.

3.1.2 Simulator Functionalities

Besides the general simulation idea, it is important to describe specific features related to MES that should be present in the simulator.

These are the functionalities that should be integrated in the simulator:

- Send material to a resource's queue to be processed later, also known as dispatch and track in;
- Remove material from a resource after it has been processed and move it into the next production step, also known as track out and move next;
- Attribute material processing times following a probability distribution;
- Record material losses following a probability distribution;
- Restock material at the first production step of every production line once its quantity is below the stipulated threshold. This will guarantee a continuous production;
- Make resources unavailable. Both the unavailability occurrence and duration should follow a probability distribution;
- Choose between multiple resource options in the same production step;
- Accelerate simulation time.

3.2 Interface between Simulator and MES

Before the simulation starts, a factory setup must be previously created containing all the necessary entities like resources, materials, etc., necessary to represent the manufacturing processes to be simulated.

Once the simulator is executed, a primary loading action must take place before simulation starts. This initial activity will fetch from the MES instance all the entities that will be involved in the simulation and create a model around which the simulation will revolve.

After the loading phase ends, the simulation can then start and the MES operations that represent the execution of regular manufacturing actions will automatically be performed by the simulator.

Both the factory data fetching and MES operation execution will be performed through the API provided by MES.

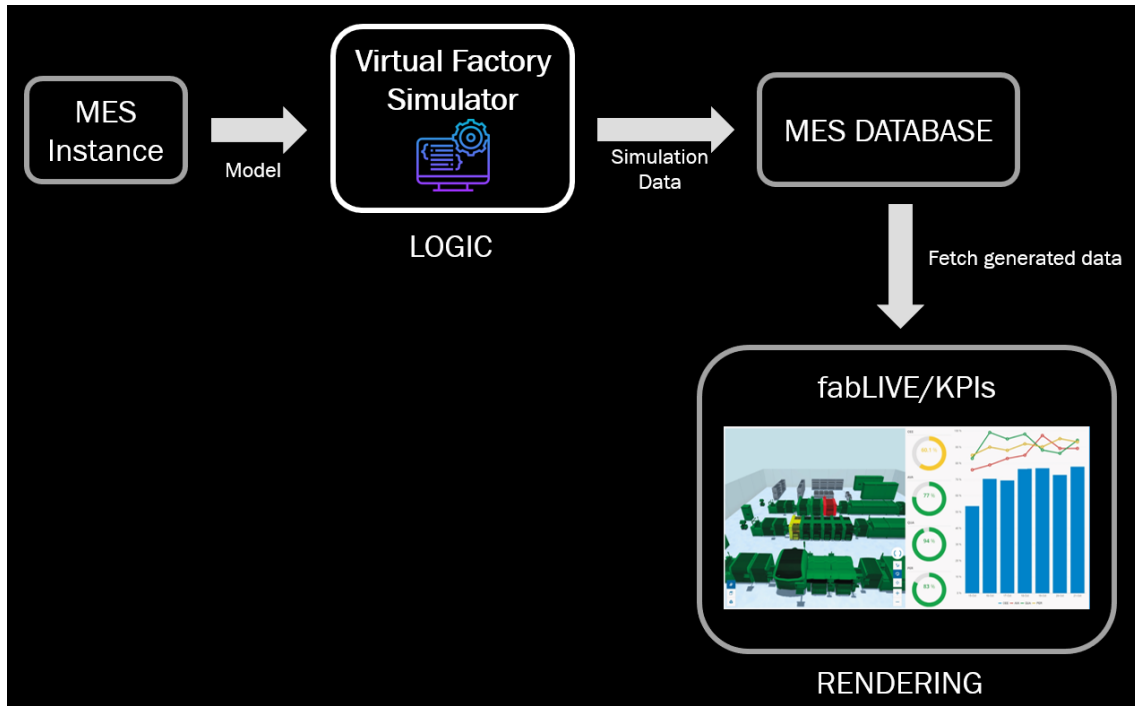


Figure 3.2: Data flow between MES and simulation software

Fig. 3.2 explains the data flow between the MES instance, the simulation software and the fabLIVE and KPIs modules. The simulator constructs a model based on the information fetched from MES and performs the simulation logic based on that same model. The simulation is executed through MES operations and, consequently, simulated data is generated and inserted into MES database. The MES' fabLIVE and KPIs modules will then render their visual components based on the information fetched from the database, which represents the current state of the virtual factory being simulated.

The dashboard containing a fabLIVE screen view and KPIs must be manually set by the user. The simulator will only generate the data to be used in KPI calculation and fabLIVE rendering.

3.3 MES Database Architecture

To discuss how the dates mismatch problem can be solved it is important to understand the MES database architecture first.

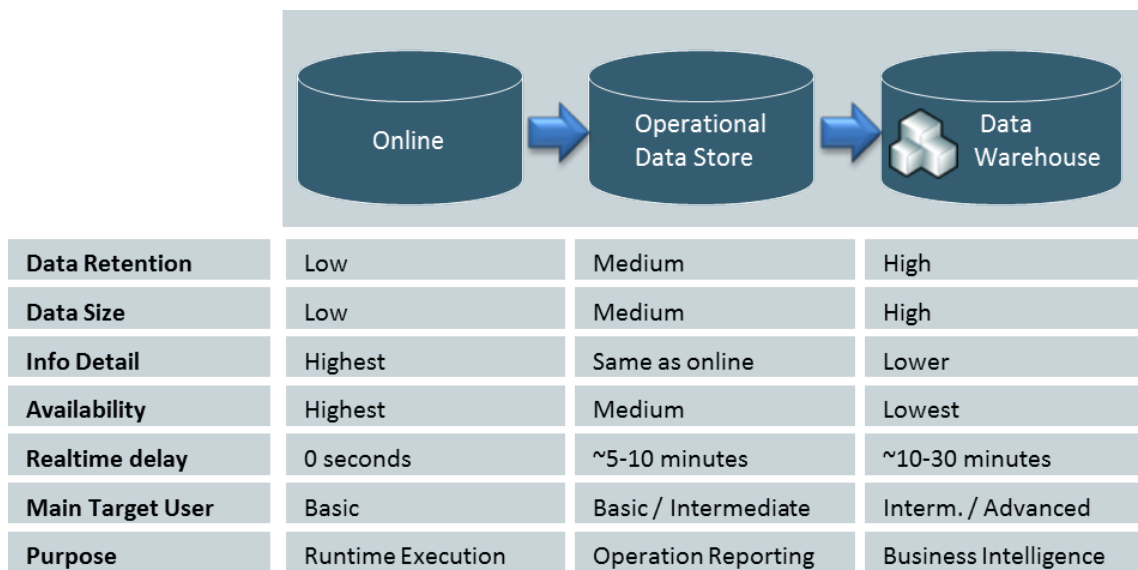


Figure 3.3: MES database architecture

Fig. 3.3 describes the MES database architecture. It consists of three sub-databases: Online, Operational Data Store (ODS) and Data Warehouse (DWH). A replication process exists to ensure data is cloned from one sub-database to another.

The online sub-database contains the runtime data up-to-date. Some minutes after the data has entered in the online sub-database it is replicated to the ODS sub-database.

The ODS sub-database contains all the online sub-database in a non-aggregated form and it is the primary source of reporting. A few times per day an aggregation job is executed, to aggregate the ODS data into the Data Warehouse.

The data used for KPI calculation is fetched from Operational Data Store or Data Warehouse, depending on the KPI type.

3.4 Proposed Solution

The first decision is to choose the programming language and technology for implementation. The simulation software will be using the MES API to communicate with the MES instance where the simulation is occurring. Therefore, the software should support the API programming language.

The MES API is implemented in both .NET and Javascript. Since the objective is to create a software implementing the simulation logic without requiring too much effort to be put on the interface development, an object-oriented approach is the best decision. Thus, a .NET implementation is the most justified.

The solution proposed consists in a simulation software whose architecture is based on the common architecture found on the reviewed literature in subsection 2.1.4 (Fig. 2.4).

An Input Reader component will be in charge of fetching the information from the MES instance and create a model based on that same information.

The discrete event simulator engine will be responsible for managing the current state data and decide which and in what order events must occur, through its event schedule module. The execution of an event always leads to an MES operation being invoked through the API. The engine must contain a probability distribution module to generate random numbers whenever required by an event.

The Output Generator component responsible for presenting the data in the correct format to later be used by the rendering system will not be present in the simulation software. Since the information display is carried out by the MES instance using its fabLIVE and KPI modules, the wisest decision is to use its data management system. Thus, the simulator is only required to invoke the MES operations through the API, and MES will automatically introduce the necessary data into the database to be used later on by fabLIVE and KPI display.

3.4.1 Dates Mismatch

To solve the dates mismatch problem (3.1.1) two main routes can be taken: modify the dates automatically generated by MES or maintain those generated dates and manipulate the KPIs' time frames instead.

Modify generated dates

As explained in section 3.3, the online sub-database contains the up-to-date information about the MES system. This data is then replicated to the Operational Data Store. Finally, that same data is aggregated into the Data Warehouse.

One solution to modify the automatically generated dates is to manipulate the replication script, so that the dates are automatically converted to the correct simulation date when the replication process from online to ODS takes place. When posterior aggregation job is performed to send the data from ODS to DWH, the data being aggregated will contain the rectified dates already.

Since the KPIs will be fetching the data from the ODS and DWH sub-databases, the data used for their calculation will already reference the correct simulation dates.

Modify KPI dates

KPIs are composed by time intervals declaring the first and last date time during which the measurements will take place. That being said, an alternative to the previous approach is to leave the automatically generated dates as they are and try to modify the date range used for the KPI calculation.

For example, imagine that the current simulation date is located in June 2021 and that is the month we want the KPI to target. However, the automatically generated dates contain the current system date, located in June 2020. According to this approach, instead of modifying the dates from June 2020 to June 2021 the objective is to find out what system date corresponds to the desired simulation date and use it as the time frame for the KPI calculation. In this case, the KPI

time interval would be of June 2020 but, in reality, it would be pertaining June 2021, the date in which the simulation is currently located.

Both presented solutions allow the creation of KPI entities that can reference simulation dates differing from the current system date. However, only the direct data modification approach allows the simulated data to be further used by presales staff for customer demonstrations containing the correct simulation dates. Therefore, the modification of the automatically generated dates will be the solution to implement.

3.5 Working Methodology

Critical Manufacturing's working methodology is based on agile development. Agile development is characterised by continuous iterations and testing that take place during the entire software development life cycle [29]. The agile methodology is less document-centric and more code-oriented, improving productivity in environments requiring adaptation to frequent changes [30]. This becomes a valuable methodology to *Critical Manufacturing* due to the several customized projects they have with other companies, and the constant changes in the manufacturing industry.

Several agile development methods exist. Scrum is the one adopted by *Critical*. To familiarize with the company policies the same methodology was used for the dissertation development. Many concepts are present in scrum but only the sprint idea was used throughout the dissertation.

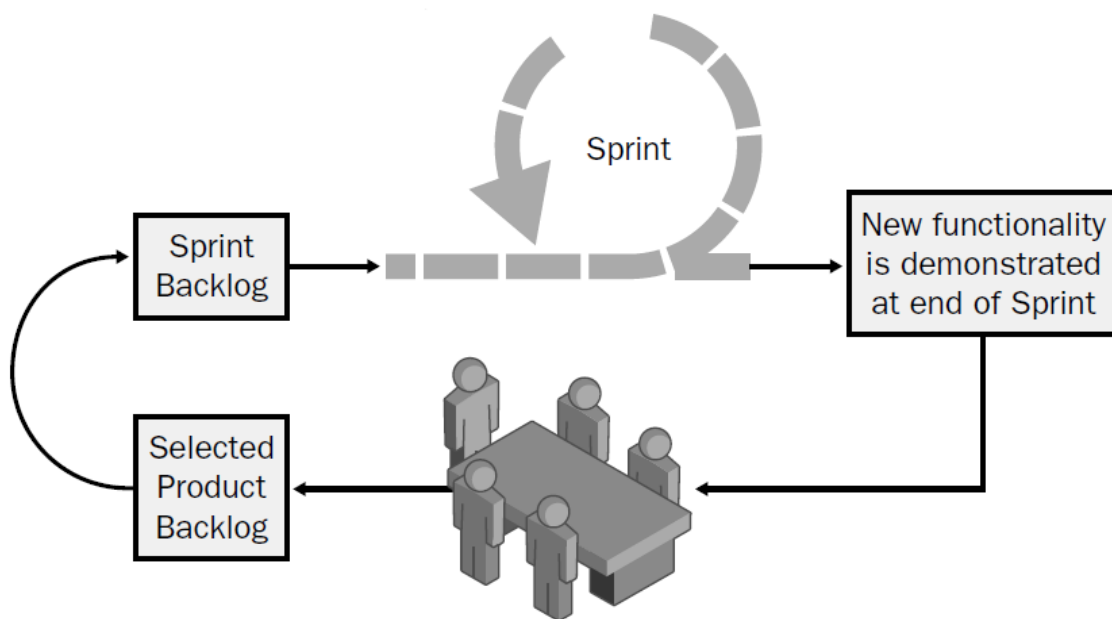


Figure 3.4: Scrum sprint process overview [4]

Scrum sprints are development phases that generally last from 1 to 3 weeks [29]. *Critical* implements 2 week sprints, so the dissertation development followed the same structure.

The product backlog is a list containing all requirements to be implemented. At the start of every sprint, a set of requirements is selected from the product backlog to be implemented for the next sprint, called sprint backlog [31]. At the end of every sprint, the implemented functionalities are presented and the unfulfilled requirements from that sprint are moved on to the next sprint backlog [4].

Chapter 4

Solution Implementation

This chapter presents all the important aspects of the solution implementation.

Section 4.1 defines some important vocabulary terms related to MES and the simulator. Sections 4.2 and 4.3 list all the implemented events and queries in detail. Section 4.4 explains the entity relations concept and how they can be obtained through the MES API.

Section 4.5 approaches the need for a multithreaded implementation and the mechanisms used to achieve it. Section 4.6 describes in detail all the implemented class and respective methods along with UML class diagrams. Section 4.7 explains how date manipulation was performed to solve the dates mismatch problem.

Finally, in section 4.8, the workplan applied during the dissertation development is presented.

4.1 Vocabulary

Before explaining the implemented solution, it is important to understand certain concepts related to MES actions and the implemented classes essential to the solution development besides the simulation terms already presented in subsection 2.1.1.

A dictionary with vocabulary terms related to implemented classes and MES actions can be found below.

*Facility (MES)

A facility corresponds to a physical shop floor location. A factory may contain several facilities.

For simplification purposes, the simulator assumes that only one facility exists.

*Step (MES)

A step represents a manufacturing process operation.

A step may contain several resources. Each one of those resources is used to provide a service to a material.

*Service (MES)

In a regular production line, a service must be provided to a material for it to move from the current step to the next one.

For a service to be provided to a material, a resource capable of providing such service is required in the current material's step.

*Flow (MES)

A flow represents a pre-defined processing route for a material.

A flow is a sequence of other flows or steps.

*Product (MES)

A product represents an abstract definition used to generate materials.

A material is an instantiation of a product.

Dispatch & Track In (MES)

A *dispatch* operation assigns a material to a given resource for later processing.

A *track in* operation starts the processing of a dispatched material at the assigned resource.

In MES, these are two separate actions. However, for simplification purposes, they are always performed atomically by the simulator.

Track Out & Move Next (MES)

A *track out* operation finishes the processing of a material that has been previously tracked in.

A *move next* operation places a material in its next step. If a material is currently in the last step of its flow, the *move next* operation will not be executed.

In MES, these are two separate actions. However, for simplification purposes, they are always performed atomically by the simulator, except when the material is in the last step. In that case, it will only be tracked out.

*Loss Reason (MES)

A loss reason is used to apply a certain loss quantity to a material when it is tracked out.

When a loss reason is applied, the material quantity is reduced.

Query (Simulator)

A query is an implemented simulator class used to fetch information from the MES instance through the MES API.

The simulation model is initially built thanks to the multiple queries that fetch several MES entities, such as facilities, products, flows, steps, etc.

Service Waiting Queue (Simulator)

A service waiting queue consists in a list of materials that require that same service and have been previously scheduled for a dispatch and track in. However, no resources were available to provide the service at that time. This might happen due to resources unavailability or limited capacity.

* Class implemented by the MES API.

4.2 Events

The simulator is driven by events and time advances based on their execution. An event in the implemented simulator always leads to an MES operation being executed through the API.

The events are managed by an implemented class called *EventSchedule* which stores and chronologically organizes all the events to be executed in the future.

All events must have an occurrence date associated, unless they are supposed to be executed as soon as possible. Also, an event requires input values used as context for its execution. An event input can be a material, product, resource, step, etc.

An event might have follow up operations related to event scheduling and simulation state update performed after the MES operation has been executed. Generally, an event execution leads to another event being scheduled which allows a continuous sequence of events to be automatically generated.

The simulator implements 4 different events. Their descriptions, inputs and follow up operations are listed below.

Dispatch & Track In

This event performs the *dispatch* and *track in* MES operations atomically.

Before the MES operations are executed it is necessary to select, in runtime, the resource to be assigned to the material being dispatched and tracked in. The selection is made from all the resources present at the material step capable of providing the required service.

If no resource is available at the material's step to provide the required service, the material will be added to the service waiting queue.

The *dispatch & track in* event has no occurrence date. It should be performed as soon as possible.

Inputs:

- Material – Material to be dispatched and tracked in.

Follow up operations:

- Update simulation state of resource involved in the event;
- Schedule *track out & move next* event for the material/resource pair used in this event.

Track Out & Move Next

This event performs the *track out* and *move next* MES operations atomically, except if the material is already in its final step, in which case only the *track out* operation is executed.

Before the MES operations are executed it is necessary to generate the loss quantity to be applied based on a random number generator, if there is a loss reason associated to the material's step.

The *track out & move next* event has an occurrence date calculated based on the resource processing time.

Inputs:

- Material – Material to be tracked out and moved to next step;
- Resource – Resource where the material has been tracked in.

Follow up operations:

- Update simulation state of resource involved in the event;
- Make resource unavailable, if decided by a random number generator method. A resource is made unavailable by invoking the *state change* MES operation to set the resource state to *down*;
- If resource will continue available then schedule a *dispatch & track in* event for materials at the resource's service waiting queue;
- Schedule *dispatch & track in* event for the material used in this event;
- If material is in the first step of the flow then check if a material restock is required at that step;
- If resource became unavailable then schedule a *resource repair* event for that same resource.

Material Restock

This event performs the *create material* MES operation.

The material characteristics are based on the product definition. Every product contains a maximum quantity and a minimum quantity that shall be respected when a material is created. It is possible for any of this values to be undefined.

This event will create as many materials as necessary to guarantee the minimum material threshold is met for a given product at a given step. A material restock is always performed at a flow's first step.

The *material restock* event has no occurrence date. It should be performed as soon as possible.

Inputs:

- Product – Product to be restocked;
- Step – Step where the restock necessity will be evaluated.

Follow up operations:

- Schedule *dispatch & track in* event for every created material.

Resource Repair

This event performs the *change state* MES operation to set a resource's state to *up*. This way, a resource will be available for material processing.

The *resource repair* event has an occurrence date calculated based on the resource repairing time.

Inputs:

- Resource – Unavailable resource to be repaired.

Follow up operations:

- Update simulation state of the repaired resource;
- Schedule a *dispatch & track in* event for materials at the resource's service waiting queue;

4.3 Queries

The simulator uses queries to fetch information from the MES instance regarding simulation entities.

Queries are executed at the simulation loading phase to fetch all the entities to be used throughout the simulation, and the initial factory model is created based on that fetched information.

Queries also serve the purpose of updating a certain entity's state during simulation execution. This usually happens when the new entity's state cannot be derived by the simulation logic and requires to be fetched directly from the MES instance.

All queries return a list with entities implemented by MES API classes.

Some of the queries search for specific entities and require an input used to search them.

The simulator implements 10 different queries. Their descriptions and inputs are listed below.

*Get All Active Loss Reasons

Returns all active loss reasons.

*Get All Active Materials

Returns all active materials.

*Get All Effective Flows

Returns all effective loss reasons.

Flows require an extra procedure to be used. They become active when created, and require to be made effective to be used in a MES production activity, unlike most MES entities. Therefore, this query will fetch for effective flows instead of active ones.

*Get All Effective Products

Returns all effective products.

Products go through the same logic as flows, that being the reason why effective products are fetched instead of active ones.

*Get All Resources

Returns all resources.

*Get Facility

Returns a facility by its ID.

Inputs:

- ID – ID number identifying the facility to be fetched.

Get Material

Returns a material by its ID.

Inputs:

- ID – ID number identifying the material to be fetched.

*Get Materials At Step

Returns all active materials located a step. The step is identified by its ID.

Inputs:

- ID – ID number identifying the step whose active materials will be fetched.

Get Resource

Returns a resource by its ID.

Inputs:

- ID – ID number identifying the resource to be fetched.

*Get Resources Processing Time

Returns all resources' processing time.

* Queries only used at simulation loading phase because no more entities of that type will be created during the simulation. More entities of that type can be created if the simulator purposely creates them due to the simulation logic (e.g., materials). In such case, the simulator can access the entity without executing the highlighted queries.

4.4 Entity Relations

An MES entity can be associated with other entities. For example, a resource can provide several services. In MES, this is seen as a relation between the resource and the services.

The MES API provides a way to load all the relations of a given entity. An entity's relations can be accessed through the *RelationCollection* attribute. This attribute represents a dictionary mapping the relation name (e.g., *ResourceService*, *FlowStep*, etc.) into a list containing the relation objects.

This facilitates the access to entities related to another entity without requiring a query execution. For example, when we want to access all the steps belonging to a flow, it is as simple as invoking, for a flow object, the API function responsible for loading an entity's relations, and then access the *RelationCollection* value mapped by the "FlowStep" key. There's no need to create a new query to specifically fetch all the steps containing a certain flow id.

The relation loading operation has three levels, ranging from 0 to 2. Higher loading levels increase the relation entities' information. For example, when loading the steps of a certain flow with level 0, the step objects obtained from the "FlowStep" relation will only contain basic information, such as, name and ID. The loading level must be increased for more detailed information. To receive full information, the higher level (2) should be selected.

4.5 Locks & Multithreading

During simulation multiple events can be executing at the same time. Some can stand idle while waiting for their occurrence date to arrive. This becomes difficult to implement by using only one processing thread during the entire simulation as the waiting process of some events can block the main thread's execution. For this reason, a multithreaded approach was implemented.

The need for several threads running simultaneously while processing different events may create deadlocks and inconsistency issues when two separate threads concurrently access and manipulate the same information.

.NET implements a lock mechanism limiting the information access to only one thread at a time. This will cause some overhead, since a thread may need to wait for another to finish its access. However, this procedure is important to guarantee the information stays consistent and uncorrupted.

The software implements a lock object for every variable where concurrent access is possible. This implies creating a lock object for entities and data structures shared across multiple threads, e.g., resource entities and event queue.

4.6 Classes & Methods

Below, a detailed description of all the implemented classes that constitute the simulation software, along with their methods and UML class diagrams, can be found.

4.6.1 Simulator

The simulator class represents the central piece for the simulation process and acts as an interface for the simulation actions, namely, simulation start and acceleration change. It is responsible for creating the events and ordering their execution.

Attributes:

SimulationStatus

Indicates the current simulation execution status: Finished, Running or Preparing. Preparing means the shop floor information is still being loaded from MES and the factory model is being created.

EventsExecuting

List containing all the events that are currently executing.

Clock

Simulation clock used to control time.

EntityManager

Stores and manages the simulation state.

EventSchedule

Stores and organizes the events to be executed.

EventLock

Lock object used to access the EventsExecuting list.

Methods:

void CreateInitialEvents()

Creates the first simulation events, namely, material track ins and material restocks.

void CreateTrackInEvent(Material)

Schedules a dispatch & track in event for a given material.

void CreateTrackOutEvent(Material, Resource)

Schedules a track out & move next event for a given material and resource.

void CreateRepairEvent(Resource)

Schedules a resource repair event for a given resource.

void CheckForStepRestock(Step, Product)

Evaluates the accumulated quantity of all materials of a given product at a given step. A material restock event is scheduled if the quantity is below a certain threshold.

void EventFinished(EventId)

This method is used to signal the simulator that an event has finished executing.

Removes the event with the given id from the list tracking the currently executing events. It also verifies if the simulation has finished.

void StartSimulation(StartingSimulationTime)

Starts the simulation execution. Initiates the simulation clock with the given date expressed in simulation time.

void ChangeAcceleration(NewAcceleration)

Changes the current simulation acceleration to the given one.

4.6.2 EntityManager

The entity manager class stores and manages the current state of all entities used throughout the simulation. The initial entities' state is fetched from MES at the simulation loading phase performed before the simulation start.

Attributes:

Facilities

List containing all the facilities fetched from MES. Although all facilities are fetched, only the first one on this list is used during simulation execution. This happens for simplification purposes, as stated previously.

Flows

List containing all the effective flows fetched from MES.

Materials

List containing all the active materials fetched from MES.

Products

List containing all the effective products fetched from MES.

LossReasons

List containing all the active loss reasons fetched from MES.

ServiceResources

Dictionary associating, to each service, all the resources who provide it.

ServiceQueues

Dictionary associating, to each service, the materials on its waiting queue.

ResourcesProcessTime

Dictionary associating, to each resource, its processing time.

ResourcesDownTime

Dictionary associating, to each resource, its repairing time.

ResourceEventLocks

Dictionary storing one lock object for every resource. The locks are used by events requiring a certain resource for execution.

ProductRestockLocks

Dictionary storing one lock object for every product. The locks are used when restocking a product.

ResourceAccessLock

Lock object used when accessing the ServiceResources dictionary.

ServiceQueuesLock

Lock object used when accessing the ServiceQueues dictionary.

Methods:

void LoadEntities()

Fetches all the entities from MES using the query classes implementing the IQuery interface. Initiates class attributes with the fetched information.

List<Material> GetQueuedMaterials()

MES materials have several states inside the MES instance. Queued state indicates that a material is in a step waiting for a dispatch & track in to happen. Thus, this method returns all the materials ready to be dispatched and tracked in.

List<Reason> GetStepLossReasons(Step)

Returns all the loss reasons affecting a given step.

Resource GetResourceForService(Service)

Returns a resource available to provide the given service.

If several resources are available, returns the one currently processing less materials. Returns null if no resources are available.

double GetResourceProcessTime(Resource)

Returns the processing time for a given resource, in seconds. The value is randomly generated based on a normal distribution.

double GetResourceDownTime(Resource)

Returns the repairing time for a given resource, in seconds. The value is randomly generated based on a normal distribution.

Resource GetResource(ResourceName)

Returns the resource entity with the given name.

object GetResourceLock(Resource)

Returns a given resource's lock.

object GetProductRestockLock(Product)

Returns a given product's lock. The lock will be used for material restock regarding the given product.

Step GetFlowFirstStep(Flow)

Returns the first step of a given flow.

bool IsMaterialInFinalStep(Material)

Returns true if a material is located at the last step of its flow. Returns false otherwise.

bool IsMaterialInFirstStep(Material)

Returns true if a material is located at the first step of its flow. Returns false otherwise.

decimal GetStepStock(Step, Product)

Returns the material stock for a given product at a given step.

void AddMaterialToQueue(Material)

Adds a material to its required service waiting queue.

List<Material> GetAndEmptyServiceQueue(Service)

Returns a list with all materials from a given service waiting queue and empties that same queue.

void UpdateResource(UpdatedResource)

Updates the state of resource that corresponds to given updated resource.

4.6.3 EventSchedule

The event schedule class stores all the events to be executed and organizes them by occurrence date.

Attributes:

EventQueue

List containing all the events to be executed, organized in ascending order by occurrence date.

EventQueueLock

Lock object used when accessing the EventQueue list.

Methods:

void ScheduleEvent(NewEvent)

Adds a new event to the event queue and reorganizes the queue in the correct order.

Event GetNextEvent()

Returns and removes the next event from the queue.

bool IsEmpty()

Returns true if the event queue is empty. Returns false otherwise.

4.6.4 ProbabilityDistribution

The probability distribution class is responsible for generating random numbers based on several probability distributions. This class is static.

Methods:

double GenerateNormal(Mean, Variability)

Generates a random number based on a normal distribution with a given mean and variability. The range of generated values goes from [mean-variability, mean+variability].

The generation method uses the Box-Muller Transform which converts two random numbers generated through a uniform distribution into a single number as if it was generated through a normal distribution.

The returned number is a double.

int GenerateNormalInteger(Mean, Variability)

Applies the same logic as the previous number but the return value is an integer instead.

bool GenerateBinary(TrueProbability)

Returns a true or false value based on a Bernoulli distribution.

$$P(\text{true}) = \text{TrueProbability}$$

$$P(\text{false}) = 1 - \text{TrueProbability}$$

4.6.5 Clock

The clock class stores previous acceleration changes and performs time conversion from real date to simulation date and vice-versa based on previous acceleration records.

Attributes:

StartingRealTime

Stores the simulation start date expressed in real time.

StartingSimulTime

Stores the simulation start date expressed in simulation time. The simulator can be initiated with a simulation date different from the real date.

AccelerationChangeRecord

List containing all the previous acceleration changes.

Methods:

DateTime ConvertToSimulationTime(RealTime)

Converts a given real date to its corresponding simulation date.

DateTime ConvertToRealTime(SimulTime)

Converts a given simulation date to its corresponding real date.

void Start(InitialAcceleration)

Creates the first acceleration record with the given acceleration and initiates the StartingRealTime attribute with the current time. This method is invoked upon simulation start.

void RegisterNewAcceleration(int newAcceleration)

Register an acceleration change. This method is invoked when the simulator class changes its acceleration.

4.6.6 AccelerationChange

The acceleration change class represents an acceleration change record to be stored on clock class.

Attributes:

Acceleration

Acceleration of this record.

RealTime

Real time date of the acceleration change occurrence.

SimulTime

Simulation time date of the acceleration change occurrence.

4.6.7 Event

The event class represents a simulation event that will eventually be executed and result in a MES operation. This class is abstract and is derived by 4 different instantiable classes, each one representing one of the four events previously explained in section 4.2. All the event derived classes must implement the *ExecuteSpecific()* method with the algorithm responsible for executing that specific event.

Each event is identified by a unique ID number.

Attributes:

IDCounter

Counter used to generate new event IDs.

ID

ID number identifying this event.

Simulator

Simulator object responsible for controlling the simulation.

InstantOccurrence

Boolean indicating if this event should be executed as soon as possible. If false, then an occurrence date should be provided.

OccurrenceDate

Event occurrence date expressed in simulation time.

PossibleLastEvent

Boolean indicating if there's a possibility of this event being the last one of the simulation. Later used to verify if the simulation has finished.

E.g., a simulation can finish with a *track out & move next* event, but not with a *dispatch & track in*.

Methods:

void Execute()

Starts the event execution.

This method implements the Template Method design pattern by invoking the *ExecuteSpecific()* method implemented by the derived classes which starts the real event execution, and then executes the Simulator's *EventFinished()* method.

abstract void ExecuteSpecific()

Contains the event execution algorithm. This is an abstract method implemented by the derived class. This method implements the Strategy design pattern since every event derived class has its own and unique execution algorithm.

void TryToTrackInWaitingQueue(Resource)

Schedules a *dispatch & track in* event for all materials in the waiting queue of the given resource's services.

void WaitForOccurrenceDate()

Waits until the current simulation date reaches the event's occurrence date.

CompareTo(Event other)

Compares two events by their occurrence date.

This method is used by the Event Scheduler class to organize the event queue.

string GetServiceComment()

Generates a JSON string containing the simulation info to be used in MES service notes for later database date modification.

4.6.8 Query

The query class is an interface to be implemented by all the queries previously explained in section 4.3. It is also a C# generic class because different queries can fetch different entities and, therefore, will return different class types.

Only MES API classes are used as the generic type by the implemented queries.

Methods:

List<T> Execute()

Executes the specific query implementing the interface and returns a list with the entities returned by the query.

This method implements the Strategy design patten, since every query uses its own execution algorithm.

4.6.9 Utils

The utils class is a static class used for utility methods which don't properly fit in any other class and can be required by multiple classes.

Methods:

double ConvertToSeconds(TimeLength, TimeUnit)

Converts, into seconds, a given time length expressed in a given time unit, namely, days, hours or minutes.

List<T> LoadRelations<T>(EntityList, LevelsToLoad)

Loads the relations of all entities in a given list with a given loading level.

Returns the same entity list received as parameter with the entities' relations loaded.

List<T> LoadRelations<T>(Entity, LevelsToLoad)

Loads the relations of a given entity with a given loading level.

Returns the same entity received as parameter with the entity's relations loaded.

4.6.10 Class Diagrams

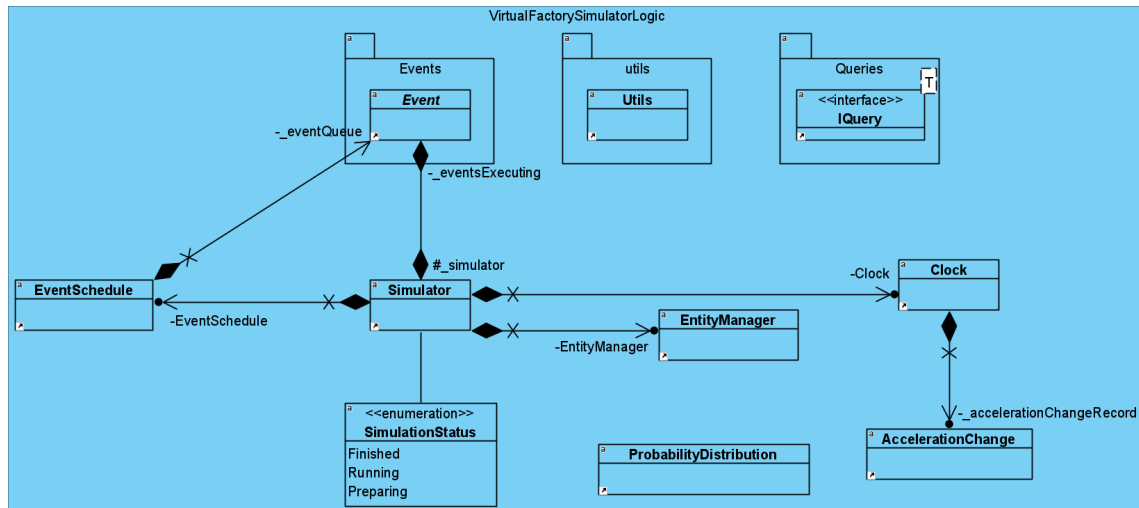


Figure 4.1: UML class diagram covering the entire software implementation

Fig. 4.1 presents the UML class diagram of the entire architecture. No derived event classes or queries are shown for image simplicity.

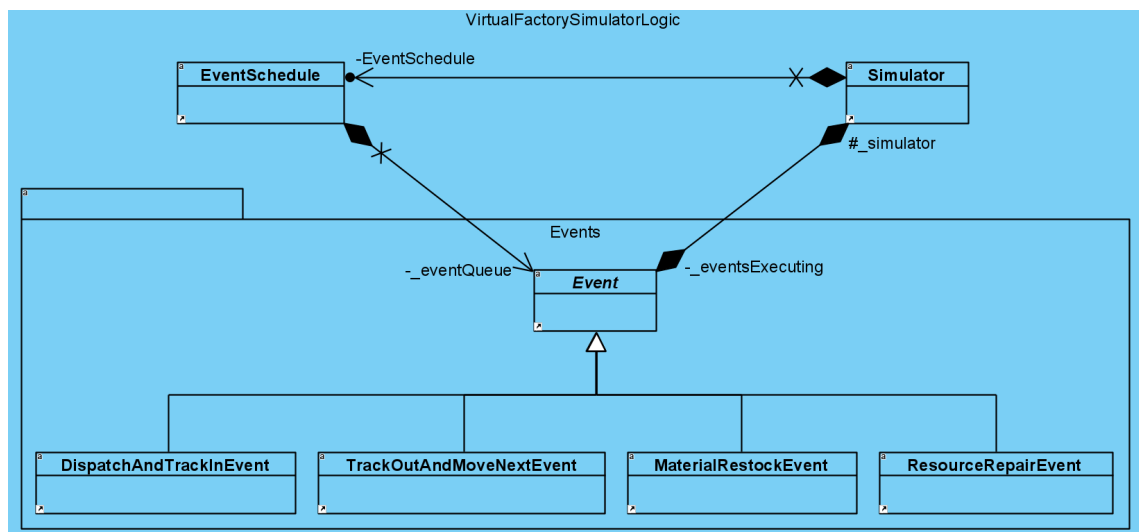


Figure 4.2: UML class diagram covering the events package and its associates

Fig. 4.2 presents the UML class diagram for the events package and its associates, i.e. EventSchedule and Simulator classes. The derived event classes can be seen.

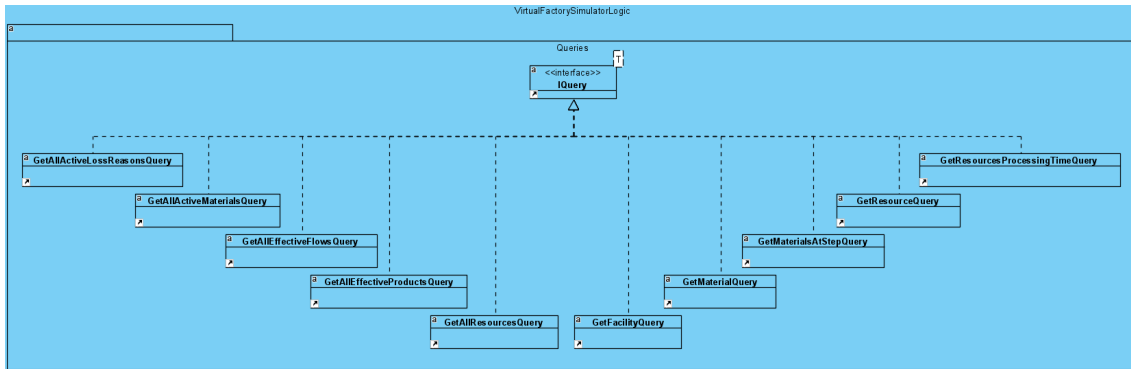


Figure 4.3: UML class diagram covering the queries package

Fig. 4.3 presents the UML class diagram for the queries package. All implemented queries can be seen.

4.7 Database Replication & Date Modification

To solve the dates mismatch problem it is necessary to understand the replication process from Online to ODS sub-databases performed by MES database.

4.7.1 ReplicationTables

A table called *ReplicationTables*, located at the Online sub-database, stores, for each table to be replicated, a T-SQL script to be executed when replication is performed. These T-SQL scripts are constituted by insert and merge statements that fetch the information from the Online table to be replicated, and insert it into the respective table in the ODS sub-database.

ReplicationTables contains four important attributes: *SchemaName*, *TableName*, *InsertScriptForReplication* and *ParametersForReplication*.

SchemaName and *TableName* are used to identify the table to be replicated.

InsertScriptForReplication contains the T-SQL script to execute the replication of the respective table.

ParametersForReplication is an empty column not used by MES where useful information can be stored to aid in date modification during the replication process.

4.7.2 Stored Procedures

There are two stored procedures responsible for replicating the data from Online to ODS: *DataReplication* and *DataSyncGenerator*.

The *DataReplication* procedure is responsible for iterating through all the replication scripts stored in *ReplicationTables* and executing them so that the data is duplicated from Online to ODS.

The *DataSyncGenerator* is responsible for generating the replication scripts stored on *ReplicationTables*. This procedure invokes another procedure called *GetCommaSeperatedColumnString*.

GetCommaSeperatedColumnString is responsible for generating the string used to select the columns in the replication T-SQL script.

For example, assume *GetCommaSeperatedColumnString* generates string *S*. Considering the replication script as an insert on table *T*. The final replication script generated would be similar to:

```
INSERT INTO ods.T (S)
SELECT S FROM online.T
```

Since *S* represents the list of columns to be fetched, the objective is to manipulate this generated *S* string so that a mathematical operation is applied to the date attributes to convert them into the corresponding simulation date.

4.7.3 Date Attributes

Now that the replication scripts generation is well known, it is necessary to understand how the *GetCommaSeperatedColumnString* procedure gains access to a table's attributes to generate the string used in the replication script.

The information related to a table's columns is stored in a view called *ColumnInformation*. This view contains 3 important attributes: *TableName*, *ColumnName* and *DataType*.

TableName contains the name of the table with its schema name included.

ColumnName contains the name of the columns belonging to a table.

DataType indicate the column variable type. E.g., bigint, varchar, datetime, etc.

Since the objective is to manipulate dates, the columns to be manipulated by *GetCommaSeperatedColumnString* must be of datetime type. However, not all date attributes are to be modified. There's certain tables like the ones related to MES calendar where a date modification could make the software defective.

The *CreatedOn* and *ModifiedOn* date attributes are shared by many tables and can always be modified. If any other datetime attribute requires modification, it should be included in the *ParametersForReplication* column on *ReplicationTables* table.

4.7.4 Date Modification Mathematical Operations

Information regarding the simulation execution is required to convert a real date to its corresponding simulation date. When converting real dates, it is necessary to have access to the acceleration (Accel) applied at the given real date, as well as the real date (AccelSetRealDate) and simulation date (AccelSetSimulDate) from when that acceleration was set.

With these three variables, the converted simulation date can be calculated as follow:

$$\text{ConvertedSimulDate} = \text{AccelSetSimulDate} + (\text{OriginalRealDate} - \text{AccelSetRealDate}) * \text{Accel}$$

4.7.5 Service History

Besides knowing the mathematical expression to convert real dates into simulation dates, it is also necessary to send the required simulation information to the database to later be used in the mathematical operations by the replication scripts.

This is done through MES services. All MES actions performed by events are seen by MES as services. Whenever a service is executed, a new row is added to the *ServiceHistory* table.

Luckily, services contain an attribute accepting a modifiable string called Notes. For the replication scripts to have access to the required variables, the simulation info must be introduced into a service's notes when a MES operation is executed by an event. The date should be introduced in JSON format to facilitate the information access at the database.

Since the simulation info will be stored at *ServiceHistory* table, the replication scripts need to reference this table when selecting the columns for replication.

The selected names for the service notes' JSON attributes were *Acceleration*, *StartRealDate*, *StartSimulDate*. Here's an example of service notes content:

```
{ "Acceleration": 4, "StartRealDate": "2020-06-17T01:24:26", "StartSimulDate": "2021-06-17T04:24:26" }
```

Notice that the string dates must be in "yyyy-MM-ddThh:mm:ss" format.

4.7.6 Date Modification

Now that all the necessary information required for date modification is known, let's analyze changes applied to the code.

A modification was made to the *GetCommaSeperatedColumnString* procedure. Instead of simply adding a column's name to the output string variable, the following T-SQL statement is executed to generate the string to be added to the output string variable:

```
1 case
2   when DataType = 'datetime' and
3     (ColumnName = 'CreatedOn' or
4      ColumnName = 'ModifiedOn' or
5      @p_ColumnsToConvertDate LIKE '%,' + ColumnName or
6      @p_ColumnsToConvertDate LIKE ColumnName + ',%' or
7      @p_ColumnsToConvertDate LIKE '%,' + ColumnName + ',%')
8   then 'case
9     when ISJSON(Srvchstry.Notes) = 1
10      and ''Acceleration'' in (select [key] from openjson(Srvchstry.Notes))
11      and ''StartRealDate'' in (select [key] from openjson(Srvchstry.Notes))
12      and ''StartSimulDate'' in (select [key] from openjson(Srvchstry.Notes))
13      then dateadd(SECOND, datediff(SECOND, CONVERT(datetime, JSON_VALUE(Srvchstry.
        Notes, '$.StartRealDate'))), '+@SourceNomenclature+'.['+ColumnName+']) *
        CONVERT(decimal, JSON_VALUE(Srvchstry.Notes, '$.Acceleration')),
        CONVERT(datetime, JSON_VALUE(Srvchstry.Notes, '$.StartSimulDate')))
```

```

14     else '@SourceNomenclature+' + ColumnName + ']'
15     end as [' + ColumnName + ']'
16     else @SourceNomenclature+'[' + ColumnName + ']'
17 end

```

Lines 1 to 7 specify when the date should be modified. It is mandatory for the column `DataType` to be `datetime`. Additionally, the column name must be *CreatedOn*, *ModifiedOn* or contained in *@p_ColumnsToConvertDate* procedure parameter. The *@p_ColumnsToConvertDate* parameter will receive the *ParametersForReplication* attribute from *ReplicationTables*.

If the column should not be modified (line 16), the procedure simply returns the column name preceded by the nomenclature given to the table at the table's replication script.

Otherwise (lines 8 to 15), the column name is replaced by the arithmetic operation presented in subsection 4.7.4 implemented in T-SQL. The arithmetic operation is only applied if the service history row associated to the row being replicated contains the notes in the correct JSON format. If not, just produce the same result as line 16.

Previously, if a column contained columns A, B and C, and C was a date to be modified, the *GetCommaSeperatedColumnString* procedure would return the following string:

```
[column A], [column B], [column C]
```

Supposing the table being replicated received "SRC" nomenclature, after the script modification the returned string will be the following:

```

column A, column B,
case
  when ISJSON(SrvcHstry.Notes) = 1
  and ''Acceleration'' in (select [key] from openjson(SrvcHstry.Notes))
  and ''StartRealDate'' in (select [key] from openjson(SrvcHstry.Notes))
  and ''StartSimulDate'' in (select [key] from openjson(SrvcHstry.Notes))
  then dateadd(SECOND, datediff(SECOND, CONVERT(datetime, JSON_VALUE(SrvcHstry.
    Notes, '$.StartRealDate')), SRC.[column C] * CONVERT(decimal, JSON_VALUE(
    SrvcHstry.Notes, '$.Acceleration')), CONVERT(datetime, JSON_VALUE(SrvcHstry
    .Notes, '$.StartSimulDate')))
  else SRC.[column C]
end as [column C]

```

As a final step, the service history table needs to be included in the replication script, since it is being referenced in the string generated by the *GetCommaSeperatedColumnString* procedure as "SrvcHstry".

Assuming *GetCommaSeperatedColumnString* generates string *S*. Considering the replication script as an insert on table *T*. Here's a simplified prototype to explain how the final replication script will look like:

```
INSERT INTO ods.T (S)
SELECT S
FROM online.T SRC LEFT JOIN online.ServiceHistory SrvcHstry
ON (SrvcHstry.ServiceHistoryId = SRC.ServiceHistoryReferenceColumn)
```

4.8 Executed Workplan

The executed workplan followed a timeline mainly constituted by two week sprints, from February 10th to June 26th. The tasks performed at each stage are listed below.

Feb 10th – Feb 26th (2.5 weeks)

- Familiarize with MES tool and concepts;
- Discuss software requirements.

Feb 26th – Mar 4th [Sprint 1] (1 week)

- Discuss software requirements;
- Create basic simulator interface;
- Progress on obtaining MES entities through API, for simulation model creation purposes.

Mar 4th – Mar 18th [Sprint 2] (2 weeks)

- Finalize simulation model creation and MES entities fetch through API;
- Implement generation and execution of a simple sequence of *dispatch & track in* and *track out & move next* events.

Mar 18th – Apr 1st [Sprint 3] (2 weeks)

- Implement automatic simulation execution by pressing play button (previous sprint required one play button click for every event to execute);
- Implement material restock;
- Implement variability in resources' processing time according to normal distribution.

Apr 1st – Apr 15th [Sprint 4] (2 weeks)

- Implement simultaneous material processing at same resource;
- Implement material losses with variable amount according to normal distribution;
- Support multiple resources providing the same service at the same step;
- Implement resource unavailability;
- Implement service waiting queues.

Apr 15th – Apr 29th [Sprint 5] (2 weeks)

- Implement time acceleration;
- Study possible solutions for dates mismatch problem.

Apr 29th – May 13th [Sprint 6] (2 weeks)

- Fix material restock bug;
- Study Online -> ODS data replication process.

May 13th – May 27th [Sprint 7] (2 weeks)

- Progress on Online -> ODS replication procedure modification for date manipulation.

May 27th – June 26th [Sprint 8] (1 month due to holidays)

- Finalize Online -> ODS replication procedure modification for date manipulation.

Chapter 5

Validation Tests

This chapter presents the testing scenarios created to validate the simulator implementation. In section 5.1, the testing methodology used is briefly explained. Section 5.2 describes the case studies used to validate the implemented features at each sprint.

5.1 Testing Methodology

The testing methodology follows the Scrum ideology. Every sprint, a testing scenario was used to validate the implemented simulation functionalities. As more sprints went by, the testing scenarios increased in complexity.

The performed tests' setup and results validation was performed manually through the MES instance interface.

All tests consist in a factory case study environment constituted by a group of MES entities, where simulation is performed. The entities' state is manually tracked during the simulation and the test succeeds if the obtained state matches the expected.

5.2 Testing Scenarios

A testing scenario was created for each sprint, except for sprint 1 where no testable functionality was implemented. All test scenarios derive from the same case study used as a basis.

At each sprint, new MES entities were added to the previous case study environment to increase the test complexity and validate the new implemented functionalities.

5.2.1 Case Study Basis

The initial case study environment used as basis for the sprint tests creates a minimalist, yet realist, manufacturing scenario to test the main simulation aspect: the execution of an automatically generated *dispatch & track in* and *track out & move next* event sequence.

Table 5.1: Basis case study entities

Entity	Amount	Names
Facility	1	F1
Flow	1	DemoFlow
Step	5	DemoStep1, DemoStep2, DemoStep3, DemoStep4, DemoStep5
Resource	5	DemoResource1, DemoResource2, DemoResource3, DemoResource4, DemoResource5
Service	5	DemoService1, DemoService2, DemoService3, DemoService4, DemoService5
Product	1	DemoProduct
Material	1	DemoMaterial
Loss Reason	0	–

Table 5.2: Basis case study variables and values

Variable	Mean	Variability
Resource Processing Time	5s	–
Resource Repairing Time	–	–
Resource Breakdown Probability	–	–
Loss Quantity	–	–
Restock Threshold	–	–
Quantity Per Material	100	–

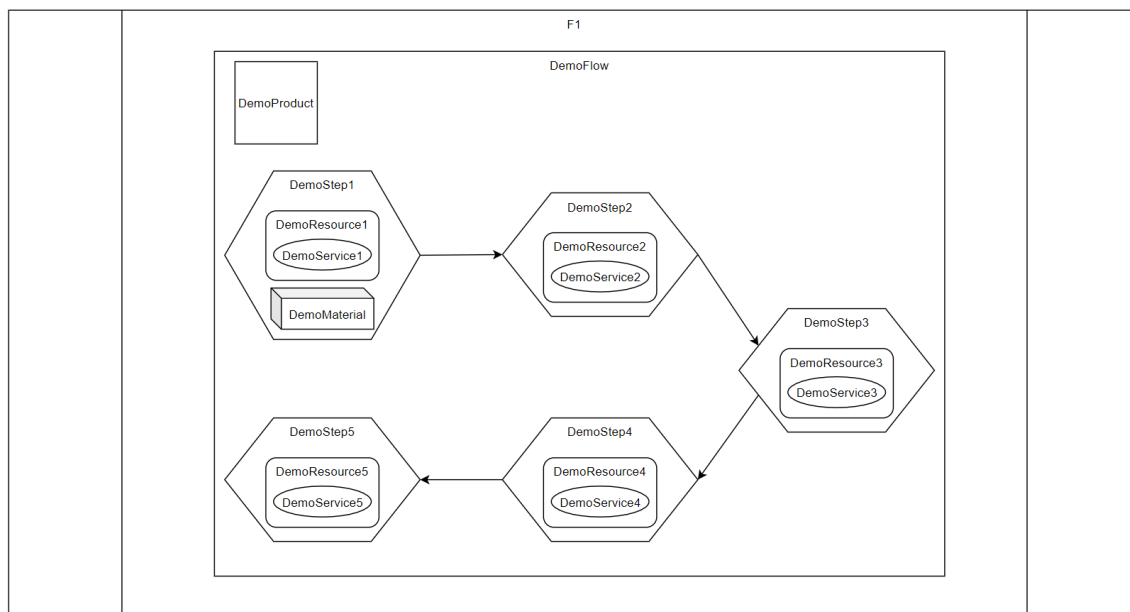


Figure 5.1: Diagram explaining the manufacturing setup used as the basis case study

Table 5.1 lists the entities used in the basis case study. Table 5.2 lists variable values, related to entities' properties, used by simulation events.

At this case study, each resource provides one service and each step contains one resource. The association between steps, resources and services is dictated by the entity number. Resources have an unlimited processing queue. A diagram of the case study's manufacturing setup is presented in Fig. 5.1.

5.2.2 Sprint 2 - Case Study and Results

In sprint 2, the generation and execution of a simple sequence of *dispatch & track in* and *track out & move next* events was implemented. The tests used the basis case study to validate functionality.

A typical material processing was observed during the entire flow, while respecting the established resource processing time.

The only case study material registered a processing duration of 5 seconds at all resources and required 25 seconds to finish its production. The observed results validated the implemented functionality.

5.2.3 Sprint 3 - Case Study and Results

In sprint 3, material restock and variability in resources' processing time was implemented, which required a change to the basis case study.

Table 5.3: Sprint 3 case study entities

Entity	Amount	Names
Facility	1	F1
Flow	1	DemoFlow
Step	5	DemoStep1, DemoStep2, DemoStep3, DemoStep4, DemoStep5
Resource	5	DemoResource1, DemoResource2, DemoResource3, DemoResource4, DemoResource5
Service	5	DemoService1, DemoService2, DemoService3, DemoService4, DemoService5
Product	1	DemoProduct
Material	0	–
Loss Reason	0	–

Table 5.4: Sprint 3 case study variables and values

Variable	Mean	Variability
Resource Processing Time	30s	10s
Resource Repairing Time	–	–
Resource Breakdown Probability	–	–
Loss Quantity	–	–
Restock Threshold	100	–
Quantity Per Material	100	–

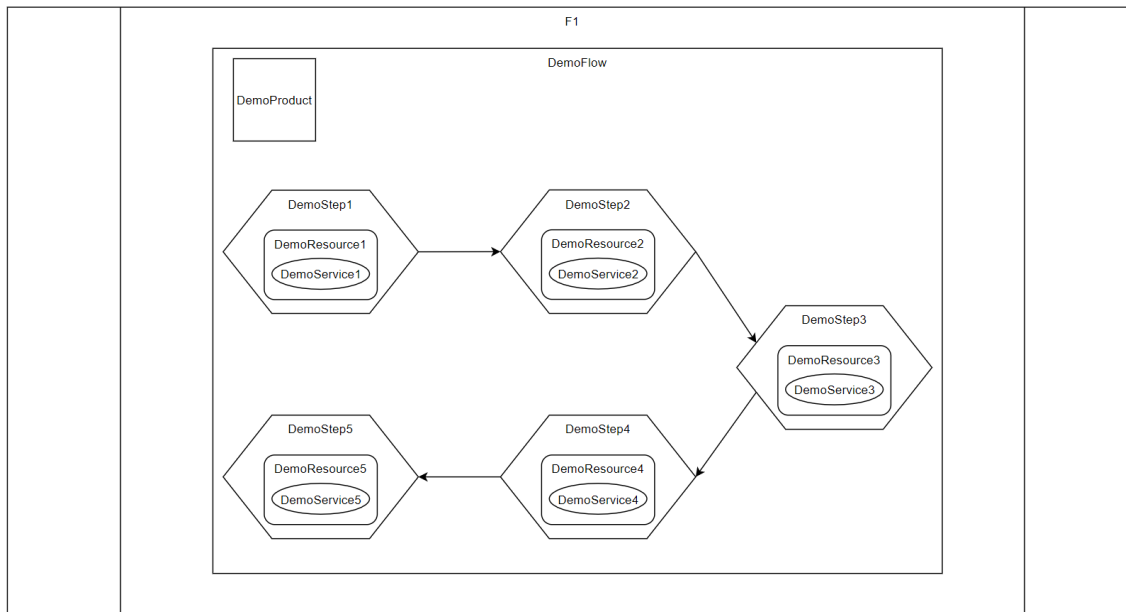


Figure 5.2: Diagram explaining the manufacturing setup used as sprint 3 case study

In comparison to the basis case study, materials were removed (Table 5.3) and a change in simulation variables' values (Table 5.4) occurred, with a resource processing time variability and restock threshold quantity being introduced. A diagram of sprint 3 case study's manufacturing setup is presented in Fig. 5.2.

A material restock event was scheduled before the simulation start due to the material absence on DemoStep1. As the restocked materials were being moved to DemoStep2, a new material was being created on DemoStep1, validating the correct implementation of the material restock functionality.

All registered processing times were in the $[20, 40]$ seconds value range, validating the variability in resource's processing time.

5.2.4 Sprint 4 - Case Study and Results

In sprint 4, simultaneous material processing at same resource, losses with variable amount, multiple resources per step, resource unavailability and service waiting queues was implemented, which required a change to the sprint 3 case study.

Table 5.5: Sprint 4 case study entities

Entity	Amount	Names
Facility	1	F1
Flow	1	DemoFlow
Step	5	DemoStep1, DemoStep2, DemoStep3, DemoStep4, DemoStep5
Resource	6	DemoResource1, DemoResource2, DemoResource3, DemoResource3_Limited, DemoResource4, DemoResource5
Service	5	DemoService1, DemoService2, DemoService3, DemoService4, DemoService5
Product	1	DemoProduct
Material	3	DemoMaterial1, DemoMaterial2, DemoMaterial3
Loss Reason	1	DemoLoss

Table 5.6: Sprint 4 case study variables and values

Variable	Mean	Variability
Resource Processing Time	30s	10s
Resource Repairing Time	60s	10s
Resource Breakdown Probability	10%	—
Loss Quantity	50	49
Restock Threshold	1000	—
Quantity Per Material	100	—

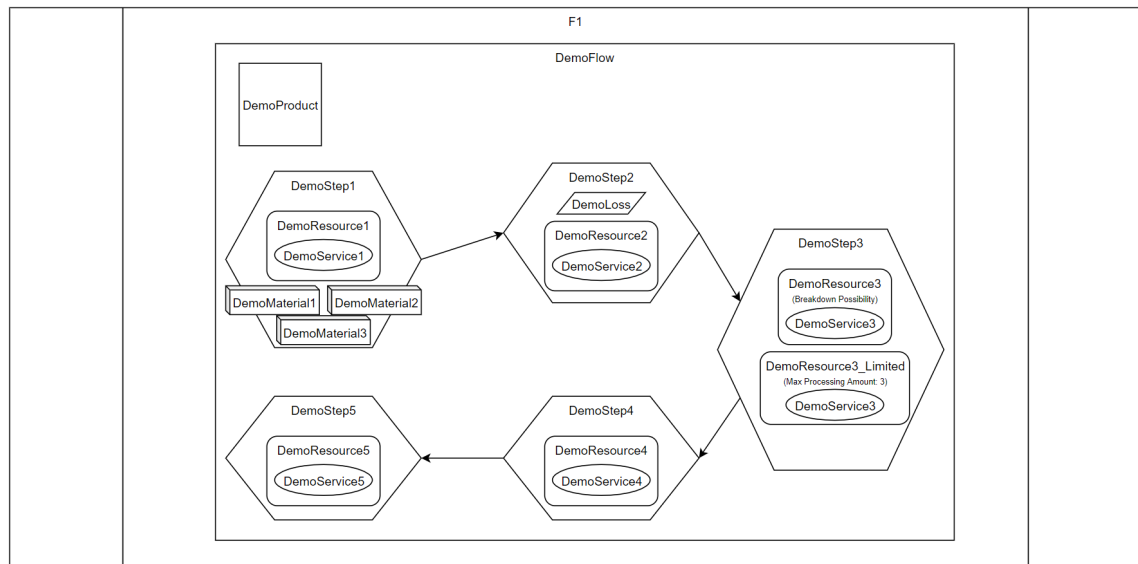


Figure 5.3: Diagram explaining the manufacturing setup used as sprint 4 case study

In comparison to sprint 3 case study, three materials were added to DemoStep1, a loss reason was added to DemoStep2, and a resource called DemoResource3_Limited with a limited processing capacity of 3 materials in simultaneous was added to DemoStep3, providing DemoService3

(Table 5.5).

DemoResource3 has an unavailability probability of 10% for every tracked out material. Other simulation variables' values were changed (Table 5.6). Resource repairing time and unavailability (breakdown) probability were introduced, along with loss quantities. The restock threshold at DemoStep1 was increased to 1000, enforcing a minimum of 10 materials at that step.

A diagram of sprint 4 case study's manufacturing setup is presented in Fig. 5.3.

Materials arriving at DemoStep3 presented a quantity in the [1, 99] range due to the loss reason at DemoStep2, validating the loss functionality with variable quantities.

DemoResource3 registered unavailability periods ranging from 50 to 70 seconds, validating the resource unavailability functionality with variable duration.

All resources were capable of processing more than one material simultaneously, validating the simultaneous material processing at same resource feature.

At DemoStep3, DemoResource3 and DemoResource3_Limited were capable of processing materials at the same time, validating the ability to manage multiple resources per step.

Whenever DemoResource3 became unavailable while DemoResource3_Limited reached its maximum processing ability, the number of materials at DemoStep3 increased considerably, reaching a peak of 20. As DemoResource3 became available again, the number of materials at DemoStep3 stabilized at around 10, the minimum material threshold for DemoStep1. The stated observations validated the service waiting queues implementation.

5.2.5 Sprint 5 - Case Study and Results

In sprint 5, the time acceleration feature was implemented. Since this feature only changes event duration, the case study used for the previous sprint was maintained with the acceleration variable being introduced. Resource processing and repairing times were eliminated to easily understand how event durations are affected by acceleration (Table 5.8).

Table 5.7: Sprint 5 case study entities

Entity	Amount	Names
Facility	1	F1
Flow	1	DemoFlow
Step	5	DemoStep1, DemoStep2, DemoStep3, DemoStep4, DemoStep5
Resource	6	DemoResource1, DemoResource2, DemoResource3, DemoResource3_Limited, DemoResource4, DemoResource5
Service	5	DemoService1, DemoService2, DemoService3, DemoService4, DemoService5
Product	1	DemoProduct
Material	3	DemoMaterial1, DemoMaterial2, DemoMaterial3
Loss Reason	1	DemoLoss

Table 5.8: Sprint 5 case study variables and values

Variable	Mean	Variability
Resource Processing Time	30s	—
Resource Repairing Time	60s	—
Resource Breakdown Probability	10%	—
Loss Quantity	50	49
Restock Threshold	1000	—
Quantity Per Material	100	—

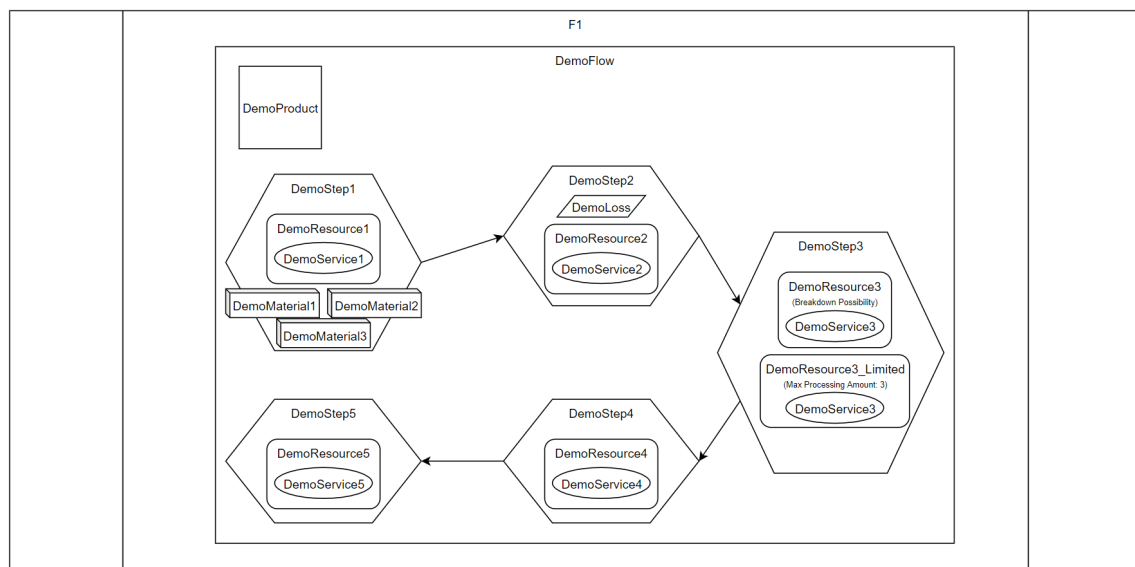


Figure 5.4: Diagram explaining the manufacturing setup used as sprint 5 case study

The acceleration feature was tested with values ranging from 1 to 100. The registered resource processing and repairing times were in accordance with the used acceleration. Whenever acceleration was changed during a repairing or processing operation, the duration was properly recalculated.

Event processing started behaving abnormally with acceleration values high enough to produce processing and repairing times smaller than 0.500 seconds. However, no limitation was imposed to the maximum acceleration, since original processing and repairing times can assume any value in real simulation scenarios. It is up to the user to choose the acceleration value carefully.

5.2.6 Sprints 6, 7 and 8 - Case Study and Results

Sprints 6 through 8 were only focused in implementing the database date conversion feature. Although no new features have been added to the simulator, new entities were added to increase the case study complexity, allowing a more meticulous test to all the implemented functionalities.

Table 5.9: Sprints 6 through 8 case study entities

Entity	Amount	Names
Facility	1	F1
Flow	2	DemoFlow, DemoFlow_2
Step	10	DemoStep1, DemoStep2, DemoStep3, DemoStep4, DemoStep5, DemoStep1_2, DemoStep2_2, DemoStep3_2, DemoStep4_2, DemoStep5_2
Resource	10	DemoResource1, DemoResource2, DemoResource3, DemoResource3_Limited, DemoResource4, DemoResource5, DemoResource1_2, DemoResource3_2, DemoResource4_2, DemoResource5_2
Service	10	DemoService1, DemoService2, DemoService3, DemoService4, DemoService5, DemoService1_2, DemoService2_2, DemoService3_2, DemoService4_2, DemoService5_2
Product	3	DemoProduct1, DemoProduct2, DemoProduct_2
Material	3	DemoMaterialP1, DemoMaterialP2, DemoMaterial_2
Loss Reason	2	DemoLoss, DemoLoss_2

Table 5.10: Sprints 6 through 8 case study variables and values

Variable	Mean	Variability
Resource Processing Time	30s	10s
Resource Repairing Time	60s	10s
Resource Breakdown Probability	10%	–
Loss Quantity	50	49
Restock Threshold	1000	–
Quantity Per Material	100	–

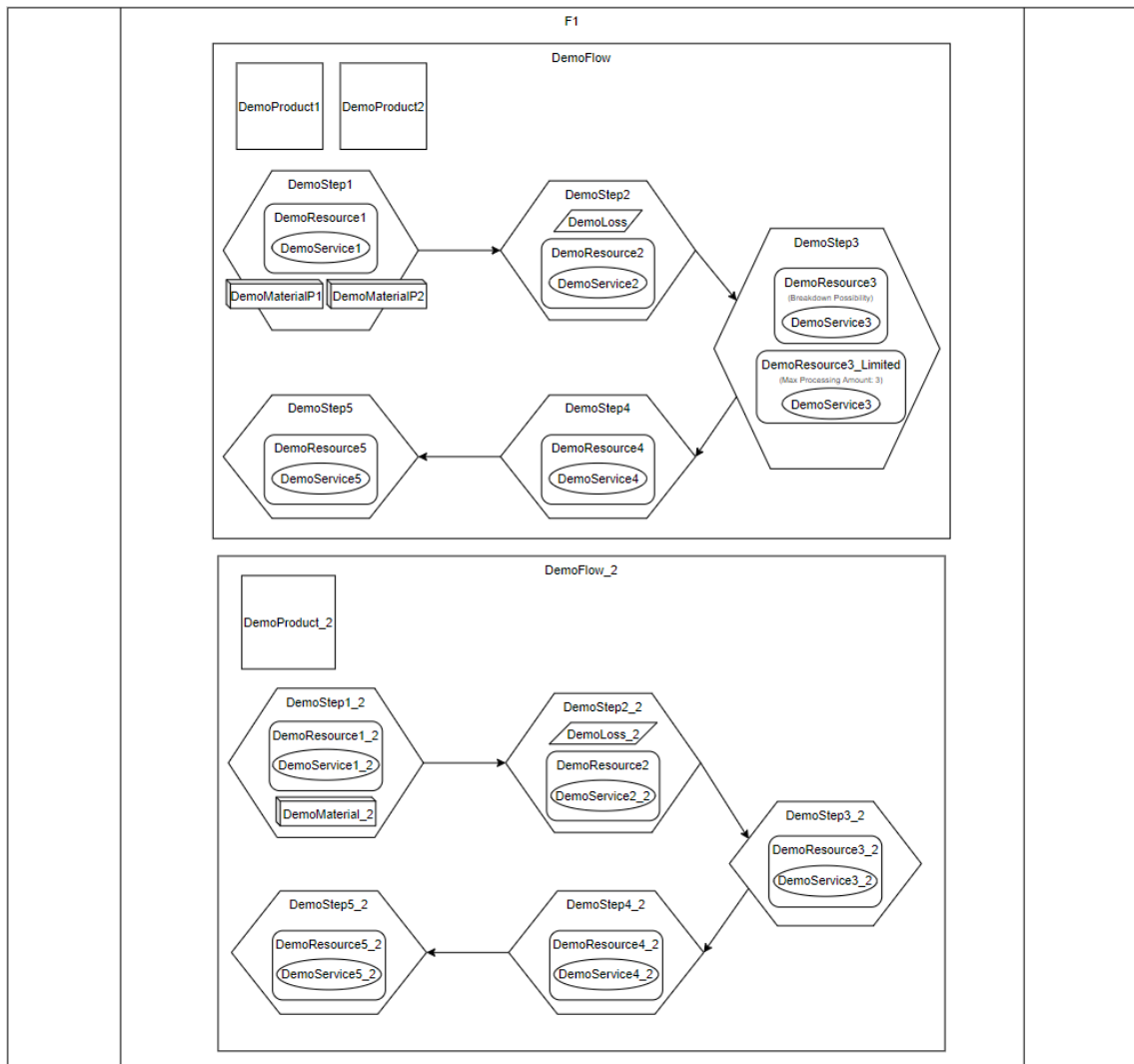


Figure 5.5: Diagram explaining the manufacturing setup used as Sprints 6 through 8 case study

In comparison to sprint 5, a new flow was created with 5 new steps and a new product, with a loss reason on DemoStep2_2. DemoResource2 now provides 2 services (DemoService2 and DemoService2_2) at 2 different steps (DemoStep2 and DemoStep2_2). DemoFlow now contains 2 products. Although there was a changed in the entities used (Table 5.9), the variables remained the same (Table 5.10). A diagram of sprints 6 through 8 case study's manufacturing setup is presented in Fig. 5.5.

The implemented functionalities weren't compromised with the increase in the case study complexity.

Table 5.11: Simulation information used to validate date modification

Simulation Information	Value
Acceleration	1 and 4
Real Date Starting Time	2020-06-17T01:24:26
Simulation Date Starting Time	2021-06-17T04:24:26

The date modification occurring at the database was tested with acceleration values of 1 and 4 to replicate date conversion with and without acceleration (Table 5.11). Random records added to Online sub-database were compared to their equivalent version in ODS sub-database.

From the 10 randomly selected database records with both acceleration values 1 and 4, all presented a correct date conversion. The observed results validated the date conversion algorithm.

Chapter 6

Conclusions and Future Work

This chapter presents conclusions drawn from the dissertation development. Future improvement work is also suggested.

6.1 Conclusions

Simulation tools can aid companies in many areas. The possibility to test multiple scenarios and preview their outcome without their actual implementation can bring great value to an organization. One of the benefited areas is manufacturing. Enterprises can validate different production models and test hypothetical equipment arrangements by emulating their results through simulation tools. With the Industry 4.0 era, simulation limits are extending further and companies are starting to understand how technology can improve their results.

The goal of this dissertation was to develop a simulation software capable of properly representing and simulating a shop floor's manufacturing system. The software purpose was to cycle through the execution of a sequence of material and equipment transactions in a stochastic environment to properly replicate the manufacturing processes of a real factory.

The first step was to define the simulation functionalities suitable for the dissertation project. The adopted working methodology was based on *Critical Manufacturing*'s Scrum ideology adopted by its employees. This methodology allowed to have a testable simulator prototype from early on with small functionality increments at each sprint.

The simulator was integrated with *Critical Manufacturing*'s MES solution, which enforced the use of its provided API. The acceleration aspect allied to the necessity to generate data according to the correct simulation timetable forced the database date conversion so that the generated data could be used by KPIs and later on for presales demonstrations.

The creation of multiple case studies that originated different testing scenarios had an important role in validating the implemented simulation software with realistic environments. The

adopted scrum methodology enforced frequent testing of the implemented functionalities, which also increased the confidence in the simulation results.

The proposed objectives were met and the final simulator software solution is definitely a start to aid companies visualize their manufacturing processes in action with the help of MES.

The developed project was of complex implementation, increased by the necessity to work with a prebuilt software tool and database. The extra effort required to integrate the simulation software with a complex unknown commercial MES system and its database can justify the dissertation's lack for academic complexity.

6.2 Future Work

After performing a retrospective on the developed project some improvement ideas can be suggested for future work to improve the simulator abilities.

Although the main MES functionalities are supported by the simulator, there is a wide range of important MES features that are not considered by the software. The most noticeable are optional steps, sub-materials, material transfers, BOMs and support for factories with several facilities.

Automated tests would also present a great improvement by diminishing the required testing time caused by the manual setup of testing scenarios and subsequent result validation. Test automation would also allow to easily increase the case studies' complexity by getting more entities involved, leading to more thorough test cases.

Another valuable add-on would be an interface with the possibility to indicate parameter values associated to instances, such as, resource processing times, resource repairing times, loss quantities, etc.

References

- [1] Narciso M. S. C. Caldas. Abordagem híbrida de otimização/simulação para a gestão de cadeias de abastecimento flexíveis, 2018.
- [2] Ilya V. Grigoryev. *AnyLogic 8 in Three Days: A Quick Course in Simulation Modeling*. Fifth edition, 2018.
- [3] H. Meyer, F. Fuchs, and K. Thiel. *Manufacturing Execution Systems: Optimal Design, Planning, and Deployment*. McGraw-Hill, 2009.
- [4] Ken Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.
- [5] Mohammed Ali Berawi. Utilizing big data in industry 4.0: Managing competitive advantages and business ethics. *International Journal of Technology*, 3(1):430–433, 2018.
- [6] Jerry Banks. Introduction to simulation. In *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, pages 7–13, 1999.
- [7] D. Mourtzis, M. Doukas, and D. Bernidaki. Simulation in manufacturing: Review and challenges. *Procedia CIRP*, 25:213–229, 2014.
- [8] Robert E. Shannon. *Systems Simulation: The Art and Science*. Prentice Halls, 1975.
- [9] Mauricio Pereira dos Santos. Introdução à simulação discreta. *Rio de Janeiro: UERJ*, 1999.
- [10] Francisca C. Pereira. Modelos integrados de otimização/simulação para balanceamento de linhas de produção e alocação dinâmica de recursos, 2019.
- [11] Christopher A. Chung. *Simulation modeling handbook: a practical approach*. CRC press, 2003.
- [12] Romão F. D. Santos. Estratégias híbridas de machine learning e simulação para a resolução de problemas de escalonamento, 2018.
- [13] Ricki G. Ingalls. Introduction to simulation. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 1374–1388, 2011.
- [14] S. Terzi and S. Cavalieri. Simulation in the supply chain context: a survey. *Computers in industry*, 53(1):3–16, 2004.
- [15] Andrei Borshchev. *The Big Book of Simulation Modeling: Multimethod Modeling with AnyLogic 6*. 2013.
- [16] Antuela A. Tako and Stewart Robinson. The application of discrete event simulation and system dynamics in the logistics and supply chain context. *Decision support systems*, 52(4):802–815, 2012.

- [17] José G. P. Torres. Plataforma web para simulação de linhas de produção, 2019.
- [18] Peer-Olaf Siebers, Charles M. Macal, Jeremy Garnett, David Buxton, and Michael Pidd. Discrete-event simulation is dead, long live agent-based simulation! *Journal of Simulation*, 4(3):204–210, 2010.
- [19] Charles McLean, Al Jones, Tina Lee, and Frank Riddick. An architecture for a generic data-driven machine shop simulator. In *Proceedings of the Winter Simulation Conference*, volume 2, pages 1108–1116, 2002.
- [20] Tiago A. L. Constante. Contribution for a simulation framework for designing and evaluating manufacturing systems, 2018.
- [21] Kiatkajohn Worapradya and Thaweepat Buranathiti. Integration of manufacturing execution system and simulation. 2007.
- [22] Julie Fraser. Mes explained: A high level vision. Technical Report 6, MESA Internacional, 1997.
- [23] B. Saenz de Ugarte, A. Artiba, and R. Pellerin. Manufacturing execution system – a literature review. *Production Planning & Control*, 20(6):525–539, 2009.
- [24] E.M. Shehab, M.W. Sharp, L. Supramaniam, and T.A. Spedding. Enterprise resource planning: An integrative review. *Business Process Management Journal*, 10(4):359–386, 2004.
- [25] Nada Sanders. *The Definitive Guide to Manufacturing and Service Operations: Master the Strategies and Tactics for Planning, Organizing, and Managing how Products and Services are Produced*. Pearson Education, 2013.
- [26] Gary A. Langenwalter. *Enterprise Resources Planning and Beyond: Integrating Your Entire Organization*. St. Lucie Press, 2000.
- [27] Julie Fraser et. al. Mes functionalities & mrp to mes data flow possibilities. Technical Report 2, MESA Internacional, 1997.
- [28] Michael McClellan. Introduction to manufacturing execution systems. In *MES Conference & Exposition*, pages 3–11, 2001.
- [29] A. Srivastava, S. Bhardwaj, and S. Saraswat. Scrum model for agile methodology. In *International Conference on Computing, Communication and Automation (ICCCA)*, pages 864–869, 2017.
- [30] F. Paetsch, A. Eberlein, and F. Maurer. Requirements engineering and agile software development. In *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 308–313, 2003.
- [31] M. Mahalakshmi and M. Sundararajan. Traditional sdlc vs scrum methodology – a comparative study. *International Journal of Emerging Technology and Advanced Engineering*, 3(6):192–196, 2013.