

Comparative Assessment of a Collaborative ModelOps Platform

Henri Katvio

School of Science

Thesis submitted for examination for the degree of Master of
Science in Technology.
Espoo 2023

Supervisor

Prof. Ahti Salo

Advisors

MSc (Tech) Antti Villberg

PhD Sami Majaniemi

Copyright © 2023 Henri Katvio



Author Henri Katvio

Title Comparative Assessment of a Collaborative ModelOps Platform

Degree programme Master's Programme in Computer, Communication and
Information Sciences

Major Security and Cloud Computing

Code of major SCI3084

Supervisor Prof. Ahti Salo

Advisors MSc (Tech) Antti Villberg, PhD Sami Majaniemi

Date 2023

Number of pages 65

Language English

Abstract

Many software platforms are dedicated to making computer model creation more straightforward and accessible to wider user groups. Enabling collaboration between different types of users and accessing the models in a continuous fashion, however, has remained a challenge. ModelOps-term has been coined to resemble the existing DevOps and MLOps terms but to be more specific as regards integrating modelling and operations. This Thesis presents a generic model and language-agnostic ModelOps platform called Modelling Factory. A comparative assessment of the platform is performed in comparison with the chosen state-of-the-art proprietary and open source solutions. In addition, some important functionalities of the platform are demonstrated in two use cases.

Keywords ModelOps, Modelling Factory, Continuous Integration, Continuous Delivery, Computer modelling and simulation, Digital platforms



Tekijä Henri Katvio

Työn nimi Kollaboratiivisen ModelOps-Alustan Vertaileva Arviointi

Koulutusohjelma Master's Programme in Computer, Communication and
Information Sciences

Pääaine Security and Cloud Computing

Pääaineen koodi SCI3084

Työn valvoja Prof. Ahti Salo

Työn ohjaajat DI Antti Villberg, TkT Sami Majaniemi

Päivämäärä 2023

Sivumäärä 65

Kieli Englanti

Tiivistelmä

Monet ohjelmistoalustat on ensisijaisesti tarkoitettu tietokonemallien luomisen helpottamiseksi ja niiden tuomiseksi laajemmille käyttäjäryhmille. Eri käyttäjäryhmien välinen yhteistyö ja mallien jatkuva saatavillaolo on kuitenkin edelleen pysyvä haaste. Olemassaolevia termejä DevOps ja MLOps mukailleen käyttöön on otettu uusi termi ModelOps kuvaamaan mallinnuksen ja ohjelmistokehityksen operatiivisen toiminnan välistä suhdetta. Tämä diplomityö esittelee geneerisen malli- ja kieliagnostisen ModelOps alustan nimeltään Modelling Factory. Alustaa arvioidaan vertaillen sitä erilaisiin uusimpiin sekä suljetun että avoimen lähdekoodin ohjelmistoihin. Lisäksi alustan tärkeimpiä ominaisuuksia esitellään kahden käyttötapauksen kautta.

Avainsanat ModelOps, Modelling Factory, Jatkuva integrointi, jatkuva jakelu,
Tietokonemallinnus ja -simulointi, Digitaaliset alustat

Contents

Abstract	3
Abstract (in Finnish)	4
Contents	5
1 Introduction	7
2 History and State-of-the-Art	11
2.1 The History of Modelling	11
2.2 Modern Tools and Their Support for Modelops	13
2.2.1 Functional Mockup Interface	13
2.2.2 Open Simulation Platform	16
2.2.3 Insight Maker	17
2.2.4 Posit, RStudio and R Markdown	19
2.2.5 Excel Simulation Environment	20
2.2.6 Mathematica and Wolfram Demonstrations Project	21
2.2.7 Python Ecosystem	23
2.2.8 Machine Learning Models	24
2.2.9 Virtual Materials Market Place	26
2.2.10 Simulink	27
2.3 Summary	29
3 Modelling Factory	30
3.1 Simantics Platform	30
3.2 The Role of Modelling Factory	33
3.3 Features	36
3.4 Technologies Used	39
3.5 The Structure of Modelling Factory	41
4 Evaluation of Modelling Factory	44

	6
4.1 Use Case 1 - Wood Product Process	44
4.2 Use Case 2 - CORTOOLS	47
4.3 Summary	52
5 Comparison With State-of-the-Art	53
6 Discussion	58
7 Conclusion	60
References	61

1 Introduction

With the ever-increasing computational power, more complex problems can be solved with computers. One approach for solving these problems is computer modelling. Computer modelling and simulation have been popular topics, and many frameworks and platforms have been developed to make creation of computer models easier. First, it is helpful to define some terminology.

Often, computer and simulation models are used interchangeably, but there is a clear distinction to be made between them. A computer model is a program that represents either a process or a system [1]. Simulation models are a subset of computer models with the distinguishing factor that they are not static and can be stepped through time so that their state changes. Another flavour of computer models are static models, whose results are computed only once as they do not change over time. The former group comprises, for example, so-called system dynamics and agent-based models, which focus on system-level features and individual system component dynamics, respectively. These two terms are used and specified in more detail in section 2.2.3. [2]

Typically, computer software goes through five distinct phases during its lifecycle. These stages are initial development, evolution, servicing, phaseout and closedown. [3] A computer model follows the same lifecycle. In the first phase, (1) the model is created, then (2) it is tested and fixed to be more accurate or faster. In the third phase (3), it is used to obtain results, and then (4) it becomes obsolete as better models are developed. Finally, (5) the model is retired. As mentioned, many platforms are designed to enable easy creation and usage of computer models. These platforms focus primarily on the first part of that lifecycle, the creation. Henceforth, they are called computer modelling and simulation platforms. A platform focused on the second and third part of the lifecycle, aiming to service continued development, deployment and usage of models as efficiently as possible, is called a ModelOps platform.

Why has simulation become an important area of focus where large amounts of

money and human resources are committed? The significant advantage of simulation models is that they allow us to test processes, products and new ideas before committing to creating and testing them in the real world. Thus, development becomes cheaper and more environmentally friendly. Another factor is the scarcity of data in some situations. For example, catastrophes, such as earthquakes, are in an area with no large amounts of data, and the infrequency of events makes the data hard to collect. Nevertheless, the data would be useful in loss estimation for insurance companies and engineering to prevent losses. [4]

The construction of good quality static and dynamic computer models typically requires knowledge from several different areas: coding and application domain knowledge and logical and mathematical mastery to formulate the problem for effective computation properly. Seldom the same person possesses all these required skills. In addition, for the results to be useful for the end users, typically high-level decision-makers, the model outcome must be formulated in the language they understand. When the results from the model are obtained, it can be difficult to interpret them without the help of an expert. The said simulation expert is often different from the person interested in the results; that is the role of the decision maker. This decision-maker, however, may not know how to run the model. In general, it would be advantageous to offer access to the models for all the above-mentioned user groups in their "native language", specific for the knowledge representation in their respective fields. This access should be continuous so that long waiting periods can be avoided and the end users, e.g. decision-makers, could participate in developing a model, which, eventually, is supposed to serve them in the best possible way. The same philosophy applies to the outcomes of the models, as well. When models are equipped with graphical user interfaces (web app GUI), the end users (decision-makers, designers etc.) can perform dynamic planning more effectively as they can help themselves to produce new model-based business scenarios or product performance predictions without wasting much time hiring external consultants to do it for them. In a very concrete sense, GUIs enable the models to be used as computational engines, and the GUI acts as a "dynamic report", which the end user

can produce without consulting an expert. Naturally, sometimes it is also a good idea to get expert help interpreting the results, but in straightforward cases, end users can help themselves cost-effectively.

The ModelOps paradigm presented in this Thesis is a new concept which has been coined to resemble the existing DevOps and MLOps terms. The integration and automation of the work of software development (Dev) and IT operations (Ops) is called DevOps; MLOps, on the other hand, integrates and deploys machine learning (ML) models into a usable form for higher-level end users. ModelOps paradigm aims at integration and continuous deployment of general model-based design methods so that high-level end users (e.g. decision-makers and designers) can utilize them without a deep domain, IT or modelling knowledge. ModelOps term has garnered traction and is presented in more detail in [5]. Modelling Factory¹ presented in this Thesis is a ModelOps platform. More precisely, it is a model-agnostic platform for publishing and interacting with the simulation models, initially based on the Simantics platform but later developed to work independently of it.

Though the idea of an easy-to-use platform for model publishing is not novel, a de facto solution for the industry has yet to emerge. Currently, no fully-fledged ModelOps platforms exist. The working hypothesis of this Master's Thesis is that the Modelling Factory platform (VTT's trademark product) offers one possibility for changing this situation. The combination of model paradigm and language agnostic integration, publishing and deployment features of the Modelling Factory platform to be presented in the subsequent chapters make it a novel project in its generality.

This Master's Thesis will look at the Modelling Factory -software, try to document its usage, and explore how well it works in bridging the gap between the simulation results and the decision-makers. Section two of the Thesis is a survey of other platforms and frameworks on the market to determine if software like Modelling

¹Modelling Factory 1.0 version is currently available at address <https://modellingfactory.org/>. Modelling Factory 2.0, which is described in this Thesis, answers at <https://gitlab.systis.modelingfactory.org/>. The 2.0 version will be moved to VTT controlled servers during spring 2023, which also causes a change in some of the web addresses related to the service.

Factory is needed. The third section describes the functionality of Modelling Factory, and the fourth section evaluates its functionality through two use cases. After the functionality of the Modelling Factory is examined, it is compared against the market offerings in section five, after which the results are discussed and summarised.

2 History and State-of-the-Art

This section first details the history of computer modelling and simulation and then gives an overview of the current situation of modelling. The examples presented here can not cover every technique and solution on the market, but they provide a good overview of what is available. What can be seen in the examples in this section is that different solutions are often specific to one domain, and no fully general mainstream solution exists.

2.1 The History of Modelling

In a way, humankind has always created representations to simplify and comprehend different topics. The first numbering systems were created to represent and communicate different amounts of goods traded and catalogued for taxation and other bookkeeping purposes. This would make the history of modelling roughly thirty thousand (30000) years old. For modern-day computer models and simulations, starting at the first numbering systems is not meaningful. Instead, starting at the ordinary differential equations makes more sense since much of the modelling and simulation is based on ODEs. [6]

Ordinary differential equations started their history in the mid-1600s [7]. One of the earliest contributors to differential equations and their notation is Sir Isaac Newton, though Gottfried Wilhelm von Leibniz influenced modern notation, specifically. Even though the first works detailing differential equations were more of an extension of calculus at the time, they quickly started to be used for other purposes. The methodology was applied to the practical problems by finding tangents and areas of closed curves or explaining the properties of the real world [8]. This has been refined during the following hundreds of years by notable mathematicians, such as Euler, Lagrange and Laplace. Of these famous figures, Euler, for example, has heavily contributed to the meaning of functions. In the 19th century, Joseph Fourier initiated the investigation into the Fourier series and contributed to Fourier analysis. Maxwell is another notable name that expanded the applications into other fields such as

heat theory, optics, electricity and magnetism. It was clear that the applications where mathematical models could be applied were becoming increasingly intensive to calculate by hand or even by using the available mechanical calculators. A breakthrough in this area came in 1941 when the Atanasoff-Berry computer or ABC [9] was created. It was specifically designed to solve differential equations. The more famous and often hailed as the first electronic computer, ENIAC, was completed in 1945. The first-ever task given to the computer to calculate was a numerical simulation used in design of a hydrogen bomb. This illustrates, besides the concurrent times and political climate, the need for computers in more complex tasks. Calculating the task took twenty seconds from ENIAC compared to forty hours by using a mechanical calculator. [10]

The advent of computers kicked off the research and development of appropriate methods for simulation. While computers were getting faster and more powerful, writing the models for them was still tricky. Several Simulation Programming Languages (SPL) were created to ease the development process, such as SIMULA in the 1960s. Nevertheless, more models were created using General Programming Languages (GPL) than SPLs. Developing models for computer modelling and simulation remains a nontrivial exercise because the system characteristics should be translatable into a logical language, and the results must be computable in a finite time. Purely data-based AI models, while often useful in relieving the modeler from the formulatability of the problem in terms of mathematical laws, are also often more unreliable in untested parameter domains and require huge amounts of training data and time. The existence of SPLs do not make said part of the model creation process easier. Thus the next step was to create Conceptual Frameworks to support the translation of system characteristics. [8] The frameworks often rely on graphical representations, such as flow charts and block diagrams. Different visualisation techniques and natural language have been and continue to be potential development targets. [11] Now computer modelling and simulation is used for a vast domain, such as crash dummy crashworthiness simulation [12], simulating feedback mechanisms in adolescents [13] and digital twins to get and provide feedback from and to processing

plants [14].

2.2 Modern Tools and Their Support for Modelops

There are numerous solutions for modelling, each for a different purpose. This means that each solution could have differing requirements for deploying the models created by using them. It is also much extra work for the modeller and the decision-maker to manage multiple different methods for making and using the models. This subsection covers a look into the current situation and prominently available technologies.

Modelling and simulation platforms are functionally different from ModelOps platforms and aim to offer solutions for different parts of the model life cycle. In this section, the technologies and platforms chosen to be presented in greater detail mostly implement features typical for modelling and simulation platforms and, to some extent, ModelOps platforms. Since Modelling Factory is, first and foremost, a ModelOps platform that offers solutions for deploying and interacting with general models, it does not make sense to take a deeper look at every modelling and simulation platform with minimal or partial ModelOps capabilities or with a minimal application range. Their existence is recognised but omitted from this section. Some examples of these platforms are SIMULIA Simulation Portfolio by Dassault Systèmes [15], LMS Virtual.Lab by Siemens [16], Adams by Hexagon [17], AnyLogic by The AnyLogic Company [18], and Stella Simulator by isee systems [19].

In order to better understand how the different libraries and software relate to one another, they are assigned a set of attributes or properties inherent to the solution. The attributes for categorising the solutions are as follows:

2.2.1 Functional Mockup Interface

Functional Mockup Interface (FMI) is a free standard and specification for defining and packaging different models to guarantee their interoperability. The standard is still in active development as a Modelica Association project, and new versions are introduced frequently to account for the rapidly changing landscape of modelling.

Attribute	Explanation
Type of solution	Categorises between complete platforms and libraries meant to augment other technologies or platforms.
Domain of solution	Simulation and modelling platforms are different things than generic ModelOps platforms. Differentiates between these two.
Specificity	A solution can work with a specific technology, or it can be a general solution and work with many technologies.
Open-source	Whether the solution is open source or not.
Active development	Whether the solution is still being actively developed or the development has stopped.

Table 1: Different attributes for categorising solutions.

The newest major version was introduced in 2021, that being FMI 3.0. As stated, the standard encompasses different types of models, such as static or Monte Carlo models, dynamic simulations or stochastic process models. [20]

The FMI is very important in guaranteeing interoperability and co-simulation possibilities and is widely used across industries in modelling. Modelling Factory uses Functional Mockup Units as well as a tool to enable co-simulation possibilities across programs and models that would not otherwise support it.

Functional Mockup Interface defines a container of XML files, binaries and C-code. This container or package is called the Functional Mockup Unit or FMU, an implementation of the Functional Mockup Interface. The XML file usually contains the interface definitions in an easily readable format. Interface definition in this context means information about the inputs for the FMU, the outputs and the information of their units and formats. In short, these XML files contain the metadata for using the model, which is usually in compiled form as binaries. Since the binaries differ between platforms, it is a common practice to include compatible binaries for Windows and Linux. This is meaningful since, in a production environment, Linux is often used as the underlying platform, which still holds for containerised applications using technologies such as Docker and Kubernetes. On the other hand, Windows is more often used in the modellers' work computers; as such, it is helpful for the FMU to also work in a Windows environment.

As the documentation for FMI states, the standard contains three different interface types for various purposes. These are co-simulation, model exchange and scheduled execution.

All of these interface types have significant parts in common. The FMI C Application Programming Interface, XML-based description schema and ZIP file-based distribution mechanism are all shared between the different types of FMIs. The C API contains standardised C functions that the importer can call to execute the required computations. The description schema defines how the model description file called “modelDescription.xml” is structured and what it contains. The modelling environment generates this file. All the files needed, model description files, binaries and required libraries are comprised in a ZIP file. ZIP file is a good choice as a transmission medium since it is widely supported on different platforms and allows for easy compression of the files, even when that is a secondary and not very meaningful feature.

In FMI for Co-Simulation specification, the FMU implements the model algorithm and the required solution method and it is the most self-contained package of the three. In this mode, the FMU contains the models and their solvers as runnable code exported by the modelling environment. The importer can run several different FMUs with the help of the co-simulation algorithm. The algorithm itself is not part of the interface definition and is the responsibility of the importer. The responsibility of this algorithm is to advance the overall simulation time as well as to exchange the input and output data, trigger the input clocks and handle events. In short, the co-simulation algorithm is responsible for coordinating the different subsystems. The subsystem is only one part of the complete system and can be formed into an FMU, but it is usually contained in the FMU of the entire system. [21, 22]

The FMI for Model Exchange is simpler in terms of what the FMU contains and hence requires more from the importer. The FMU implements the model algorithm in this mode but not the solver. It is up to the importer to have a fitting solver that the FMU can use. This means that the FMU implements an interface for solving

ordinary differential equations. Co-simulation FMU is often evaluated in discrete time steps. In contrast, a model exchange FMU is evaluated only at a specific time instant, which means they might be only discrete time equations. Co-simulation FMU can also be fully calculated in just a single step, so if this is required, the modeller is not restricted to using FMI for model exchange. Model exchange FMU can also enter a continuous-time mode where all the continuous-time variables are updated between the events. Performing the integrator steps is left to the importer. [23, 22]

A model partition is an algorithm that computes a subset of variables and is connected to a clock. Invoking the model partition is done by activating the clock attached to the algorithm. One FMU can contain many of these model partitions. FMI for Scheduled Execution is designed for a system with only one thread available for computation but a need for concurrent computation of multiple partitions on that thread. In this case, the importer is tasked with having a scheduler to control the time progress and schedule each of the model partition's execution.

2.2.2 Open Simulation Platform

Open Simulation Platform (OSP) is not as much of a platform as it is a library enabling the co-simulation capability for any simulation software. OSP was created for the maritime industry; thus, the application domain is also restricted to the maritime industry. OSP is designed to implement this goal by building a Functional Mockup Unit out of the model. For this, OSP includes a C/C++ library called *libcosim* that can be integrated into the software in need of co-simulation capability, a specification of the OSP interface, demo models and a demo application. [24]

The demo application of OSP uses the supplied co-simulation reference model called *DP-Ship*. The reference model comprises five smaller models working together to produce the results. The FMUs in DP-Ship are named *OSOM*, *NLPobserver*, *ReferenceGenerator*, *DPCcontroller* and *ThMPC*. A more thorough look at the FMUs, their functions, and their parameters are available on Open Simulation Platform reference models page in [25].

Open Simulation Platform is a valuable tool when solving a problem requiring co-simulation, but it is limited in other cases. The demo application appropriately displays co-simulation capability, but other parts of the system are left to be implemented differently. The server for the demo is written in Go and the user interface is written with Clojurescript. While this approach is admittedly flexible, it makes implementing and deploying the application still a big undertaking with much application-specific code. To summarise, Open Simulation Platform is a helpful library when the problem domain lies in the maritime industry.

2.2.3 Insight Maker

Insight Maker is a free and open-source web-based software for creating system dynamics models as well as agent-based models. The constructed models are called, as the name suggests, Insights. Two significant features of Insight Maker are sharing and collaborating in terms of Insights. All the created Insights are, by default, public, and anyone can view or run them. Sharing Insight with another person happens by sending them a link to Insight, published as an interactive web app. It can also be found by exploring public Insights. This does not allow the collaborator to edit the model, however, but permission can be granted from the properties of Insight. If Insight is not ready to be made public, it can be set to private, though that is not the intended standard operating mode. Public models can also be easily shared by copying a provided HTML code and adding that to a web page or blog post, thus embedding the model to the page. The embedded model can be configured and run directly from the embedding. [26]

The system dynamics models are meant for simulating systems on a coarse level without any detailed knowledge of the dynamics of individual system constituents. These models are essentially time-delayed ordinary differential equations, meaning they are solved numerically to explore how interesting variables evolve. The building blocks for the system dynamics models, called primitives, are simple but can be easily combined to create complex models. There is no specific limit as to how complex a model can get. There are four types of primitives: stocks, flow, variables and links.

Stocks store materials, flows move materials between stocks, variables are calculated constants or dynamically updated values, and links connect two primitives to show they are related.

Agent-based models are meant for use cases where the interest lies in the behaviour of an individual object or agent [27]. Agent-based models typically use geography to create the models. There are two options for geography, spatial geography and network geography. Spatial geography is the more common of the options. It refers to a plane where agents are mapped based on latitude and longitude, while network geography means creating a network of agents by connecting them. Several helper functions help place the agents on a spatial plane or link them to create a network.

A lot of the modelling is based on equations that are defined using a domain-specific language (DSL) created for defining the equations. The language is simple and mostly defines accessing variables with square brackets and mathematical operators. For those cases where the ready blocks or DSL is not enough, Insight Maker provides an API to be used with JavaScript.

There are cases where the user might want to use the simulation outside the Insight Maker environment. However, the tools provided by Insight Maker would be beneficial in getting the model created more rapidly. For this, the team behind Insight Maker has also created a package called simulation published in GitHub [28] and NPM. The package supports both system dynamics models as well as agent-based models.

Insight Maker is extremely easy to use, collaborate and share models with. The drawback of the system is that it relies heavily on the browser for running the Insights. The NPM package could be used in a JavaScript application run with a runtime, but even that is not meant to be run outside of the browser. Additionally, the technologies available for creating the models are limited to the parameters, DSL and JavaScript. Everything being either wholly public or private does not offer great granularity in sharing the model, which does not make Insight Maker very suitable for more significant projects.

2.2.4 Posit, RStudio and R Markdown

Posit is a company that brands itself with the phrase "The open source data science company" [29], with a mission to create open source software for data science. The company was previously known as RStudio, but it rebranded itself as Posit in 2022 [30]. As the name, branding, and mission suggest, almost all of the software is open source, and the source code can be seen in GitHub under the "rstudio" organisation [31].

When writing software with the R language, a popular choice for the platform to use is the RStudio IDE (Integrated Development Environment), developed by Posit. While Posit products, such as RStudio and R, are usually used for data analysis, there are some possibilities to make system modelling with the tools. R is a general-purpose programming language and can be used to create computer models, both static and dynamic simulation models. As it is a popular language with data analysts and suitable for model creation, Modelling Factory also supports it natively.

In addition to RStudio, Posit has developed R Markdown, a markup language made to enable a notebook-like development in RStudio. This notebook environment written with R Markdown supports formatting text similarly to Markdown, adding code segments and inlined code and rendering the result as either a static document in various formats or an interactive document that can be viewed in a web browser. Interactive documents can be created by using JavaScript visualizations with *htmlwidgets*-library [32] or by using reactive components created with *Shiny* [33] Posit offers an easy way to interact with the notebooks created with R Markdown in a web browser by using RStudio Cloud. A project created there can be made public for sharing, visibility limited for easy collaboration or private when still developing the notebook. Besides sharing the R Markdown or the rendered file, the project can be put into a Git repository. As the R Markdown file is plain text, version control tools such as Git work with it without problems. A CI/CD pipeline can be set up for building and deploying the notebook. As such, the notebooks can also be used with Modelling Factory to take care of the deployment.

RStudio and the notebooks created with R Markdown are complete software solutions that are reasonably general-purpose. Besides R, the notebooks can contain other code, such as bash and Python. RStudio is a development environment, but R Markdown and notebooks can be considered a ModelOps environment since their main concern is creating a way to interface and share the code. The software is open source and still being actively developed.

2.2.5 Excel Simulation Environment

There hardly exists a software suite as widely used as Microsoft Office for typical office tasks. Included in Microsoft Office is the Microsoft Excel spreadsheet manipulation software.

Excel possesses a relatively robust suite of tools for creating different equations with usual mathematical operators and an extensive collection of functions to modify the data. This allows the user to create static models inside the spreadsheet easily. As long as the equations are correctly defined, Excel will automatically recalculate every cell depending on the changed value of the input variables. Excel is thus often used for Monte Carlo simulation, where the same calculations are done many times with varying random initial conditions. Moreover, many add-ins have been developed for process simulation.

There are mainly two different methods for interfacing with Excel, that is, by either writing variables and equations directly to cells or by using Visual Basic for Applications (VBA). The first approach is considerably easier to use, while the latter provides significantly greater flexibility. By using VBA, the user could parse XML files and call DLLs. As such, even calling an FMU from the filesystem is theoretically possible. While not practical, it would be possible to use co-simulation in an Excel spreadsheet.

While Excel is functional in creating static models and, in theory, harbours the capability for dynamic systems modelling, interfacing with the spreadsheet is limited to editing some cells that contain input variables. While Excel is a familiar tool for

almost everyone, manually inputting values to cells is not a functional user interface for the end user of the model. Collaborating in model creation is possible with the built-in tools. However, the only viable version control system for Excel files uses file history that works only when the file is saved to Microsoft OneDrive or SharePoint. In the same fashion, practical collaborating works only with OneDrive or SharePoint with the built-in tools. Using Excel for general simulation is challenging and error-prone, but static models are easy to build. Excel spreadsheets (without VBA layer) can be automatically converted into the internal data representation of Modelling Factory. While the conversion does not cover all the functions available in Excel, it supports the most commonly used in numerical modeling.

2.2.6 Mathematica and Wolfram Demonstrations Project

Wolfram is a company established in 1987 by Stephen Wolfram to improve the tools and technologies for computation and computational knowledge. The first product ever released by Wolfram was Mathematica [34], a software based on the Wolfram Language is meant for everything between calculus and chemistry, machine learning and data visualisation. Mathematica has always been the flagship product of Wolfram, and it is still available in a web browser loaded from the cloud or natively on Linux, macOS and Windows.

A big success for Wolfram was the introduction of Wolfram|Alpha, a computational engine that could interface with it with natural language. Wolfram|Alpha quickly rose to success with its ability since interfacing with it was easy and using it was free. Nowadays, it is the most well-known product of Wolfram, along with the symbolic computation program Mathematica. It has been available for over a decade and is being actively developed, with its goal to implement every known model, method and algorithm eventually. Wolfram|Alpha does not try to be a free version of Mathematica but instead a computational engine with natural language input. Very recently, the AI assisted chatbot technologies by Google, Microsoft and OpenAI [35] have become a hot battling ground. Usage of Wolfram language has been suggested [36] as one possibility of correcting the failures of the ChatGPT virtual assistant as

what comes to logic based reasoning. While the basic usage of Wolfram|Alpha is free, the more comprehensive results are behind a paywall.

In 1988 Wolfram developed the Notebooks, an interactive document for immediate computing. Today the same kind of paradigm is used by well-known Jupyter with Python, and few know that Wolfram Notebooks precede Jupyter by sixteen years. Wolfram Notebooks gather features from other Wolfram products, for example, the computational capabilities from Mathematica and the natural language processing and input from Wolfram|Alpha. The Wolfram demonstrations project is a repository of Demonstrations or interactive Wolfram Notebooks. Another way to phrase it is that it is a collection of web apps published by and for third parties, similar to open work rooms in Modelling Factory. These can be created by anyone using Mathematica and then sent to Wolfram to be included in the repository and made publicly available. The main difference between the Wolfram demonstrations project and the Modelling Factory open work room GUIs is that Modelling Factory does not restrict its users to use just one technology, or language, to produce the models.

Wolfram Notebooks and Demonstrations are incredibly powerful at handling and visualising data, computing equations and using machine learning and computer vision. It is possible to build many kinds of static and dynamic simulation models, even when simulation models could be created easier with purpose-built tools.

While the possibilities are almost endless, harnessing the power of different Wolfram products requires the user to learn Wolfram Language and use that to create the models. Collaboration is possible with Wolfram Cloud, but no real-time collaboration solutions are offered. Wolfram Cloud can be used to share all of the different projects created with Mathematica, not just Demonstrations, but that only works with Mathematica projects. Any static or simulation model that is built to work as a Demonstration must be fully self-contained since there is no co-simulation possibility implemented yet. It is however possible to convert a Wolfram Notebook to an FMU. Publishing models as Demonstrations is a complex process since the Demonstration goes through a thorough review process similar to academic

publications. The process takes much manual work and can take a long time with no guarantees of getting through.

2.2.7 Python Ecosystem

Originally Python programming language was released on the 20th of February, 1991. The idea for Python and the first implementation came from Guido van Rossum, but the language has since evolved to one of the most widely used languages contributed to by many [37]. It is now maintained and developed by a non-profit organisation called Python Software Foundation. The widespread usage of Python can be attributed to its simple syntax and ease of use. Currently, Python is ranked as the most used programming language. The language is designed to be very easy to understand and use but still be powerful. Python can be classified as an interpreted, object-oriented, dynamic, high-level, general-purpose language. [37]

Since Python has gained tremendous popularity, many libraries have been created for every purpose. Even without libraries, Python can easily be used to create both static models and dynamic simulation models. In addition to data analysis, the area that has the most widely adopted Python is probably machine learning. Different machine-learning libraries are introduced in the following section 2.2.8. In the Python universe, there also exists a large number of various libraries intended to create and calculate different kinds of models.

There exist many libraries which serve both generic modelling and scientific computing, such as the ubiquitous NumPy and Pandas tools. Many other libraries written for Python use one or both of these under the hood since they help minimise the one significant downside of Python, speed. As a dynamic and interpreted language, Python is usually comparatively slow, which is why the two libraries are written in C to provide faster running speeds.

The Python ecosystem is vast, and there exists a package for almost any imaginable application. There are multiple ways of collaborating and using co-simulation within the Python ecosystem. The standard collaboration method is to utilise a version

control tool, usually, Git. For co-simulation, the Python programs or models can be converted into FMUs, and they can be used, or a translation layer that connects two services or different models and their solvers can be placed in different microservices. Then communication between them can happen with messages. This approach is effectively the same as various web services communicating with each other using HTTP or WebSockets.

2.2.8 Machine Learning Models

In 2022 company OpenAI released a large-scale machine-learning model called ChatGPT. This has sparked much public interest in machine learning models. Still, the model's size makes running it almost impossible without access to a large computing cluster. Nevertheless, the code for ChatGPT and other similar-size models by OpenAI is proprietary and thus not available to everyone. Many have created their own models inspired by new language learning and conversation-oriented machine learning models. Multiple libraries aim to make machine learning model creation easier for the general user. The industry standard language for interfacing with the libraries is Python, which is what most libraries use. Examples of these include but are not limited to TensorFlow, Aesara, PyTorch, OpenCV, Keras and scikit-learn.

Taking a look at TensorFlow as an example. TensorFlow is a library created by Google for creating and using machine learning models easily with little code. When using TensorFlow, no intimate knowledge of the math behind machine learning is required. Additionally, TensorFlow is open source and thus easily accessible to everyone and verifiable by nature. TensorFlow is primarily used with Python, which is what examples and tutorials use, but there are also bindings for Java and C. There are also Go bindings, but they are not maintained anymore after TensorFlow release 2.10.0.

TensorFlow Hub is a repository for creating a library of reusable machine-learning models that are already pre-trained. This increases the accessibility of machine learning and reduces the computational load of everyone not needing to retrain the model. The downside is that sometimes existing biases in the learning dataset

propagate to other users as well, which is warned about in TensorFlow Hub web pages.

For the use case of experimenting with the models others have produced in their code, TensorFlow Hub is easy to use. The process for this in Python is simply importing the *tensorflow_hub* module and using its *load()* or *KerasLayer()* functions with a URL to get a pre-trained TensorFlow model ready to be used in code [38].

TensorFlow does not offer co-simulation facilities with the models created using the library. Instead, the easiest option for getting two or more TensorFlow models to work together in a co-simulation environment is to transform them into Functional Mockup Units. There is a lightweight framework to do so, called PythonFMU, which is one possibility, though not the only one. Unfortunately, it also requires a Python interpreter as well as other third-party libraries the model leverages to be present in the environment the FMU is meant to be running in. As a result, the resulting FMU will not be fully self-contained and portable. This is because Python is an interpreted language with no real possibility for it to be compiled as an executable.

TensorFlow is undoubtedly a library rather than a complete software solution. On the other hand, TensorFlow Hub exhibits features of full software but only in the way of any other website that offers downloading and uploading files. As such, it could be compared to Google Drive or Dropbox in functionality. It only works for models created with TensorFlow, so it is limited in the domain, but it scales easily to host hundreds of thousands of TensorFlow models.

Let us take another example from the ML domain. OpenCV, short for Open Source Computer Vision Library, is a library meant to enable easy computer vision and machine learning. It is licensed under the Apache 2 license, making it easy to adopt and modify for individuals and companies. While the library is written natively in C++, it offers additional interfaces for C++, Python, Java and MATLAB. Though the development of an interface is a slow process that takes years, often over a decade, more of them are being developed. Currently in development are interfaces for CUDA and OpenCL. [39]

OpenCV consists of modules, that can be freely imported based on what is the desired functionality. The modules focus on computer vision, as that is the main application domain of the library, though it also contains a module for deep neural networks (DNNs). The module is designed for image recognition, classification and segmentation with machine learning models. The models can be used with a variety of devices, such as a traditional computer, but also Android devices and web browsers. The models can also be created using PyTorch or TensorFlow and imported to be used with OpenCV, either with Python or C++. No facility of any kind is offered to help in sharing and deploying the models.

OpenCV is, by its very nature, a library and offers nothing to make it a complete software solution. In terms of generality, the intended problem domain for the library is relatively narrow, considering mainly computer vision in terms of image recognition, classification and modification. In that domain, the library is very adept at a wide variety of tasks. In theory, the library does not restrict the user regarding what kind of models created with PyTorch or TensorFlow can and can not be imported. However, the library is particular in its purpose and design. The domain of the library is modelling and simulation. As the name suggests, the project is open source, and the source code is available on GitHub and is actively being developed.

These examples highlight the situation with machine learning. The market is full of different kinds of machine-learning libraries aimed at making the creation and evaluation of machine-learning models accessible. However, nothing exists to help in publishing and interfacing with other models. OpenAI has created a playground for people to play with their models, showing that such a platform is feasible to implement, but no general solution is freely available.

2.2.9 Virtual Materials Market Place

Virtual Materials Market Place (VIMMP) is a project under CORDIS, or The Community Research and Development Information Service, a "primary source of results from the projects funded by the EU's framework programmes for research and innovation, from FP1 to Horizon Europe" [40]. It is created in collaboration with

the European Materials Modelling Council, also known as EMMC. The purpose of VIMMP is to create a suitable platform for all stakeholders for materials modelling. VIMMP was born out of the need to connect industrial clients to experts who could translate the problem into a solvable model. Virtual Materials Market Place is designed to be a web-based marketplace similar to TensorFlow Hub. However, it also integrates other materials besides just the models. As the objective states, the idea is "To enable a seamless and fully integrated environment" [41]. The Virtual Materials Market Place is based on Open Simulation Platform, enabling anyone that implements the standard an easy way for collaboration and co-simulation.

The marketplace should be located at the URL <https://vimmp.osthus.com>, but as of the beginning of 2023, the server does not respond to requests. It is noteworthy that the project ended on the 30th of June, 2022, after which the server was probably also shut down. This means that while VIMMP is a promising platform technology, it is not an instantly viable solution since it is not clear if it is in active development, nor is it easily available for testing at the time of writing this Thesis. Moreover, the application domain seems to be limited to materials modeling.

2.2.10 Simulink

MathWorks is a company that created and now further develops MATLAB [42], a well-known environment for computing with math, graphics and programming, and Simulink [43]. Simulink is an environment for creating computer models with block diagrams and simulating them in multidomain systems. The primary purpose of Simulink is to make Model-Based Design easy to employ and use in an engineering process. It allows the user of the software to create system-level dynamic models, simulate with them before moving to hardware and deploy ready code for the hardware, all while enabling the user to use agile software development practices with short iteration cycles, continuous integration and team integration, along with automated testing and code generation.

Simulink ships with a desktop application called Simulink Editor which is used to create the models. The user interface on the application offers a canvas where the

user can drag and drop ready blocks or components and connect and configure the components. After creating the models, the user can use the application to simulate the model, view and analyse the results and export the results to MATLAB for further analysis. In addition to allowing the exportation of results to MATLAB, components and variables can be imported from MATLAB. Since the components are a big part of creating the models, creating the components must be made accessible. Thus, they can be created by combining existing components, a powerful technique used in many other places too, or by coding them with MATLAB or C/C++. It is also possible to import co-simulation or model exchange FMUs as components and use them together with the rest of the components as a part of the model or as the whole model.

Simulink has built-in support for source control with both Git and Apache Subversion (SVN), as well as a Software Development Kit (SDK) to write an integration with another third-party tool. The ability to easily integrate with Git and add files to a repository enables easy collaboration with other modellers. Since Git is an industry-standard tool, the learning curve is minimal. SVN is also very widely used and, with Simulink, works similarly to Git. Since the model file extensions, *.mdl* and *.slx*, are binary files, neither tool can automatically merge model files without corrupting them. Instead, when two people are working on different computers simultaneously, they need to ensure that they are modifying different files or merging their changes to the same file manually using the editor.

Simulink does offer help with the deployment of the model. Simulink Report Generator helps to create a report documenting and sharing information about the model and the simulation results. The report can contain block diagrams, Stateflow charts, and MATLAB function blocks, and it can be exported as PDF, Microsoft Word, PowerPoint or HTML files. With Report Generator, it is possible to create web views to be embedded in HTML code and thus viewed on a web browser. Simulink additionally includes Support Packages that enable the user to create and run the Simulink models on various hardware. This feature can be used to test whether the hardware works as intended with the system or whether the system works with

the hardware for it. One possibility for exporting the model is to create either a tool coupling FMU or a co-simulation FMU out of the model. Tool-coupling FMU requires the FMU running environment to contain a Simulink Editor where the code is then run but co-simulation FMU has the solver and is thus a standalone FMU.

Simulink is a complete software solution both for computer modelling and simulation as well as deploying and using the models; it exhibits features of a ModelOps platform as well. The platform is quite general in that it does not restrict itself to any problem domain, but it only works with Simulink models and is thus specific. While the platform is closed-source and monetised, it is still actively developed by MathWorks, and new software versions are released regularly.

2.3 Summary

As the libraries and applications chosen for a closer look show, none of the previously presented platforms focus on ModelOps but instead aim to make computer modelling and simulation easier within some existing framework. Some of them do, however, have features for deploying the model and interfacing with it either through a web-based user interface or an application. None of the solutions is a purely ModelOps platform or focuses on the features characteristic of generic ModelOps platform as detailed previously. Simulink does have a comparatively wide selection of options for deploying the model to another platform. However, the focus is more on running the model on different hardware rather than making it as easily accessible as possible.

3 Modelling Factory

Modelling Factory is a platform or a collection of software meant to solve the problem of enabling complex and tool-specific model publishing while also allowing the modellers to collaborate during the model development. VTT Technical Research Centre of Finland Ltd owns the Modelling Factory trademark and develops the platform further.

The problem of disconnect between the modellers and the people whom the results of the models are meant for was explained in the previous sections. This section will explore the functionality Modelling Factory in more detail. Historically, there have been two versions of Modelling Factory. Version 1.0 is based on the Simantics Platform, which is explained first. This version could be dubbed the legacy version. The newer of the two, version 2.0, is based on GitLab, CI/CD pipelines with Helm charts and containerisation. In this thesis, the version of Modelling Factory explored is version 2.0.

The Simantics platform is explained first, followed by a more detailed explanation of the features, base technologies and the structure of the Modelling Factory platform.

3.1 Simantics Platform

The Simantics platform is a solution for integrating different simulation tools and models. It was initially developed at the VTT Technical Research Centre of Finland but has since evolved into a more significant open-source project contributed to by other companies. One notable contributor to the development of Simantics is Semantum Oy. The original view of Simantics has been presented in [44]. Further historical steps have been documented in [45] and [46]. The need for the platform arose because commercial products did not fulfil all of the needs for the system dynamics modelling, especially in one package. Simantics is an open source software, whose development is taken care of by THTH Association of Decentralized Information Management for Industry.

While the Simantics platform provides many of the features found in the Modelling

Factory, deploying models with Modelling Factory could be divided into two building pipelines, of which only one is based on Simantics. By using standard tools for model publication and UI generation, it is possible to have all the same features from the Simantics-based and generic tool pipelines. The main difference between the two can be summarized as follows: The Simantics based model publication pipeline constructs user interfaces utilizing graphical UI generation system specific for Simantics (via so-called Simupedia software component), which requires practically no code writing skills, whereas the generic publication pipeline is free to utilize any modern UI development technology. The Simantics based publication pipeline is currently more developed due to the many legacy software components and simulators developed at VTT, but the situation may change in the near future as there are many modern methods and active user bases for other techniques that can be utilized for the same purposes.

Both model publication pipelines (Simantics-based and generic) of Modelling Factory serve the end user web app through which interaction with the model becomes easy. No software installations are required by the end user, a simple web browser and internet access suffices. Modellers utilizing the Simantics based publication pipeline, need additionally a Simantics Desktop software, through which the web app is built. In addition, Simantics implements a web server that serves a web page containing the UI. The Simantics Desktop offers a variety of user components (widgets) from which the UI can be constructed. For modellers requiring more power, new widgets can be constructed at will. Simantics Desktop features a fully-fledged programming language of its own, Simantics Constraint Language, or SCL for short. SCL is described in more detail below.

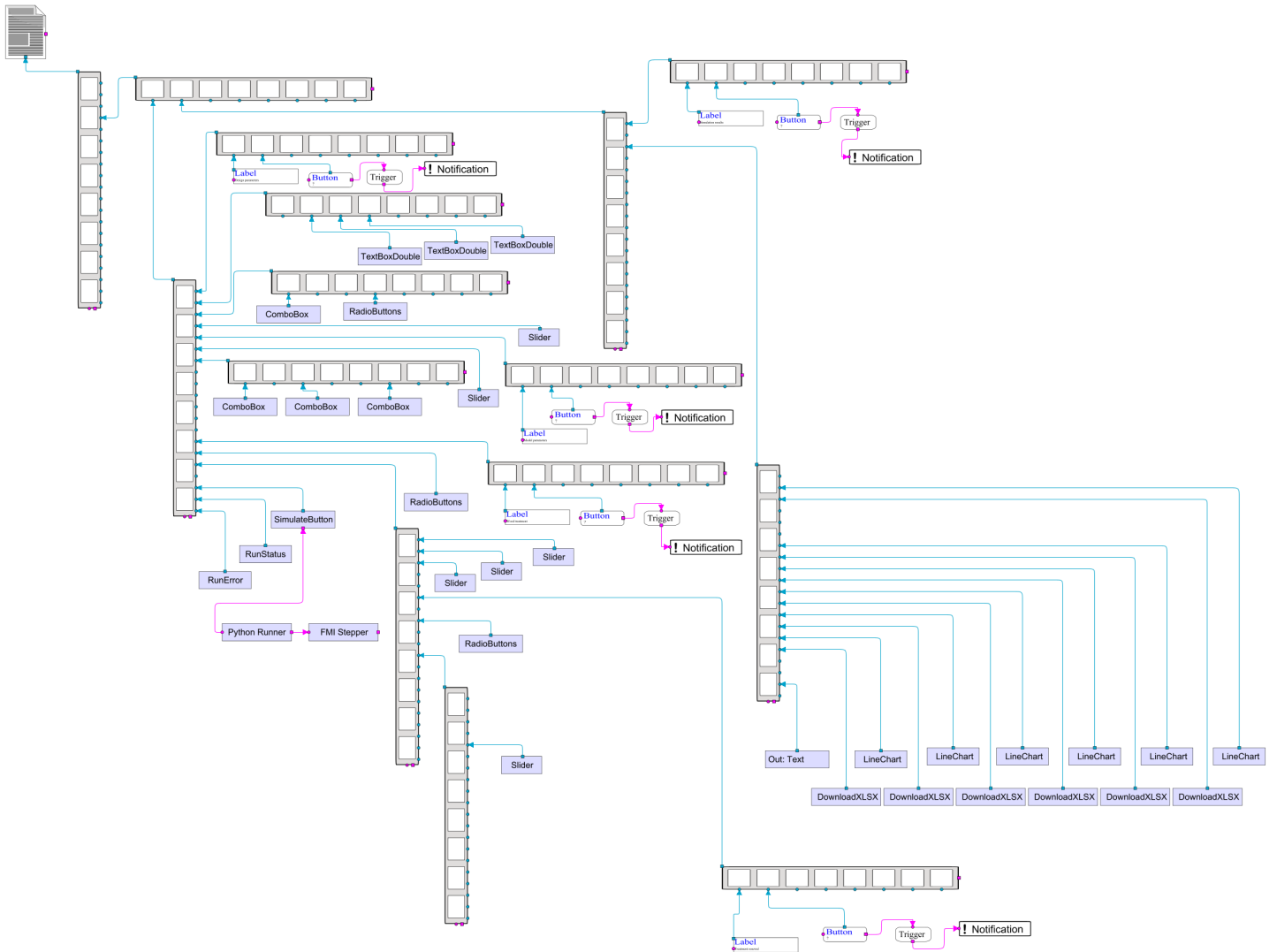


Figure 1: Defining UI with Simantics desktop application

Figure 1 pictures the graphical view in the Simantics desktop application where the user interface, UI for short, can be defined using graphical programming means. Components are represented by icons and boxes and connections between them with lines. The icon representing a page with a folded corner in the upper left corner of the figure is the (empty) web page and the graphical widget representing it is called the root component. Everything connected to it is added to the page and becoming visible to the end user of the application. The layout on the web page, regarding

the position of each component, can be easily defined with a grid that can be made with the horizontal and vertical bars shown in the figure 1. Vertical bars, like the one connected to the root widget (upper left corner), are columns in the grid and horizontal bars are rows. They are analogous to the concept of rows and columns in CSS Flexbox. In figure 1, a column is attached to the root page, and a row is placed inside that column. There are two columns next to each other, one for inputs and one for outputs. A picture of the UI is in the next section, figure 3.

The ready-made components (widgets) include, for example, a Python runner and an FMI stepper, which already adds the ability to run any arbitrary Python code and work with imported Functional Mockup Units. There are ready-made components for various types of graphical outputs and data visualisation needs as well. For time series data, there are graph components, and discrete values can easily be presented with textual output based components. All of these include easy labelling and automatic updating.

The Simantics platform uses its own language for scripting in the environment, called Simantics Constraint Language or SCL. The language is a dialect of Haskell, but it differs in its evaluation strategy, which is strict in SCL, and the different standard libraries. Syntactically, SCL is still so close to Haskell that Haskell tutorials are still useful in learning the language. SCL can be extended by importing the lacked functionality from other languages: in particular Java code can be easily used by declaring them in an *importJava* block.

3.2 The Role of Modelling Factory

The Modelling Factory platform is an integration and publishing platform for generic computer models that serve static or dynamic simulation applications. The integration capabilities are enabled by (1) the underlying Simantics data representation capabilities (e.g. the specialised semantic database) and (2) co-simulation capabilities brought about by the FMI standard for interoperability. The publishing features supported by Modelling Factory refer to the possibility of publishing a graphical

user interface (GUI) or an application programming interface (API) for the human-simulation model or machine-to-machine interactions with the simulation models, respectively. Modelling Factory also enables the publication of the simulation codes and code libraries as Gitlab repositories enabling standard developer collaboration related to code-level development. Importantly, collaborative development of the models can be taken further, allowing high-level end-users to test and communicate their ideas to the model developers enabled to produce the next improved version owing to the continuous deployment pipelines of the Gitlab environment.

Modelling Factory usage employs three different major user groups. At the 'bottom' are the developers with more profound coding skills needed to maintain and update the platform's code base. The second group includes the model developers (which, among themselves, can be divided into different subcategories). In general, model developers program the models using their domain-specific languages and environments. For example, a biotech expert is likely to prefer an R-language-based model, whereas a government employee making workforce predictions may prefer an Excel program. Modelling Factory aims to help the publication of GUIs for these and other types of models through which also third parties can use them in addition to the original developers. Moreover, the Modelling Factory platform supports the integration of the R mentioned above and Excel models as well as a wide variety of other models without enforcing a restriction to a particular type of modelling, platform or integration paradigm *ab initio*. Naturally, not all paradigms or languages can be supported at the same level. Nevertheless, the degree of generalizability and agnosticism concerning specific technology preferences separates Modelling Factory as a ModelOps environment from most commercial and OS competitors. While Modelling Factory does not support all possible available technologies (no solution does), it makes a serious effort to help different level model makers and programmers to publish and integrate their work into a bigger trunk of the model base, which does not compel the user to restrict herself to some specific closed 'programming paradigm universe'.

Modelling Factory is not meant to be a supercomputing or data management

platform. Supercomputing is meant for computationally hefty models run on dedicated hardware using an extremely parallelised architecture with, for example, many Graphics Processing Units (GPUs). Modelling Factory, in contrast, mainly supports (computationally lightweight) fast-to-compute models, which are suitable to serve as computational engines for web applications. Human users of web apps do not typically want to wait for more than a few seconds to get the simulated answers to their questions. For this reason, Modelling Factory provides several templates which enable the publication of GUIs for machine learning (ML) models and other types of surrogates. Surrogate models are replacements for the original, accurate, slow-to-compute models. Surrogates are almost as accurate as the original precise simulation models but much faster to run and, therefore, ideal for GUI interaction or optimisation runs, where several surrogates run in succession.

In summary, while Modelling Factory can technically serve slow and accurate simulation purposes, it is better to let this type of simulation for supercomputing and other specialised clusters. For example, many problems involving materials science applications require computationally intensive multiscale simulations, which are better left for dedicated scientific computation frameworks to handle. Instead, the fast-to-compute surrogate versions of these types of multiscale models produced, e.g. by ML methods, are ideal inputs for Modelling Factory workflows. Due to their fastness, they can even be integrated with a higher-level decision and design model workflows utilising material modelling inputs to determine, for example, material durability effects on ecological and economic performance of product on larger time scales.

The fast-to-compute property of the models enables Modelling Factory to act as an integration and collaboration platform, which utilises fundamental processes and material information to feed system-level models aimed to answer large-scale ecologic and economic questions by end users of the web apps. Most web apps currently published on Modelling Factory and their underlying fast computational models address design questions related to circular economy design, sustainability, and such domain-specific questions.

As such, the Modelling Factory is an entirely general integration and publishing platform and can provide model-based answers, which require integrated knowledge from many different domains. It is not domain or language-specific, but certain prioritising choices have been made to support more commonly acquired features as specified below.

3.3 Features

Modelling Factory is designed to support system-level modelling and models, such as fast surrogates, whose inputs can be utilized in system-level modelling. The main characteristic features supported by Modelling Factory can be summarized as follows:

- Easy model code, web UI and API publishing
- Co-simulation with different models
- Easy model management and version control system for model developing
- Scalability
- Easy interfacing/interaction with published models (GUI/API)

These features are broad and could be attributed to a system designed to support different kinds of models as well. How the features are implemented depends on the exact domain the platform is designed to operate on.

The driving factor behind the creation of Modelling Factory is the difficulty and the lack of generic model agnostic software to publish the model's code, GUI and API in one go. For the software to be helpful in model publishing and not be just another model-specific publishing system, it needs to be model agnostic. As outlined in the previous section, the problem is that most of the current publishing systems are not made for using or running the models in a fast-to-deploy model-agnostic environment. One example of this is TensorFlow Hub. Models can be saved and shared for everyone to see and use, but that requires loading them from Tensorflow Hub and writing more code that utilises the model for calculations. This has the

upside of making extendability easy, but it requires coding knowledge and knowing how the model inputs and outputs work. It does not make the results and quick reiteration with different inputs available to the average user.

In many cases, it would be advantageous to have the ability to use different kinds of models working together to achieve the final result. One example would be using a static model, such as one created with the eco-impact evaluation model Sulca [47], which can provide pre-computed initial values for a dynamic simulation model, e.g. a design model for cleaner mass transport, or enhanced life-cycle assessment, which requires product life-time estimation. For many commercial tools, this kind of free combination (data integration) is not automatically guaranteed as interoperability remains an issue not only on the level of data but also in other respects. To remedy this fact is one goal of the Modelling Factory. This can be either achieved via the semantic data integration capabilities of the underlying Simantics platform or addressed on a higher level of published models as functional mockup units by using the possibility of making different models communicate with each other through the FMI standard protocol.

In most cases, the code in software development projects is organised using a distributed versioning system, mainly Git. This is usually seen as a good coding practice. Git is a version control system, which is ubiquitously employed in code development, and it does not make sense for Modelling Factory to veer from the established practice. Hence the same Git-based system is used in Modelling Factory to organise and manage code. More specifically, Modelling Factory uses the GitLab platform that provides version control and DevOps -features. Using an existing platform already featuring easy code-level collaboration practices allows Modelling Factory to implement the same features. For some projects, the additional difficulty for cooperation is created by the fact that modellers occasionally use binary files, which cannot be directly edited or merged through Git, as part of their project. Using AproS process modelling software [48] as an example, the AproS modelling software saves the models as .aprosz files. These files contain no easily user-viewable data; they must be opened with the AproS software. Consequently, they cannot be

merged directly via standard GitLab code-handling procedures but must be handled manually using the software that created them. Of course, many models are saved in a text-based format, so Git should have no trouble merging the two files. This can sometimes lead to a broken model if the two versions are not merged correctly, though that should happen only rarely.

Starting to use the Modelling Factory should be straightforward and familiar to many code developers since the technologies at the core of the Modelling Factory are already widely used in the industry. The following subsection will go into more detail about the technologies, but these are all very familiar to any DevOps engineer, and every software developer has at least heard of them. This is crucial for the easy use of model management and version control systems. The well-known fact is that if the procedures are not familiar to the potential contributors to a project, they will not be utilised to their full potential and are less likely to be adopted.

Scalability is an essential feature for any system on a stable growth path. The provider of a cloud service of any software platform must take care of this aspect, as well, to be successful. This solution can become extremely expensive quickly, if not paid proper attention to from the start. Cloud providers often run the service either in a virtual machine or as a container in a computing cluster. With the freely available software setting up a cluster for hosting containers is relatively straightforward for smaller deployments. Kubernetes, to name one, is an open-source container orchestrator that can be easily deployed to run a cluster of containers. This is also what Modelling Factory does. The architecture allows for high-demand deployments to have multiple pods serving the same application. The number of containers in pods can easily be adjusted also on demand to scale with the additional demand when needed.

As one of the essential functions of the Modelling Factory is to bridge the gap between the modeller and the decision-maker or any end user interested in the model. The end user needs to be able to give inputs to the model, run it and get outputs without having extensive knowledge about the technologies used to implement the

model. On Modelling Factory, this requirement has been taken care of by allowing access to the published model via standard web browser or well-documented REST API (Representational State Transfer API). The UI will be run in a web browser, and it can be created by either using the Simantics Platform or any other popular technology for creating web-based user interfaces.

3.4 Technologies Used

Before examining the structure of the Modelling Factory, the most critical technologies involved will be presented in this subsection.

GitLab is a DevOps platform that offers many useful features, of which the most important and visible component are the code repositories. The repositories can be created and managed either by using the web-based UI of a GitLab deployment or Git. Two features make GitLab especially suitable to be used as a code management tool for the Modelling Factory are that it is possible to easily make private deployments of the GitLab platform and the focus on easy and functional CI/CD pipeline (Continuous Integration, Continuous Deployment) configurations. The CI/CD pipeline is a series of steps to build and deploy an application. In GitLab, this is controlled by a `.gitlab-ci.yml` file in the repository's root. When a modeller makes a new model version, the CI/CD pipeline runs tests, builds, and deploys the model. When the template-repository of Modelling Factory is used to create a repository for the model and deployment to the Kubernetes cluster, minimal coding is needed since the template already contains ready-made definitions for the deployment; only a few parameters need to be tweaked.

Docker Inc. is a company that created Docker containers and maintains the product. Docker containers today are an essential technology for packaging and delivering applications to people quickly and efficiently. While more technologies doing the same thing exist, such as LXC, rkt, and Windows Containers, Docker has become almost universally the de-facto standard for containerisation.

Traditionally, software was written as an application binary copied to a computer

and an operating system used to run that specific application. Every application shares the same operating system and the same runtime environment. Different applications have different requirements from the host OS, and sometimes these requirements conflict. Thus it can be complicated to build an environment that could accommodate all of the software.

Containers aim to solve this by packaging every application into a container image package. That container image provides an isolated environment separated from all the other software which, in turn, provides all the requirements for the packaged application. The container image then becomes a container at runtime. The container images are lightweight packages that only provide the necessities to run the application. A container runtime is still needed to run a container image. For Docker container images, this means Docker Engine, a software that runs on a host operating system as a traditional application. The job of the Docker Engine is to provide the Docker containers access to hardware, network, and disk resources where needed. [49]

Containers differ from virtual machines by not offering a full operating system for each application. This reduces the overhead needed for the application isolation compared to the virtual machine. [49]

Docker containers were chosen to be used in the Modelling Factory due to their status as the industry standard and due to the widespread support for them among many different systems. Running Docker containers in Kubernetes clusters is a mechanism for running many applications that have been used by many and thus is also a well-tested technology. Additionally, building Docker images out of the application code is an easy-to-implement process in the GitLab environment. The CI/CD pipeline can build the images with minimal configuration required per image basis.

Running the container images on different servers is a challenge. The solution used with Modelling Factory is to run a Kubernetes cluster, where each application is run in a distinct pod. Kubernetes is an open-source container orchestrator that is

very popular in the industry with years of testing by many organisations.

3.5 The Structure of Modelling Factory

Figure 2 shows how the Modelling Factory software pieces combine. Repositories with the code are hosted on GitLab, from where the CI/CD pipeline will deploy them to the Kubernetes cluster. The deployment of Modelling Factory does not use the public GitLab instance available in the address gitlab.org but instead uses another instance. The Kubernetes cluster and the GitLab are hosted on virtual machines in the DMZ server farm, along with a self-hosted private registry. This Docker image registry is used for storing and pulling images from which the Docker containers are built. The image contains all the necessary files and interpreters to run SCL-code, host a web-based UI, and import and run FMUs and other models. There is a possibility to scale the Kubernetes cluster to run in multiple servers to accommodate more considerable demand.

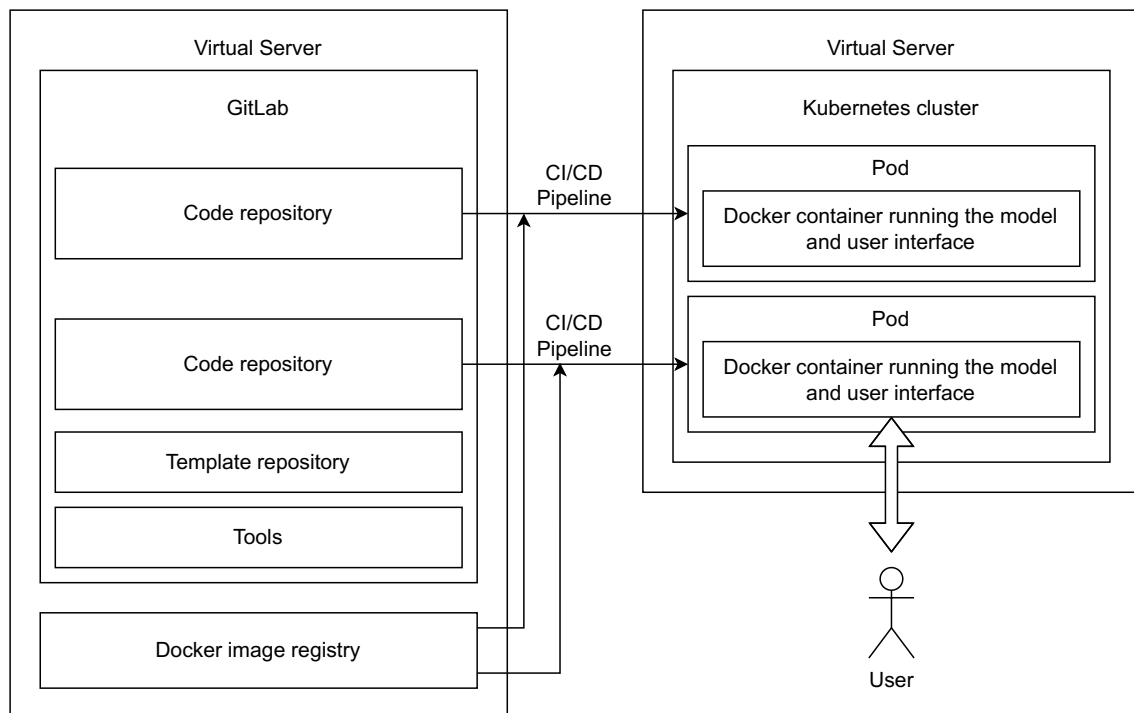


Figure 2: Modelling Factory structure

The technical assembly of Modelling Factory is flexible enough to serve different kinds of business models. Currently, the 1.0 version is reachable at web address <https://modellingfactory.org/> and the 2.0 version is according to the present plans addressable by changing the .org to .com. This naming convention becomes effective once the Modelling Factory 2.0 has been set up on the DMZ server farm as mentioned above.

The template repository in GitLab, also shown in figure 2, can be used as a base for starting a project intended for Modelling Factory deployment. GitLab offers a way to instantiate a new repository based on another repository by pressing a few buttons in the user interface. This template repository contains a ready configuration for deploying models and a user interface defined with the Simantics Platform. In this case, no additional configuration is needed.

The template repository contains the following main components:

- `fmus`
- `models`
- `sharedlibs`
- `.gitlab-ci.yml`
- `Dockerfile`
- `HeadlessServerMain.scl`

The first three, "fmus", "models", and "sharedlibs", are folders that contain the data for the model and the UI. More specifically, the "fmus"-folder is meant to contain the models in the Functional Mockup Unit format detailed in the previous section. These FMUs are imported by the Simantics platform into the workspace and are then available for use by the Simantics platform and UI of the specific model or the API built for the model in question. Section 3.1 contains explanation for how the Simantics platform can be used to create a model UI to interact with it.

The directory "models" is meant for models not in the FMU format. This folder can be used to store, for example, raw Sulca or Apros models in the container if these cannot be imported to the Simantics platform. Then these models can, either way, be used with a separate headless solver running in the same pod.

The sharedlibs folder contains the definitions for the UI as well as all of the supporting libraries. These are all saved in .sharedLibrary format that can be directly imported and used by the Simantics platform.

The .gitlab-ci.yml -file is the standard for defining a CI/CD pipeline in a GitLab deployment. The file contains the definitions for the variables to be used during the deployment, as well as a new job to build the docker container and two commands to create necessary Kubernetes secrets for docker registry and credentials and deploy the created container with the helm chart. The build-docker job uses a kaniko executor to build the container according to specifications in the Dockerfile. The variables in the file are either inherited from the group or automatically defined for the repository by GitLab. Thus, it is unnecessary to modify this file for the pipeline to work as intended.

The Dockerfile in the template is simple and only specifies the image to be pulled from the Modelling Factory GitLab-instance and copies the contents of the folders and the HeadlessServerMain.scl file to the container. This file can be easily edited to include other necessary imports to the environment, such as copying a requirements file for pip and instructing pip to install dependencies so that Python scripts running in the container can find the libraries.

The yet unexplained file HeadlessServerMain.scl is an SCL script run in the container. It is responsible for importing all shared libraries, models and FMUs to the workspace. When the container is constructed, the directories fmus, models and sharedlibs are copied under the /tmp/-directory in the container. HeadlessServerMain.scl imports everything in these three folders to the workspace.

4 Evaluation of Modelling Factory

The previous section described how Modelling Factory works and how it should be constructed. This section evaluates how well Modelling Factory works and fulfils the goals it was developed to answer as a ModelOps platform. First, the functioning and usage of Modelling Factory are looked at through two different use cases. These use cases exemplify how the platform acts as a model integration framework and how its published UIs look and function. It should be noted that the examples discussed in this chapter also demonstrate how the co-simulation of different modelling paradigm-based models (e.g. static and dynamic or different language-based models) can be brought to serve through a single user interface, which can be continuously improved very quickly. At the end of the section, the findings from these use cases are summarised.

4.1 Use Case 1 - Wood Product Process

This use case is meant to model a wood production process from raw planed timber to a more refined form that is optionally given a user-defined product treatment.

The use case combines two kinds of modelling paradigms to get an answer to an end user. The values are fed into a python script that calculates, with the help of Numpy-library, different key characteristics as a function of time. These characteristics include mass loss, strength, and values associated with certain wood products' wood treatment or protection characteristics. Additionally, the application uses the input values to calculate how much greenhouse gas (GHG) emissions in CO₂-equivalent would the process produces during its lifetime. For this life cycle assessment (LCA), a Sulca model (LCA software of VTT) is used. The Sulca model is imported to the workspace as a functional mockup unit.

The inputs are given, and outputs are presented on a user interface available as a web page in a web browser shown in figure 3. This UI uses the exact mechanism available at every model deployment built with the Simantics-based publication pipeline in Modelling Factory. The general manner in which Simantics-based UIs are

constructed has been depicted in figure 1.

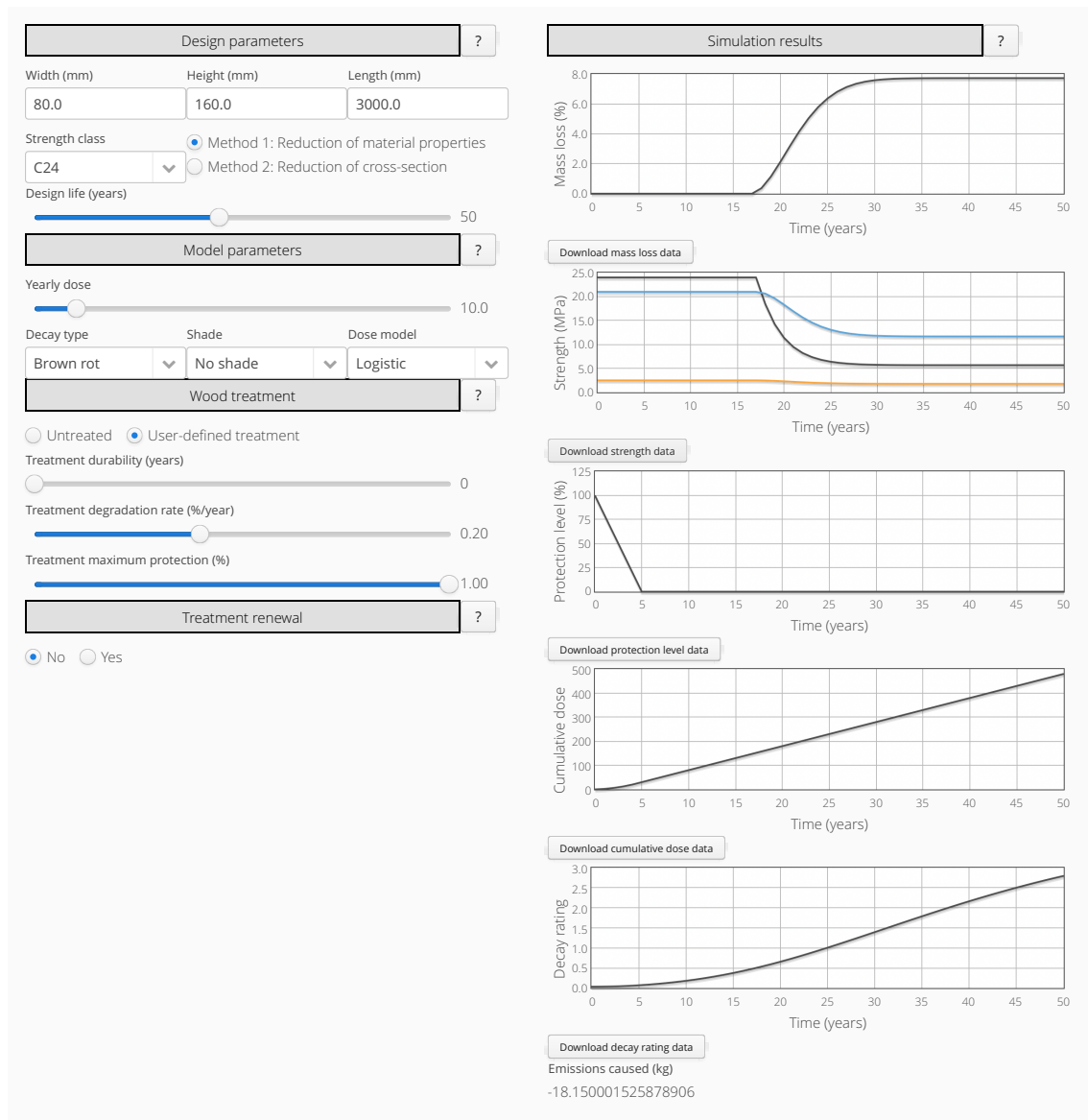


Figure 3: The UI of Use Case 1 rendered in a web browser.

Figure 3 shows what the user interface published by the modeller on the Modelling Factory platform looks like when rendered in the web browser. The inputs can be given on the left using a combination of text boxes, sliders, and dropdown menus. On the right, the simulated results are presented. Since calculating the results takes only a fraction of a second (fast-to-compute models at the core of Modelling Factory

publication protocol), the results can be updated as soon as one of the inputs is modified. When simulating with untreated wood, only the mass loss and strength are shown, but more results are shown as soon as a specific treatment is chosen, as well as the appropriate inputs to define its characteristics.

The case is built with Simantics desktop software which provides easy capabilities for co-simulation and a ready-made publication pipeline on the Modelling Factory platform. In figure 1, the component named "Python runner" is visible. This component maps the variables from the Simantics namespace to the Python namespace. The variables and how to map them are given as parameters to this component, as well as what is the starting point of the Python script to run. In this example, the Python code calculates most of the outputs. The Python code calculates all the time series data while the life cycle assessment is done by Sulca software imported as a functional mockup unit into the workspace. The universality of Simantics for simulation model integration makes it easy to use FMUs as part of the integrated calculations. It is noteworthy that even though the Python script is run as is in this case, it can also be translated to an FMU and then imported similarly to any other FMU.

In general, the FMUs are used similarly to the Python code in this example. The component "FMI Stepper" attached to the Python runner in figure 1 is used to map the variables from the Simantics namespace to the exposed endpoints in the FMU and vice versa. The FMI stepper also initialises the FMU and steps it one step forward. In this case, that is all that is needed since the LCA model is static, and the results do not change as a function of time. Dynamic simulation models that change state over time can also be easily used with the "FMI runner" -component, but in this case, the stepper is more appropriate since a static model is used. Both of these components do the same variable mapping.

Using an FMU with the Simantics platform requires the user to know which variable to map to which endpoint. Unfortunately in the case of Sulca, they are not always very explicitly named.

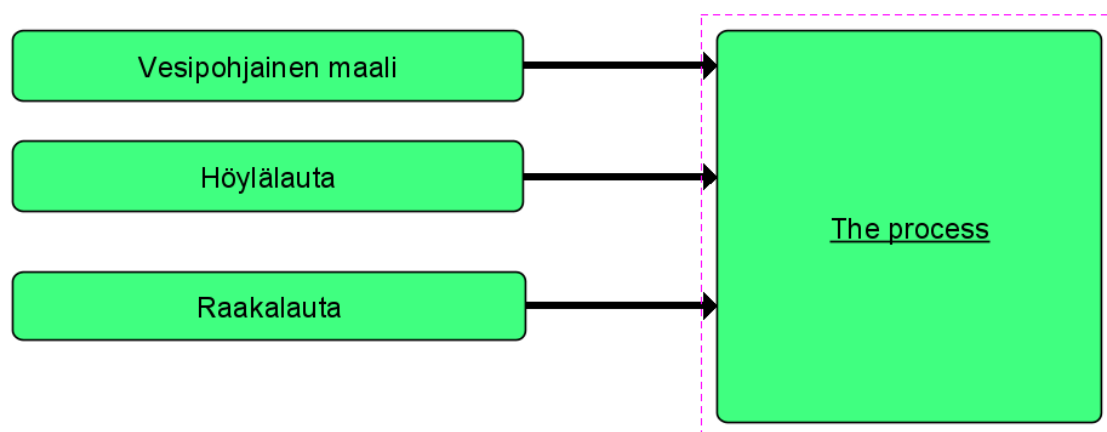


Figure 4: Visual representation of the Sulca process model for Use Case 1.

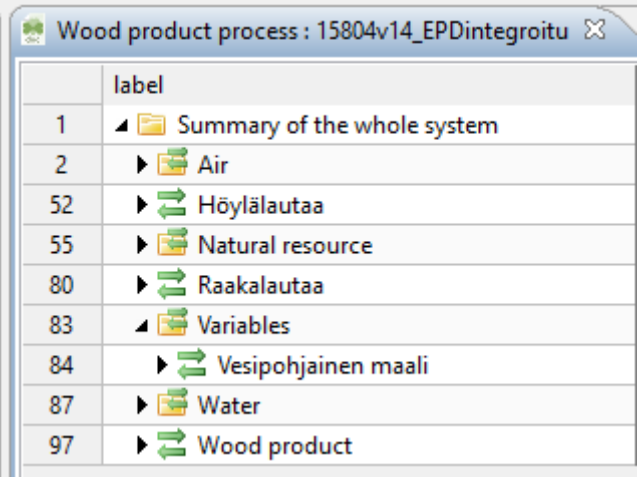
General						Exchanges						DataSheet						Equations						Parameters						Junctions						Impact Assessment					
Input Variable												Output Variable																													
Amount	Calculated	Unit	Quantity	Amount	Calculated	Unit	Quantity	Amount	Calculated	Unit	Quantity	Amount	Calculated	Unit	Quantity	Amount	Calculated	Unit	Quantity	Amount	Calculated	Unit	Quantity																		
1 Variables												1 Variables																													
2 Vesipohjainen maali												2 Wood product																													
0,1000	0,1000	m ³	Volume	c	2,500	2,500	m ³	Volume	c																																
3 Variables																																									
4 Höylälautaa																																									
2,500	2,500	m ³	Volume	c																																					
5 Raakalautaa																																									
0,000	0,000	m ³	Volume	c																																					

Figure 5: The variables defined for the node ‘The process’.

The following figures present how the variables are mapped from Sulca to the bindings in the FMU. In the GitLab code repository, the variables for the nodes are under the folder *Wood_product_process* and *c*. The upper folder name ‘Wood_product_process’ comes from the flowsheet name in Sulca; these are the same. The folder structure is presented in figures 7 and 8. Figure 6 shows all the outputs Sulca and the FMU calculate. These are mapped to the *Variables* directory shown in figure 7. All directories contain variables named *ia[]* with index. In Sulca, a choice of environmental calculation variables can be made. In the choice made for this model, there are 28 different environmental impact variables mapped to *ia[]* variables in the FMU in order.

4.2 Use Case 2 - CORTOOLS

CORTOOLS, known by its full name as Co-creation of corrosion and monitoring tools, is a software designed to simulate the corrosion of different steel types by



	label
1	Summary of the whole system
2	Air
52	Höylälautaa
55	Natural resource
80	Raakalautaa
83	Variables
84	Vesipohjainen maali
87	Water
97	Wood product

Figure 6: Output variables of the FMU as shown in Sulca software.

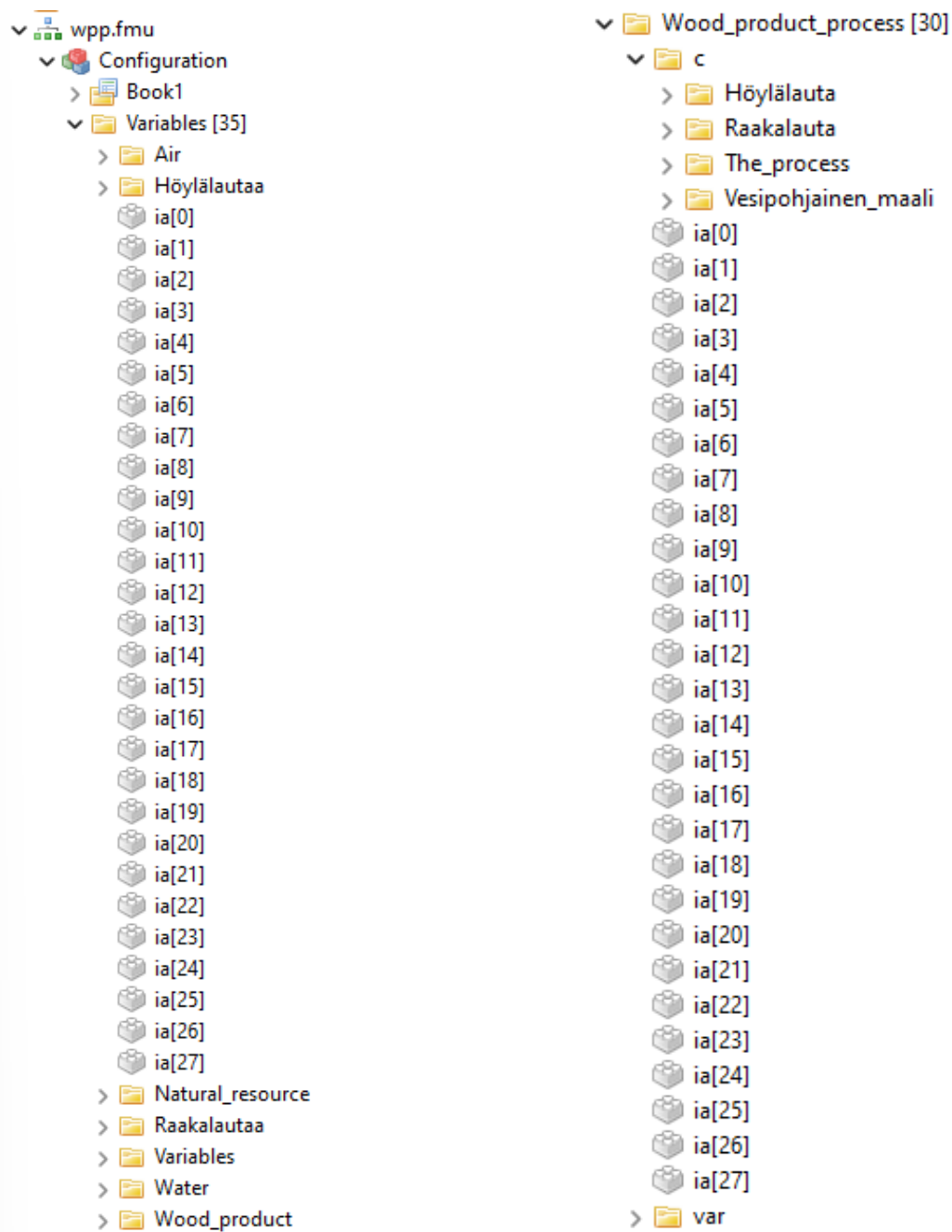


Figure 7: Output variables in Simantics desktop. Figure 8: Different variables for the nodes in Simantics desktop.

different kinds of water-based solutions.

To calculate the results, CORTOOLS uses a solver written in Python. The solver utilises HSC Chemistry software by Outokumpu to calculate chemical activities as needed by the solver and Numpy-library to accelerate the calculations. Since HSC Chemistry is a standalone Windows application, in the initial implementation of CORTOOLS, it was called to do the calculations through the Component Object Model -interface (COM). The values would then be returned with the same interface to the Python solver, which would display them in a user interface created with Simantics Desktop. The same UI is also used to define the user-configurable values for the solver. The user interface is shown below, in figure 9.

Analysis type

Pitting corrosion and pit nucleation analysis ▼

1.4162 ▼

Species amount (mol)

H2O	H(+a)	HSO4(-a)	SO4(-2a)	Cl(-a)
55.50622	1.87804	0.0	0.93902	0.174894

Species amount (mol)

Na(+a)	Fe(+3a)	Li(+a)	Al(+3a)	HPO4(-2a)
0.661157	0.100278	1.30961	0.500334	0.115273

Species amount (mol)

Mn(+2a)	Co(+2a)	Ni(+2a)	Cu(+2a)
0.32763	0.319023	0.48901	0.176239

Simulate

T, ...	NaCl concentration, mg/L									
	0	1256	2513	3770	5026	6283	7540	8796	10053	11310
27...										
29...										
31...										
33...										
35...										
38...										

Figure 9: The user interface of CORTOOLS.

The CORTOOLS example offers two different kinds of installation possibilities for the customer. The choices vary mostly in what is packaged for the installation and what is expected to be in the environment.

The first option is to deploy a complete software-as-a-service (SaaS) self-contained platform. This is the easiest way for the user since the software contains everything needed to run the solvers and get the solutions, including a ready-made user interface.

The second option is to use the Functional Mockup Interface standard and convert every simulation model to a Functional Mockup Unit that is used through an Application Programming Interface (API) offered by the Modelling Factory platform. Currently, the CORTOOLS project includes a ready-implemented REST API, which enables the end user to programmatically provide the same inputs as depicted in figure 9 and collect the results without human interaction for much faster processing in the end user's simulators. What constitutes an API to be RESTful or fulfil the requirements is not discussed in this Master's Thesis. More generally, it suffices to say that the end user can interact with the application via a human-accessible GUI or machine-to-machine route by employing the API. It is also possible to either include the databases needed for simulation models either as 'internal' software components of the Modelling Factory platform, which must be accessed through the GUI/API or as 'external' components (hosted on third-party servers irrespective of Modelling Factory), which are called by the simulation models stored in Modelling Factory repositories. The latter case requires a clearly defined public API for the database to be effective.

To deploy CORTOOLS with Modelling Factory, a small extra service is needed. HSC Chemistry works only on Windows and thus can not be added to a Linux-based Docker container. A Windows server core base image can be used to dockerise a Windows application that can be installed unattended and run in a headless mode, meaning without a user interface. Another option is to create an entire Windows virtual machine for running the application. In the same Windows-based environment that HSC Chemistry is run in, a small HTTP server needs to be run, too, to enable

communication between the Windows-based software component and the Linux cluster installation of Modelling Factory. The calculations with HSC Chemistry are done by calling the HTTP server and specifying the desired method and values. The server then calls HSC Chemistry through the COM interface and returns the response results.

4.3 Summary

All in all, Modelling Factory provides a very versatile environment to quickly deploy computer models, both static and dynamic simulation models. As long as the simulation model can be transformed into a Functional Mockup Unit, it is guaranteed to work with the Modelling Factory. The modeller is, however, not confined to this approach but may instead choose to implement a user interface for interacting with the model using almost any standard technology for UI creation without being restricted to Simantics-based UI publication methods.

However, not every approach to creating models is compatible with the use philosophy of Modelling Factory. As stated in section 3, Modelling Factory is not meant for extremely large or computationally demanding models, instead focusing on lighter models that are faster to evaluate. Even when a model would be an excellent fit to be used with Modelling Factory, if it is not self-contained but instead relies on services requiring, e.g. Windows applications, or applications specific to the user's computer, the deployment is more challenging. Naturally, Windows applications could also be integrated in the future in a more organised manner, but the current Modelling Factory platform set-up relies on a Linux cluster, which is more economical to operate in cloud environments. The Windows application communications (e.g. HSC in Use Case 2) are currently taken care of by an HTTP-server application.

5 Comparison With State-of-the-Art

To properly evaluate Modelling Factory, it is crucial to compare it with other solutions that could be used instead of Modelling Factory. Looking at the available state-of-the-art solutions, it is evident that most are domain specific. This makes the comparison of Modelling Factory to other platforms complicated since Modelling Factory aims to be as generic as possible. It is generally difficult to compare a domain-specific tool with a generic one. A specialised platform will always beat a generic one in the domain the specific one is designed for when a specialised platform can hardly do anything in other areas, and the comparison is meaningless. Overall, it seems fair to conclude that there is no other software platform that competes with Modelling Factory in its generality. Nevertheless, this section compares the Modelling Factory with state-of-the-art solutions presented in section 2.

The following table compares different features of the Modelling Factory and state-of-the-art solutions.

	Modelling Factory	Simulink	Insight Maker	RStudio and R Markdown	Excel environment	VIMMP
Model deploying capability	✓	✓	✓	✓	✓	
Collaboration capability	✓		✓		✓	✓
Co-simulation capability	✓	✓		✓*		
Works with third-party tools	✓	✓		✓	✓	✓
Accessible model sharing	✓	✓	✓	✓	✓	✓
Still actively developed	✓	✓	✓	✓	✓	
Interoperability with other software	✓					
Support for modelling		✓	✓	✓	✓	

Table 2: Properties of select commercial software.

*) RStudio and R Markdown enable the user to include many languages in one notebook. While there is no good way to import an FMU in R, it can be imported using, for example, Python. This, however, means using multiple languages in the notebook, further complicating it.

As discussed earlier, ModelOps platforms focus on the operative features rather than supporting the development of the model. In table 2, the first row of the column measures this: does the solution offer meaningful features for deploying the model code to be accessible? Collaboration capability measures whether there is a meaningful capacity for collaborating in the model and UI development. On the other hand, co-simulation capability describes if the final model can consist of multiple different models running in co-simulation mode. FMI for co-simulation is an example of this. Accessible model sharing is a measure of whether the model and UI can

be shared with other people and if there is visibility control for the running model. Still actively developed in this context means that the product is still ongoing and has new features being developed. Finally, support for modelling means that the solution offers meaningful tools to support the modeller in creating the model. In other words, is the platform fully ModelOps platform, or does it exhibit computer modelling and simulation features as well?

To reiterate, what Modelling Factory tries to first and foremost be is a platform to make deploying and interacting with all kinds of models as easy and seamless as possible. The modeller can get the model used by the relevant people as easily as with a commit to a Git repository. This is often accomplished by pressing a few buttons or providing a few commands on a command line. It should be quick and easy to create a user interface for published model code that can be viewed in a web browser and used for interacting with the model, giving inputs and reading outputs. The creation of the model is often a collaborative effort, and Modelling Factory makes collaboration possible through version control systems, mainly Git.

On the surface level, Simulink seems to be doing almost the same thing as Modelling Factory with near feature parity and thus presents a close competitor. However, in reality, the two platforms are focused on different areas. As stated, Modelling Factory focuses on the deployment of and interaction with the model, while Simulink is focused on creating the models and running them directly on different hardware. In contrast, Modelling Factory runs models in a virtualised environment. Simulink is still mainly a computer modelling and simulation platform, whereas Modelling Factory is a generic ModelOps platform which strives for a modelling environment and language agnosticism.

Functional Mockup Interface was not included in table 2 since it is not a platform but an interface definition. Therefore it is not meaningful to compare FMI to Modelling Factory. Functional Mockup Interface is, however, essential for the industry and the standard way for models created with different techniques to interoperate. As such, FMI is the chosen way to enable co-simulation and model exchange with

many platforms. This is as true for Modelling Factory as it is for Simulink.

As presented in section 2, TensorFlow Hub is a repository meant to help share machine learning models created with the TensorFlow library. The most significant difference between TensorFlow Hub and Modelling Factory is that with TensorFlow Hub, it is currently impossible to use the models without writing a new piece of code to interface with them. For decision-makers and users of the model, this is usually not possible and would require the modeller to write the code for interfacing. This is still possible, and the code can be similarly shared through version control systems that Modelling Factory does. However, without additional effort to deploy the user interface, it will not be usable or even visible to external users scattered around the Internet. With TensorFlow Hub, sharing a model for the worldwide audience is still easier since that is the default mode for TFH. In contrast, the default mode for Modelling Factory is to show the file belonging to a specific project group. The group can be opened to the entire Internet audience if necessary. In other words, sharing the model and associated files with everyone is possible by making the repository public. Modelling Factory does allow a lot more granular access control with the built-in systems of GitLab than TFH allows.

Still, TensorFlow Hub is only meant for models using TensorFlow and cannot be used for any other models. At the same time, Modelling Factory makes it possible to publish TensorFlow models and other kinds of models. While TensorFlow Hub is straightforward to use with its intended purpose, Modelling Factory is much more generic and offers a hugely wider selection of possibilities. Code utilising the pre-trained ML models can be published with Modelling Factory, so these two are not mutually exclusive.

Conceptually, VIMMP is closer to Modelling Factory than other state-of-the-art solutions listed here. The goals for VIMMP and Modelling Factory are somewhat different in that VIMMP aims to connect clients who have a scientific problem with the experts who can help to solve that problem with modelling and offer the experts as much data as possible.

For interfacing with the completed models, the Virtual Materials Market Place is not readily well suitable, and it does not, in fact, offer facilities for creating and presenting a UI to the user.

Open Simulation Platform is a promising tool to enable co-simulation between different models, but it is limited in the domain. Both OSP and Modelling Factory use the same technology or standard to enable co-simulation, namely FMI. In addition to co-simulation, Modelling Factory offers help with user interface creation, unlike OSP. Suppose the problem domain is located in the marine industry. In that case, OSP can also be used to enable co-simulation if desired, and Modelling Factory can be used for deploying and interfacing with the model.

The pattern with OSP and TFH also applies to many other services where one can export a created model to a Functional Mockup Unit. In that case, the Modelling Factory can always deploy and add a human interaction interface (GUI) or machine-to-machine interface (API) to the model. The strength of Modelling Factory comes from the fact that it is general enough to work with many third-party tools and platforms. Some applications, such as Insight Maker, do not offer FMU export or other export capabilities, so it is difficult for any solution to interface with these types of applications.

6 Discussion

Computer modelling and simulation has been a valuable tool in many industries across the board. There is, however, a disconnect between modellers creating the models and end-users interpreting the results of the simulations and applying them in practice. The decision-makers are often not adept enough at modelling, or familiar with modelling tools, which are used in model creation. In practice, modellers often need to rerun the models with different inputs for testing. Modelling Factory was created to solve this problem by making the user interface publication as easy as possible and connecting the modellers and the decision-makers by allowing the latter an immediate testing possibility of the model via a graphical user interface, which can evolve with the different model versions. As some commercial and non-commercial systems already offer this mode of interaction, Modelling Factory has been further developed to support modellers to interface their domain-specific simulation models to answer more demanding system-level questions relevant to high-level decision-makers and designers. While multiple projects offer services similar to Modelling Factory, they are, in most cases, modelling and simulation platforms that only offer the features to be used with models created with a specialized platform, which is typically not programming environment or language agnostic.

Modelling Factory is an efficient tool and generic enough to help deploy the model in many cases. It has not been designed to be used in all cases, such as with models handling large amounts of data or requiring much computational power or specialised hardware. Use case 1 demonstrates a case where Modelling Factory is very helpful and what it is intended to do: to aid higher level decision-makers, who are interested in design questions combining material performance and ecological footprint evaluation simultaneously. The case combines two kinds of models: a Life Cycle Assessment model created with Sulca and a solver written with Python to calculate time-series data. The calculations are relatively light, and through the user interface, quick changing of parameters is possible, and the results are immediately available via the graphical user interface. The user interface is created in this case

using the Simantics Platform-related specialised UI publishing tools, which means that creating it did not need a lot of work writing code because the GUI can be produced via graphical programming methods available for non-programmers. In the future, the Modelling Factory platform will serve both Simantics-based UI/API publication methods and Simantics-independent UI publication methods, servicing an even larger programmer base familiar with most modern web programming methods. Using the Simantics Platform is one of the easiest ways to create a user interface for running the models, but any other method for creating the user interface works as well, as long as the UI can be run and accessed through a web browser.

Use Case 2 demonstrates a situation where the simulation model published on Modelling Factory depends on an external program requiring Windows-based services. This creates a challenge since the HSC Chemistry software used in the case to provide initial values can not be effectively deployed to the same Docker container as the Python-based solver and Simantics-based UI. The answer is to run a small HTTP server on the same virtual machine that HSC Chemistry runs on to act as a translation layer between the COM interface of HSC and the Python-based model running in the container.

The hypothesis for this Thesis was that Modelling Factory can be an effective ModelOps platform. While the use cases summarized above show that the current implementation of Modelling Factory can indeed be an effective ModelOps platform, the restriction to run it as a pure Linux implementation requires the use of an extra communication layer between the Linux cluster and any required external Windows program.

The generality of the Modelling Factory allows for deploying different models and user interfaces. The continued development of Modelling Factory could focus on automatic tools for UI generation from FMUs or for a general way to deploy Windows-based applications as well.

7 Conclusion

The definitions of MLOps and ModelOps are somewhat muddled concepts which are sometimes used interchangeably. One of the reasons for this is the massive surge in the popularity of machine learning models in recent years. The willingness to experiment with different kinds of machine learning models has recently extended beyond the realm of modelling experts. One of the reasons for this is the emergence of models a user can easily interact with, such as ChatGPT by OpenAI. While this trend is also visible among modelling experts with machine learning being applied to an ever-growing number of domains, more traditional models with ordinary differential equations in discrete models or continuous models with different interconnected time-dependent systems are not disappearing. Instead, they hold a firm place, especially in system modelling and digital twin creation. Machine learning models are an additional helpful tool to add to the modellers' toolbox. With the growing amount of different tools, getting them to work together is increasingly important. With more people showing that popularity can be gained through accessibility, ModelOps becomes increasingly crucial for modelling companies and teams to 'get it right'.

For these types of problems, Modelling Factory is an extremely valuable tool. Modelling Factory mainly improves accessibility in two ways. Firstly, deploying a model to be available for a network of users becomes a fast task with ready-made GUI and API publishing pipelines, which require little to no additional configuration per project. Secondly, creating a user interface to bridge the gap between a user without a modelling background and the model is possible with various technologies. With the help of the Simantics platform, it takes only a short time to publish an interactive UI for any model. The features can be expanded by using the Simantics Constraint Language at will. Additionally, one can use standard UI publication technologies with Modelling Factory as well.

References

- [1] J. Carson, “Introduction to modelling and simulation,” in *Proceedings of the Winter Simulation Conference, 2005.*, Dec 2005, pp. 16–23.
- [2] L. Zhang, B. P. Zeigler, and L. Yanjun, *Introduction to Model Engineering for Simulation*, 1st ed. Academic Press, 2019.
- [3] V. Rajlich and K. Bennett, “A staged model for the software life cycle,” *Computer*, vol. 33, no. 7, pp. 66–71, July 2000.
- [4] K. M. Clark, “The use of computer modelling in estimating and managing future catastrophe losses,” *The Geneva Papers on Risk and Insurance - Issues and Practice*, vol. 27, no. 2, pp. 181–195, Apr 2002.
- [5] T. Hill, M. Palmer, and L. Derany, *Ten Things to Know About ModelOps*, 2022.
- [6] H. Schichl, *Models and History of Modelling*. Springer, Boston, MA, 2004, vol. 88, p. 12.
- [7] T. Archibald, C. Fraser, and I. Grattan-Guinness, “The history of differential equations, 1670–1950,” *Oberwolfach Reports*, pp. 2729–2794, 2004.
- [8] D. Goldsman, R. E. Nance, and J. R. Wilson, “A brief history of simulation revisited,” in *Proceedings of the 2010 Winter Simulation Conference*. IEEE, Dec 2010, pp. 567–574. [Online]. Available: <http://ieeexplore.ieee.org/document/5679129/>
- [9] Freiburger, Paul and Swaine, Michael and Hosch, William and Nirala, Satyavrat, “Atanasoff-Berry Computer: Additional Information.” [Online]. Available: <https://www.britannica.com/technology/Atanasoff-Berry-Computer>
- [10] Oak Ridge National Laboratory, “First generation electronic computers,” July 2018. [Online]. Available: <https://web.archive.org/web/20180702192315/https://www.phy.ornl.gov/csep/ov/node10.html>

- [11] S. D. Roberts and D. Pegden, “The history of simulation modeling,” in *2017 Winter Simulation Conference (WSC)*. Las Vegas, NV: IEEE, Dec 2017, p. 308–323. [Online]. Available: <http://ieeexplore.ieee.org/document/8247795/>
- [12] A. Nouredine, A. Eskandarian, and K. Digges, “Computer modeling and validation of a hybrid III dummy for crashworthiness simulation,” *Mathematical and Computer Modelling*, vol. 35, no. 7–8, pp. 885–893, Apr 2002.
- [13] N. Hosseinichimeh, A. K. Wittenborn, J. Rick, M. S. Jalali, and H. Rahmandad, “Modeling and estimating the feedback mechanisms among depression, rumination, and stressors in adolescents,” *PLoS ONE*, vol. 13, no. 9, p. e0204389, Sept 2018.
- [14] M. Azangoo, L. Sorsamäki, S. A. Sierla, T. Mätäsniemi, M. Rantala, K. Rainio, and V. Vyatkin, “A methodology for generating a digital twin for process industry: A case study of a fiber processing pilot plant,” *IEEE Access*, vol. 10, pp. 58 787–58 810, 2022.
- [15] Dassault Systèmes®, “Simulia Realistic Simulation Solutions.” [Online]. Available: <https://www.3ds.com/products-services/simulia/products/>
- [16] Siemens, “LMS Virtual.Lab.” [Online]. Available: https://www.ddsplm.com/wp-content/uploads/2017/06/Siemens-PLM-LMS-Virtual-Lab-br_tcm1023-218353-1.pdf
- [17] Hexagon, “Adams.” [Online]. Available: <https://hexagon.com/products/product-groups/computer-aided-engineering-software/adams>
- [18] The AnyLogic Company, “AnyLogic Simulation Software.” [Online]. Available: <https://www.anylogic.com/>
- [19] isee systems, “Stella® Simulator.” [Online]. Available: <https://www.iseesystems.com/store/products/stella-simulator.aspx>
- [20] A. Junghanns, C. Gomes, C. Schulze, K. Schuch, P. R., M. Blaesken, I. Zacharias, A. Pillekeit, K. Wernersson, T. Sommer, C. Bertsch, T. Blochwitz,

- and M. Najafi, “The functional mock-up interface 3.0 - new features enabling new applications,” in *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021*, Sept 2021, pp. 17–26. [Online]. Available: <https://ecp.ep.liu.se/index.php/modelica/article/view/178>
- [21] G. Schweiger, C. Gomes, G. Engel, I. Hafner, J.-P. Schoegg, A. Posch, and T. Noudui, “Functional mock-up interface: An empirical survey identifies research challenges and current barriers,” in *Proceedings of The American Modelica Conference 2018, October 9-10, Somberg Conference Center, Cambridge MA, USA*, Feb 2019, pp. 138–146. [Online]. Available: https://ep.liu.se/en/conference-article.aspx?series=eep&issue=154&Article_No=15
- [22] Modelica Association, “Functional Mock-up Interface Specification.” [Online]. Available: <https://fmi-standard.org/docs/3.0/>
- [23] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauss, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf, “The functional mockup interface for tool independent exchange of simulation models,” in *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical University; Dresden; Germany*, June 2011, pp. 105–114. [Online]. Available: https://ep.liu.se/en/conference-article.aspx?series=eep&issue=63&Article_No=13
- [24] DNV GL AS, Kongsberg Maritime CM AS, SINTEF Ocean AS, and NTNU. [Online]. Available: <https://opensimulationplatform.com/>
- [25] DNV GL AS, Kongsberg Maritime CM AS, SINTEF Ocean AS, and NTNU, “DP-Ship.” [Online]. Available: <https://open-simulation-platform.github.io/cosim-demo-app/DPSHIP>
- [26] S. Fortmann-Roe, “Insight maker: A general-purpose tool for web-based modeling & simulation,” *Simulation Modelling Practice and Theory*, vol. 47, pp. 28–45, 2014. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1569190X14000513>

- [27] C. M. Macal and M. J. North, “Tutorial on agent-based modelling and simulation,” *Journal of Simulation*, vol. 4, no. 3, pp. 151–162, Sept 2010.
- [28] Insight Maker, “Simulation and modeling for Node and the browser.” [Online]. Available: <https://github.com/scottfr/simulation>
- [29] Posit, “About us.” [Online]. Available: <https://www.posit.co/about/>
- [30] Posit. [Online]. Available: <https://www.posit.co/>
- [31] Posit, “RStudio.” [Online]. Available: <https://github.com/rstudio>
- [32] R. Vaidyanathan, K. Russell, and Posit, “htmlwidgets for R.” [Online]. Available: <https://www.htmlwidgets.org/>
- [33] Posit, “Shiny.” [Online]. Available: <https://shiny.rstudio.com/>
- [34] Wolfram, “Wolfram mathematica.” [Online]. Available: <https://www.wolfram.com/mathematica/>
- [35] OpenAI. [Online]. Available: <https://openai.com/>
- [36] S. Wolfram, “Wolfram|Alpha as the Way to Bring Computational Knowledge Superpowers to ChatGPT.” [Online]. Available: <https://writings.stephenwolfram.com/2023/01/wolframalpha-as-the-way-to-bring-computational-knowledge-superpowers-to-chatgpt/>
- [37] Python Institute, “Python - the language of today and tomorrow.” [Online]. Available: <https://pythoninstitute.org/about-python>
- [38] Google, “Tensorflow hub library overview.” [Online]. Available: https://www.tensorflow.org/hub/lib_overview
- [39] OpenCV, “About.” [Online]. Available: <https://opencv.org/about/>
- [40] Publications Office of the European Union, “About cordis.” [Online]. Available: <https://cordis.europa.eu/about>

- [41] Horizon 2020, “Virtual materials market place.” [Online]. Available: <https://cordis.europa.eu/project/id/760907>
- [42] Mathworks, “MATLAB.” [Online]. Available: https://se.mathworks.com/products/matlab.html?s_tid=hp_products_matlab
- [43] Mathworks, “Simulink.” [Online]. Available: https://se.mathworks.com/products/simulink.html?s_tid=hp_products_simulink
- [44] T. Karhela, A. Villberg, and H. Niemistö, “Open ontology-based integration platform for modeling and simulation in engineering,” *International Journal of Modeling, Simulation, and Scientific Computing*, vol. 03, no. 02, p. 1250004, June 2012.
- [45] J.-P. Laine, “Thin-client architecture for simantics simulation environment,” Master’s thesis, Helsinki University of Technology, 2009.
- [46] T. Lempinen, S. Ruutu, T. Karhela, and P. Ylén, “Open source system dynamics with simantics and openmodelica,” in *Proceedings of the 2011 Conference of the System Dynamics Society*, July 2011.
- [47] VTT Technical Research Centre of Finland Ltd, “SULCA - Sustainability tool for Ecodesign, Footprints & LCA.” [Online]. Available: <https://www.simulationstore.com/sulca>
- [48] Fortum and VTT Technical Research Centre of Finland Ltd, “Apros.” [Online]. Available: <https://www.apros.fi/>
- [49] Docker Inc., “Use containers to Build, Share and Run your applications,” Nov 2021. [Online]. Available: <https://www.docker.com/resources/what-container/>