DIGITAL TWIN MODEL FOR A GAS TURBINE

Lappeenranta–Lahti University of Technology LUT

Master’s programme in Lappeenrannan-Lahden Teknillinen Yliopisto LUT

Master’s thesis

2024

Aleksi Lehtomäki

Examiner(s): Professor Teemu Turunen-Saaresti

Matti Päivärinne MSc. (Tech.)

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUTin energiajärjestelmien tiedekunta

Energiatekniikka

Aleksi Lehtomäki

Digitaalisen kaksosen kehitys kaasuturbiinille

Energiatekniikan diplomityö

2024

52 sivua, 9 kuvaa, 2 taulukkoa ja 2 liitettä

Tarkastaja(t): Professori Teemu Turunen-Saaresti ja DI Matti Päivärinne

Avainsanat: Kaasuturbiini, dynaaminen mallinnus, automaatio, digital twin, FMI, Modelica

Yrityksen tavoite on kehittää digitaalinen kaksonen, jonka ensimmäinen suora käyttökohde on automaatio-ohjelman testaaminen virtuaalisesti kehitystyön aikana. Dynaaminen prosessimalli, jonka päälle on myöhemmässä vaiheessa mahdollista rakentaa käyttödataan perustuvaa digitaalisen kaksosen toiminnallisuutta sopii automaatiokehityksen tarpeisiin. Mallin sujuva ja helppokäyttöinen liitettävyys automaatiokehitysympäristö TwinCATiin on tärkeä ominaisuus.

Mallin toteutustavaksi valikoitui OpenModelicalla luodun Modelica-kielisen mallin käyttö Functional Mockup Interface -standardia käyttäen. Myöhemmässä vaiheessa TwinCAT-kommunikaation toteutustapa tarkentui Python-ohjelmaan, joka suorittaa simulaation ja TwinCAT-tiedonvaihdon rinnakkain erillisissä prosesseissa.

Mallin rakentamisessa tärkeintä oli kaksiakselisen välijäähdytetyn ja rekuperoidun kaasuturbiinin akselien mekaanisen liitoksen ja akselien välisen virtauksellisen liitoksen yhteensovittaminen luotettavasti ja vakaasti. Kehitysprosessi oli iteratiivinen ja kokeileva valmiin tiedon vähäisen määrän vuoksi.

Kehitystyö toimintavarmaksi suorituskykyiseksi ohjelmaksi yksinkertaisesta toteutustavan kelpoisuuden todistavasta kommunikaatioskriptistä toteutettiin käyttämällä vakiintuneita kirjastoja ja käytäntöjä.

Työn lopputuloksena on kommunikaatio-simulaatio-ohjelma, joka käyttää vapaavalintaista vaaditut muuttujat sisältävää Functional Mockup Unitia ja välittää automaation ohjaussignaalit ja simulaation askelittaiset tulokset ristikkäin tihein välein ja pitää simulaation reaaliajassa automaation ohjaamana.

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Energy Systems

Energy Technology

Aleksi Lehtomäki

Digital twin model for a gas turbine

Master’s thesis

2024

52 pages, 9 figures, 2 tables and 2 appendices

Examiners: Professor Teemu Turunen-Saaresti and Matti Päivärinne MSc. (Tech.)

Keywords: Gas turbine, dynamic modelling, automation, digital twin, FMI, Modelica

The company needs a digital twin model. The first use for the model is to test automation software during development. For this need, a dynamic physics model with the option of building data-driven digital twin functionality as an extension was chosen. Connectivity with TwinCAT was crucial.

Implementation of the model was chosen to be written in Modelica using OpenModelica as the software. For connectivity, using Functional Mockup Interface was chosen for modelling software independency. The eventual connectivity solution was a Python program that runs the simulation and communication processes in parallel.

While modelling a two-shaft intercooled recuperated gas turbine, the mechanical link in shafts and fluid dynamical link between shafts was important to model in a numerically stable and robust way. The development path was iterative and experimental as readily available information was limited.

The development from a simple proof-of-concept communicator program into a stable and performing program was done with established Python libraries, principles and practices.

The result of this thesis is a communication-simulation program that simulates a user chosen Functional Mockup Unit that has the minimal required set of variables. Program exchanges simulation and control data between the unit and the automation in fast cycles while maintaining real time simulation controlled by automation.

SYMBOLS AND ABBREVIATIONS

Abbreviations

CS	Co-Simulation
DT	Digital Twin
FMI	Functional Mockup Interface
FMU	Functional Mockup Unit
IRG2	Intercooled Recuperated 2-shaft Gas turbine process
ME	Model Exchange

Table of contents

1 Introduction

The objective for this thesis is to initiate development for a digital twin model for Aurelia Turbines A400 gas turbine. Initially, a model is required for increased efficiency in control software development for the gas turbine, as testing software on hardware is expensive, and not essential for preliminary testing. Development towards a digital twin is natural, as the model must be coupled with the automated control system, where real world data is also available. Software development is much faster when preliminary testing can be done virtually. Development of Aurelia® A400 enables further efficiency gains and increased fuel versatility, which in turn enables efficient and versatile distributed small scale power generation using various fuels.

Digital twins of different complexity levels have been developed before for various devices and systems, including reasonably similar gas turbines. Digital twins are a versatile technology that can be used for various use cases with gas turbines including maintenance management, fouling simulation, process development and software development. Automation software development aid is the main goal within the scope of this thesis.

This thesis revolves around the unique design of Aurelia® A400, and for a unique design, a pre-existing model can't be directly adapted. The initial use case of testing and developing software is suitable for a dynamic model of the system. In conclusion, the aim of the thesis is to aid software development for the physical gas turbine by researching and implementing a way to create a simple general dynamic model for the gas turbine suitable for the development and testing of software and then interfacing it with the current automation solution. The research is done in two parts, a theoretical part where a suitable software and its feasibility is researched and a practical part where the chosen software is used to implement a simple dynamic model. Time constraints limit the scope of development included in this thesis to a rather simple implementation of the desired result that is to be used as a basis for future advancements towards a model that can be used for many purposes including certifications and management. In essence, a dynamic model with automation connectivity to serve as a base for further advancements is the main goal.

1.1 Disclaimer on ChatGPT usage

ChatGPT has been used heavily to accelerate research and development for this thesis. Its versions 3.5 and 4.0 have been used for the following purposes during the research and development phase of the final product:

- Quick preliminary research
- Troubleshooting various software issues
- Snippets of Python and Modelica code mostly for syntax or library use examples
- Troubleshooting code through error messages
- Looking for academical sources
- Pointers for structuring information
- Suggestions for advancement directions
- Assistance in creating Python middleware to communicate between FMU and ADS
- Assistance in creating a Python program for analysing simulation results and real test data.

ChatGPT has not been used for the following:

- Text generation for this thesis
- Generating code directly

2 Background

The background of the thesis, the product Aurelia® A400 and the concept of digital twins are described in this chapter. The unique process design and emerging technology of a digital twin are the source of novelty for this research effort.

2.1 Description of Aurelia® A400

Aurelia® A400 is a small gas turbine capable of producing 400kW of electricity with efficiency of 40%. A400's process is designed for high efficiency and fuel versatility to be usable in various use cases of distributed electricity generation in small scale. High partial load efficiency and control speed are both important distinguishing qualities of the intercooled recuperated two-shafted gas turbine process (IRG2) compared to other types of gas turbines. Aurelia A400 is also capable of combined heat and power production, as both intercooling heat and exhaust heat can be recovered. (Aurelia Turbines, 2023)

2.1.1 Process description

Aurelia® A400 is built around Aurelia Turbines' patented IRG2 process. The process uses two shafts with a compressor, turbine and a generator inline on both shafts. Air flows from LP compressor to the HP compressor through an intercooler. After HP compressor, the air is heated in a recuperator before the combustion chamber. In the combustion chamber, fuel is added and combusted to generate high-pressure exhaust gas to drive the turbines of the process. From HP turbine, the exhaust travels to LP turbine, generating power for their respective shafts. After the turbines, the now low-pressure hot exhaust is directed to the recuperator where the heat is recuperated back to the process. Diagram in Figure 1.

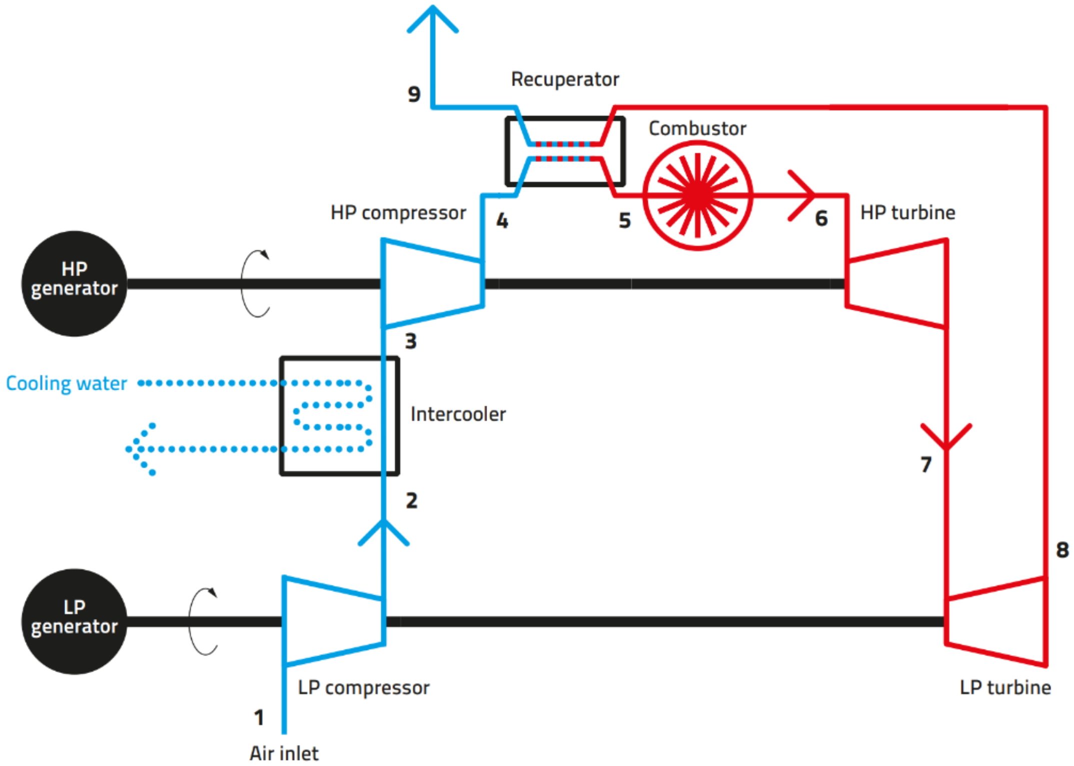

Figure 1: IRG2 process diagram

Both shafts are controlled separately from each other, which allows for exceptionally high partial load efficiency and rapid control compared to single-shaft and conventional dual-shaft designs. Intercooling increases the efficiency of the process compared to other two shaft gas turbines, where the other shaft is usually only for power generation instead of two stage compression.

2.1.2 Construction and other features

The A400 and its electrical equipment are contained in a 3.0 x 3.3 x 9.7 m enclosure and the gas turbine weighs 22 000 kg. The shafts are equipped with active magnet bearings that levitate the shafts. This allows the gas turbine to operate friction and lubricant free, which reduces requirement for maintenance. The gas turbine is vibration free, and it can be installed indoors or outdoors. The system is capable of island operation and black starts, which are highly beneficial features for distributed power generation applications. A render visible in Figure 2.

Figure 2: Render of the A400 with transparent side panel

2.2 Digital twin

A digital twin is a virtual representation of a physical asset. Compared to a conventional simulation model, a digital twin is a model that is coupled with the physical asset and trained with real data from the physical asset. Digital twin as a technology is versatile and can be used for a wide range of different kinds of physical assets and use cases. The idea of a digital twin is in no way new. A similar concept has been thought about since the 1990s and the term was first coined by NASA in 2010. NASA described it as an integrated multi-physics, multi-scale, probabilistic simulation of a system with the best physical models, sensor updates and history information. The point is to mirror an actual asset. (Singh *et al.*, 2021)

Digital twins can be applied to a wide array of different industries including but not limited to: manufacturing, agriculture, healthcare, automotive, aviation, construction, mining and utilities. The same underlying technology can be used in different ways to accommodate various use cases, and increased amount of data available is going to further enable this data-driven method. Digital twins can be used in a predictive manner to be ahead of reality when the twinning is successful. (Attaran and Celik, 2023)

As digital twins are an emerging technology, the definition of a digital twin varies by who is defining it. A simple physics model with automated tuning for increased modelling accuracy could be a digital twin, a physics model with the ability to optimise the way it is controlled could be a twin, or maybe the application must use data extensively and in a predictive manner to be a digital twin. For this thesis, a digital twin in general is a virtual representation of a physical asset, a gas turbine for this thesis. At this point, a single unit is not mirrored, and a general model that can be developed into single-unit twins is the focus.

2.2.1 Different applications of digital twin technology

Digital twins are a versatile technology and there’s a lot of potential yet to be discovered on the subject. This chapter focuses on applications of digital twins related to gas turbines.

GE Digital has used digital twin technology to create autonomous tuning for aeroderivative turbines (GE Digital, 2022). The system adjusts the combustion process autonomously to get better combustion stability and lower emissions. Compared to manually tuning for average conditions on site, the automated solution enables further optimisation based on minuscule changes in operation. This is a system that adjusts fuel splitting and flame temperature to achieve optimal performance and emissions every two seconds. Essentially the system has a machine learning model to achieve this, as direct physics or numerical approaches are not able to accomplish this functionality. Advanced control by modelling behaviour with machine learning models and digital twins is a significant area of research for gas turbines. Capacity and efficiency modelling for performance tracking has been used to detect faults early, before conventional methods can detect faults (Panov and Cruz-Manzo, 2020). Digital Twins are especially useful when multiple identical units are deployed around the world, as this helps generate massive amounts of data to base models on, and even a small abnormality can be detected this way. A small abnormality that doesn’t trigger a normal alarm or a fault but is detectable with DT technology enables maintenance and corrections before the problem gets severe. (Robb, 2024)

EPRI has been working on a digital twin for standard and combined cycle gas turbines with the aim of providing operators with information and capability that is generally limited to

OEMs and service providers. This digital twin model can detect unexpected behaviour and predict performance levels with good accuracy, and most importantly provide the owner or operator of the turbine with easily readable information about the physical asset. This includes fault diagnostics and fault prediction. This twin works on already existing data to generate information that is not readily clear from the said data. This includes unbiased estimates of efficiencies and machine health derived from logging data.(Lim *et al.*, 2021)

3 Viability and selection of software

The goal for this chapter is to determine which of the available software options are viable for developing a model suitable for automation testing. Integration with existing control software, TwinCAT from Beckhoff, is crucial. The software should be suitable for efficient modelling and simulation and be suitable for machine learning applications for future improvements.

3.1 Requirements for the digital twin

The digital twin for Aurelia® A400 must be capable of real time simulation. Output must be realistic values simulating various sensors in the machine, and input will be TwinCAT automation software. Accurate physics simulation in 3D like CFD is not necessary to accomplish this requirement, as the target is to simulate existing behaviour instead of researching how to achieve desired performance values. Twin should remain high-level and use lower level results from CFD simulations of components as basis. For this initial application a dynamic process model with potential for future machine learning features was determined to fill the requirements. The model should be versatile for further development for other purposes. The digital twin must be able to directly communicate with the control software used, TwinCAT. TwinCAT is a highly compatible software, and the requirement can be fulfilled in many ways, by using Automation Device Specification (ADS), Beckhoff's communication protocol, or having the model exported with Functional Mockup Interface, and directly imported within TwinCAT project. FMI export is generally a common feature in simulation and modelling software (Modelica Association, 2023). Usability without expertise or technical specific knowledge on simulation and the communication option is necessary.

3.2 Criteria

Practical and highly usable software for modelling of gas turbine process and physics is mostly commercial, and suitable software must be chosen with care by using sensible criteria to avoid resource waste. To be considered as a viable option, integration capability for TwinCat is essential. TwinCat uses its own ADS system for external communication and an addon is available for using Functional Mockup Interface (FMI), which is supported by most simulation software options. Connectivity with TwinCAT can be done many ways and most simulation software has support for FMIs, which are a promising way of using simulation easily and transportably.

3.2.1 General usability

The usability, availability of support, quality of the available documentation and readability of the model must be considered, as if any of these are weak, the future usefulness of the digital twin will suffer. Continuity is crucial for this project, and therefore all parts of the toolchain must be established and stable. Rare niche methods are not as favourable as commonly used methods, as expertise on the common methods is easily available, and for niche solutions future usability is limited by available expertise.

3.2.2 Computational cost

Computational efficiency must be considered, as software with efficient reduced order modelling and efficient calculation methods require significantly smaller resources compared to a computationally expensive option, as the digital twin complexity will increase over time. Measuring computational cost is not viable at a preliminary stage, but estimation with a simple model can be done. For example, a model converted into a low-level programming language and compiled is significantly faster than direct Python that interprets code at runtime, as the interpreter introduces overhead. Eventually training a machine learning model requires performance in any case, but overall performance requirements during use must remain reasonable, and excessively complicated calculation rarely gives better results. To summarise, a solution that cannot run in real time on an upper end laptop is not viable.

3.2.3 Monetary cost

Price of the software to be used is a natural factor, as overly expensive, but otherwise great software is not the best value for the use case. Ideally, a cost-effective option with the possibility to migrate to a more advanced solution in the future would be most efficient. Evaluating best value for money must take ease of use and speed of model development into consideration, as a free tool that causes constant issues and has bugs will eventually be more costly than an expensive commercial option that just works. Evaluation of how much something ‘just works’ is close to impossible beforehand without testing especially for a case where multiple programs must work together on previously undocumented territory.

3.3 Available software

This chapter showcases the possible software options and their distinguishing features for creation of the digital twin. General viability for the project is assessed, and distinguished features are compared with other options.

3.3.1 MATLAB/Simulink with SimScape

MATLAB/Simulink is a general-purpose software suite by MathWorks. Documentation and support are readily available, and consultation is available if needed.. This software is proven in creating a digital twin for a gas turbine (Yu *et al.*, 2005; Panov and Cruz-Manzo, 2020). The software is capable of real time simulation. MathWorks is an experienced vendor of mathematics software, and the level of polish and quality is good. Everything is done locally or on a customer’s server. Cloud services are also available if required. The SimScape package for Simulink extends function block diagrams to physical dynamic modelling, which is crucial for modelling a system where most of the interactions between components are bi-directional.

As a major benefit, the base software is widely taught in universities and therefore future usability is good due to widespread competence around especially MATLAB, but also Simulink.

3.3.2 Collimator

Collimator is a newer solution based on Python. This software is cloud-based and has no downloadable application. This option can’t be run locally. This is a risk due to possibility of the service going bankrupt. The creation of a model is simple and straightforward, but the general level of polish and professionalism is not at a level desired. For example, major glitches in the graphical user interface are not acceptable for professional use. This solution has potential to become excellent, but Collimator is clearly too early in development to be used for this application, as a stable and proven method is necessary. Collimator is capable of complex models, but it is lightweight compared to the older and more mature options. Computational resources available seem to be strong and results are available fast. (Collimator, 2024)

3.3.3 Ansys Twin Builder

Ansys Twin Builder is capable of advanced simulation for various kinds of equipment. Ansys as a company is a veteran on the field of simulation, and their products are valued highly. This piece of software is highly capable of using various ways of modelling. The software is compatible with Modelica and FMIs, and it can use many other types of modelling. However, this software would probably be more valuable at a later stage if the digital twin for A400 is developed further. For the initial need for automation software development aid, this option offers little benefits. Their overall software suite is highly capable for most engineering tasks. Ansys as a provider is generally on the upper end of the price spectrum. (Ansys, 2021)

3.3.4 SciLab/Xcos

SciLab/Xcos is an open-source alternative to MATLAB/Simulink. The environment is highly similar compared to MATLAB/Simulink, but as FOSS (Free and Open Source Software) it has lower quality documentation, and less support is available. The software is not as advanced as MathWorks's alternative, but the newest and most advanced features of Simulink are not essential for success in digital twin development. However, as FOSS the software is free to use. This software is probably capable of fulfilling the requirements, but this software is also open-source, and no proper technical support is available in case there is technical issues. The toolkits available seem to be of the simpler side, but a Modelica based toolkit Coselica is available. It is however lacking in features compared to using full Modelica. However no distinguishing features were found to indicate using this solution over a Modelica implementation. (Dassault Systemes, 2023b)

3.3.5 Altair Activate

Altair delivers an engineering suite reasonably similar to ANSYS with a wide array of different pieces of software. Their offer similar to Simulink is called Activate. This option is in reasonably same price range as MATLAB/Simulink without the benefit of widespread familiarity and status as a monolith in the field. However, Activate has a Modelica compiler for dynamic system modelling, and therefore it is highly capable to perform the task at hand. The software is a little bit more cost-efficient compared to the MathWorks offering. (Altair, 2023)

3.3.6 Dassault Dymola

Dassault has a highly advanced Modelica implementation called Dymola. Dymola is a commercial tool for Modelica with advanced features not available elsewhere. As the language is the same, models and classes are highly compatible across different implementations of Modelica. Migrating from OpenModelica to Dymola is a possible choice if Modelica is chosen as the modelling language and the capabilities of the open-source option are exceeded. (Dassault Systemes, 2023a)

3.3.7 OpenModelica

OpenModelica is a completely free and open-source implementation and editor for Modelica. Compared to Dymola, it has lower compatibility with various libraries due to libraries being developed with Dymola, and the user experience is not as polished when comparing to the commercial options. However, during testing the open-source option was found to be highly usable for early phase development, and the relatively easy migration to Dymola if needed is a substantial benefit for future use. (OSMC, 2023)

3.4 Comparison between options

While researching for software options, physics and model-based system modelling was presenting itself as the most beneficial option. Compared to block diagrams and causal approaches, acausal approach is highly beneficial as the product to be modelled has multiple acausal relationships and a lot of connections that would require a feedback loop if only causal modelling were to be used. For example, an increase in fuel inlet causes hotter flue to enter the recuperator which in turn increases the temperature entering the combustion chamber reducing the need for fuel. Two options that work in that manner were found, various Modelica implementations and SimScape.

Modelica as a modelling language was found to be proven in highly similar applications, both for aid of software and automation development and for the specific use case of a gas turbine (Idebrant and Näs, 2003; Link *et al.*, 2014). The language itself is equation-based and object-oriented, which enables high reusability of code. The ability to write a class from scratch or extend an existing one with relative ease in case a suitable library model is not available makes this option highly viable for the specialized product even for future developments where the behaviour of a component might differ from commonly used similar components.

The other viable option is SimScape, which is an add-on for MATLAB/Simulink. It is like Modelica in many ways, especially the way it works is highly similar. The main difference comes from their respective backgrounds. SimScape is newer and commercial from a single company, where Modelica is a lot older as a concept and open source being developed by multiple companies and universities. SimScape is tightly integrated into MATLAB/Simulink where Modelica is a language specification, and tool independent. The comparison between these modelling methods will mostly be between available libraries and usability, as the language on the background is clearly powerful enough for both options and the options are similar in functionality.

3.5 Testing the software

Two of the options researched were chosen to be evaluated further, SimScape by MathWorks and OpenModelica by OpenModelica Consortium. The most important factors that affect the overall usability for the project are available component models and overall flexibility. Some key differences in functionality also exist. Modelica's functionality was tested with OpenModelica, and SimScape was evaluated with a trial license. The Modelica library used was ThermoPower, and for SimScape it was SimScape Fluids. For Modelica, there are also other library options available for simulation of power plant systems. Multiple differences between the options were found, some meaningful, some negligible.

3.5.1 Medium models

SimScape and Modelica handle mediums differently in a very meaningful way. In SimScape, a gas domain is handled as a single gas with single properties. In Modelica, the gas domain takes a model of the medium within the flow, which can be different in every pipe of the system if necessary. These models include gas compositions and properties that vary accordingly if the composition or type of media changes within the system. For SimScape, "global" gas properties are used for a similar purpose within a gas network. The same system model can have multiple gas networks, so the properties are not actually global. To change the properties and mass flow of the working fluid at combustion chamber, a new network that gets its state transferred must be used. This is cumbersome, especially as the relationship between the gas states over the combustion chamber is bi-directional.

3.5.2 Turbomachinery models

Models for compressor and turbine behaviour are the most important for developing a model for a gas turbine with four turbomachines. For both software options and both types of turbomachines, the models work by being supplied a characteristic map. For SimScape, the maps were a lot easier initially, as the model can generate visually pleasing and clear compressor and turbine maps and curves from a set of variables with corrected mass flows and rotational speeds. However, when a tabulated version of a compressor map was to be supplied for the component, the SimScape approach was a bit more cumbersome, as the tables are taken as MATLAB variables or by direct parameter compared to OpenModelica's .txt file and parameter approach. The .txt file was easy to debug into working condition between runs, as the text file could be edited outside of OpenModelica between consecutive runs where the maps are loaded without recompiling. SimScape's integration with MATLAB resulted in a stuttering and slow user experience on the limited hardware the testing was done on. SimScape's slower user experience might be insignificant with a more suitable higher end computer. Both environments have capable models for turbomachinery components, however, inspecting and adjusting the model behaviour is easier in OpenModelica, as a viewing and editing classes is an expected action with Modelica, where it seems to be treated more as an advanced feature in SimScape. OpenModelica has a separate diagram and text views, next to each other. For SimScape the model is opened in MATLAB editing window, and switching between these is less smooth. For this thesis, advanced use is a requirement for successful development due to the complicated and highly interlinked nature of the product and its components.

3.5.3 Combustion chamber model

This part of the testing showed the most significant differences between the options. SimScape does not have a readily made combustion chamber model but uses a heated pipe model instead for examples. This does not hinder the software's ability to calculate the necessary calculations, but it requires development time to implement and validate a sufficient model for a combustion chamber. The heated pipe approach is limited for a variety of reasons:

- Mass flow does not increase in the combustion chamber. This is significant for low heating value fuels, where error can be around 10% of mass flow after combustion.
- Direct mass flow of fuel cannot be used. This and the above can be remedied by writing a combustion chamber model in SimScape
- The composition of the flue gases is ignored, and flue gases must be simplified as air. This can be circumvented, but not with a clean solution. Using the moist air domain for SimScape is possible to limit the inaccuracy from this issue. However, having chemical composition of flue gases straight out of the box is desirable for future use, for example analysing sulphuric fuels

None of these are directly crucial for control system development, but for future usability with the fuel and electrical power flexible turbine model this is an issue, as the mass flow and the ignored flue gas properties cause error in the simulation results, meaningful especially if the model is to be used for process development purposes in the future. This need for simplification with a pipe model caused major issues with initialization, as the error from this method is substantial enough to deviate from design point and cause the need to hunt for a working initialisation point.

ThermoPower for Modelica has a combustion chamber model that can calculate combustion rates, determining the flue gas composition and it adds the mass flow received from the fuel to the flow. On top of this, also heat accumulation in the combustion chamber walls is accounted for. The model at its current state takes the lower heating value of the fuel (natural gas used for testing) as a parameter, but a simple couple rows of extension for the model could be used to estimate the heating value from the fuel composition. This would enable advanced simulation of varying fuel quality and even type during runtime. For this reason, the Modelica approach on this issue is very desirable.

3.5.4 Initialisation

For initialization of the model, SimScape offers a bit more user accessible flexibility, as there’s a set priority for start values. High priority means that the value should be exactly as defined, and low priority means the value should be close to the defined value. Priority of none means just a guess value to start iterating on. The existence of low priority helps a lot on initialisation compared to Modelica, where similar concept is not available directly to the user. Modelica has no direct method to favour a value over another one to be closer to its start value. Start values are either fixed or not, low priority start values are not directly implemented. In Modelica this must be done by coercing the model to choose the desired value out of the numerically correct options or finding an exact start value for enough variables that lead to the desired result without going singular. This can be tedious for major shifts in parameters. However, adjusting parameters and start values in small steps seems to be an efficient way to reach the desired result. This issue is prominent in the early stages of model development and is a factor why Modelica has a steeper learning curve, as the model must be more correct to initialize at all.

3.5.5 Overall user experience

OpenModelica is not as graphically pleasant as MathWorks' SimScape environment. MathWorks is also an experienced company with data visualisation and analysis, and therefore these features are exceptionally good in their offering. For example, the compressor and turbine blocks can output visually pleasant graphical maps for their characteristics, and data visualisation after simulation is easy and pretty to work with. OpenModelica does have usable and clear plotting features, but SimScape is graphically more advanced. A personal preference of the writer for working with text for the details and graphically for the big picture is satisfied better with OpenModelica as text and graphical editing are considered as equal options instead of one way being primary compared to the other. OpenModelica's user interface in the plotting view is shown in Figure 3.

SimScape as a MATLAB extension is not as nice to use, as whole MATLAB is running as useless overhead, when the work to be done is mostly within SimScape. The standalone nature of OpenModelica is better suited for the task. Also writing custom libraries and using them is more straightforward with OpenModelica.

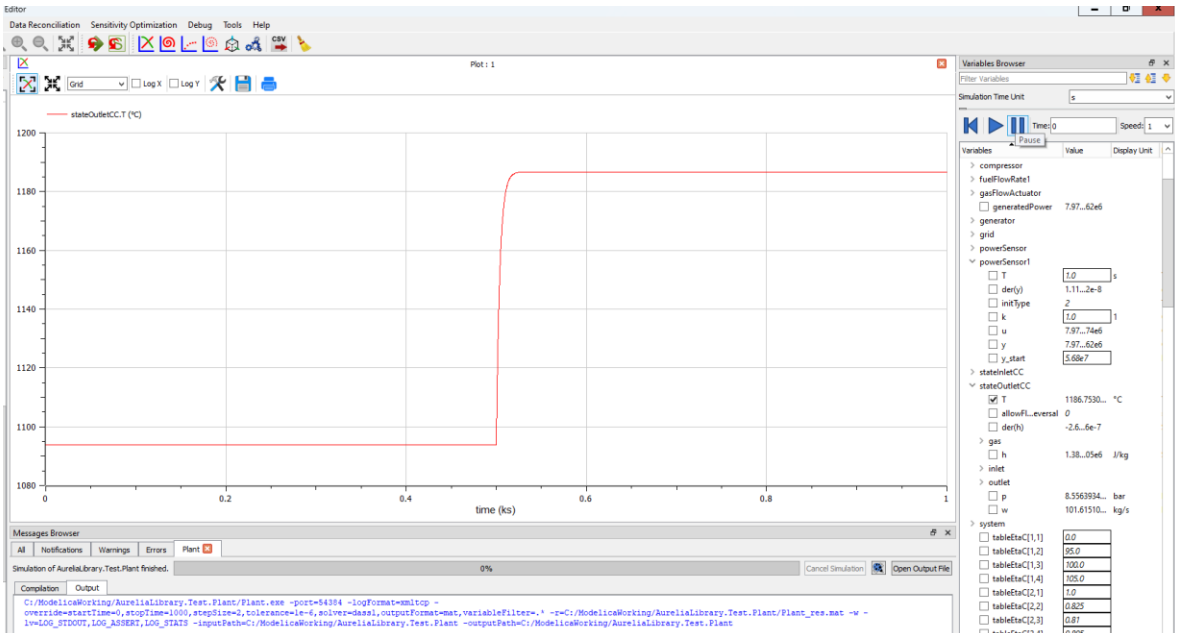

Figure 3: OpenModelica plotting window

3.6 Initial choice of software

During the testing phase for different software for their preferability and usability towards the main modeller, OpenModelica was chosen as an initial pick with a future option to switch over to Dymola or MapleSim if the limits of OpenModelica are reached. The limits of Modelica as a language are far outside the scope of this thesis, and FMI is highly supported with Modelica implementations. If a part of the system must be simulated in another environment for some reason, FMUs can be used. SimScape also has FMI functionality, but it is more limited. For this feature, full FMI support, Modelica implementations are a high-ranking choice from the start. Python can also be easily used to implement anything with concurrent simulation if there is no reasonable way to model something inside a single Modelica project and it needs to be coupled with the model. This could be the case for simulating an external device that interacts with the A400.

4 Modelica and FMI

The Modelica implementation OpenModelica was chosen as the tool to be used for the dynamic physics modelling task. This chapter contains a brief description of Modelica and OpenModelica, which is the chosen Modelica implementation.

4.1 Modelica

Modelica is an equation-based modelling language. It is an open-source effort first published in 1997 by the Modelica design group with the aim to create language for interoperability in an environment where multiple simulation and modelling languages were competing with each other (Elmqvist *et al.*, 1997). From this point onwards, Modelica has been in active development. The Modelica Association was formed in the year 2000 to handle the effort. After this, Modelica, and Modelica Standard Library (MSL) have been developed by the association that consists of software vendors, users of Modelica and academic institutions. This trifecta of stakeholders is a major strength for the language.

The main features of the language are:

- Declarative
- Object-oriented
- Multi-domain
- Equation-based

Being declarative means, that the language describes what the program should do without describing how the program should achieve the goal. For Modelica, this means that the Modelica code gives no instructions on how to achieve the goal set by the system of equations, and this work is done by the compiler that generates code, usually C, from Modelica. First the software sorts the equations into groups, and then implements code to solve said groups of differential and algebraic equations. Different compilers that translate Modelica into simulation code exist, most notable of these are Dymola, MapleSim and OpenModelica Compiler. Altair Activate uses MapleSim’s compiler for its Modelica features. The language of the final code to be executed is not determined within Modelica, and a Modelica compiler can be written to generate code in any language. Object-orientation means that instances of classes are used. For example, multiple instances of the class Resistor can be instantiated with varying parameter values for resistance. The code for Resistor is written once and called for each instance with different parameters. This enables high reusability of code. Inheritance and extending classes enable efficient reuse of code, which enables compatibility between different extensions from the same base class and effortless creation of customized classes for special use cases, for example a resistor that has a thermal domain in addition to the electrical functionality. The usage of a base class ensures that a resistor without thermals can be swapped for a resistor with thermal features easily.

The multi-domain capabilities of Modelica mean that the same model can contain various domains, for example the model written in this thesis consists of a gas domain for fluid flow, thermal domain for heat transfer in the heat exchangers, a mechanical domain for the shafts and their inertia and an electrical domain for the generators. In addition, the model also has some minor control logic implemented to achieve goals indicated by the automation software. Without this multi-domain capability, each of the domains would have to modelled separately, running each domain based on the results of the other domains.

The open nature of Modelica has led to a large catalogue of advanced open-source libraries, of which the most notable one for this thesis is ThermoPower developed in Politecnico di Milano (Casella, 2014a). Proprietary highly advanced libraries from software vendors do exist and might be a future research target. ThermoPower was determined to be advanced enough for this thesis.

4.2 OpenModelica

For the beginning of Modelica as a language, the language was open to enable a unified approach to modelling and simulation between multiple software vendors. An open-source implementation of Modelica was not available for the first ten years of Modelica. The Open Source Modelica Consortium (OSMC) was formed in 2007 to create an open-source implementation (Open Source Modelica Consortium, 2023). The members of the consortium form a trifecta similar to the one in Modelica Association.

The result of this effort is OpenModelica, which consists of OMEdit, OpenModelica Compiler and OMNotebook. OpenModelica is compliant with the Modelica language specification, and it has reasonably good compatibility with libraries developed in other environments like Dymola. OMEdit is the editor for models. It provides a graphical user interface for modelling and textual editing of the model is also available. The editor also allows for plotting results of the simulation. OpenModelica Compiler translates the model code to C for it to be executed. The compiler is also capable of generating Functional Mockup Units, which are useful for simulation outside of modelling environment and co-simulating multiple models. OMNotebook is a notebook application that can be used as an interactive notebook for presenting text and models simultaneously. A good example of its use is DrModelica, a tutoring system for teaching Modelica. This approach enables the integration of documentation and code into the same notebook to present a model and its functionality. (OSMC, 2023)

4.3 ThermoPower

ThermoPower is a Modelica library developed in Politecnico di Milano during the early 2000's. The library consists of various components suitable for thermal plant and energy conversion system modelling. The library works with three domains, gas, water and electric. Examples of what has been implemented with ThermoPower include boiler setups, complete steam plants and heat recovery systems. The library is old but updated multiple times since conception. It has been developed in Dymola, and it is also compatible with OpenModelica. The approach used in the library is to use easily readable continuity and balance equations, no 3D flow components. This is easy to read and verify. Suitability for this project is high, as it has been developed with studying control systems in mind. (Casella, 2014a)

4.4 Functional Mockup Interface

Functional Mockup Interface is a free standard for model exchange and co-simulation. The most important part of the standard is the Functional Mockup Unit (FMU). It is a zip file, that encapsulates the model's description, binaries, resources and optionally C source code. The main point for FMI is to enable interoperability between simulation environments to allow development of a complex cyber-physical systems. FMI as a technology is not directly related to Modelica, but it could be viewed as a complementary technology. (Modelica Association Project FMI, 2022)

Three different kinds of FMU exist. Model Exchange (ME), Co-simulation (CS) and Scheduled Execution (SE). The last one was added in FMI 3.0, and this thesis uses FMI 2.0 FMUs. An FMI 2.0 FMU can be either ME or CS or both at the same time. The difference between the FMU types is if the integration and solving is done by the FMU or by the environment the FMU is ran in. Co-simulation FMUs have the solver and scheduler built-in, and require no solving capabilities from the environment they're ran in. Model Exchange FMUs require an external solving mechanism to calculate states and handle stepping and scheduling. There's not a lot of difference between functionality of ME and CS FMUs if the external solver used for ME is the same solver that is built-in into the CS FMU. For example, a ME FMU that uses CVODE for solver behaves similarly to a CS FMU that has in-built CVODE as a solver. However, the amount of control for solver settings is significantly greater when the solver is initialised and used by the environment, for example Python with FMPy. With a CS FMU the FMU handles the solver, and for example modified input handling is not possible.

The FMI standard has reached version 3.0 in 2022. Earlier versions FMI 1.0 and 2.0 are currently widely available for use, and the third version is emerging. FMI 3.0 is an interesting future direction for the project started in this thesis, as FMI 3.0 has improvements on both artificial intelligence usability and digital twins compared to FMI 2.0. Co-simulation is also improved from 2.0 and there's less reasons to use ME over CS when working with FMI 3.0.(Modelica Association, 2022)

The FMI standard does have its limitations. The most significant limitations for this thesis are on the area of event handling, as predetermined (for example timed events) are easy to manage and performing the necessary reinitialisation for variables can be done without the user noticing that anything happened. For continuous real-time simulation, event detection is significantly more difficult, as having an event causes slowdown in simulation, and not using event handling at all and using only continuous mode runs into issues when the situation is no longer linear due to a rapid input change. Efficient event detection that triggers when necessary but does not trigger excessive events is a way to get more functionality out of an FMU. Event handling has been improved for FMU 3.0

5 A400 dynamic model development

The development of the model began in the software testing phase, where the OpenModelica based trial model was found to be more capable than the SimScape-based option. Modelica offered libraries with better capabilities, and for SimScape a lot of simplifications were needed from the start. These simplifications caused enough error where initialisation was difficult in the design point without developing custom components or deviating from the existing turbo maps for a fair bit. Broader library availability stemming from the more open nature of Modelica means less modelling work for components and more time to focus on their interactions.

5.1 Model components

This section describes the process of modelling and adjusting various components towards a full model. Component models and the full model were developed in parallel, and a working combination of fewer components was gradually built into a full working model addition by addition.

5.1.1 Turbomachinery maps

ThermoPower uses tabulated compressor maps for its standard compressor and turbine models. A turbine model using the Stodola equation is also available.

The existing maps had to be converted, since the maps available were in a different format tabulated as a function of corrected mass flow and rotational speed for the compressor and pressure ratio and corrected rotational speed for the turbines. For this purpose, the flow numbers, as called in ThermoPower documentation (Casella, 2014b), had to be calculated from available data with the following equation 1. This flow number is commonly known as non-dimensional mass flow.

$$\phi_c = \frac{q_m\sqrt{T_1}}{P_1} \quad (1)$$

where:

ϕ_c	flow number
q_m	mass flow
T_1	gas inlet temperature
P_1	gas inlet pressure

The turbine maps are simple and take efficiency and flow number phi as a function of pressure ratio PR and percentage of design rotational speed N_T in two 2D tables. Example data shown in combined table Table 1. Header row contains percentage of design referred speed, while left column has pressure ratios. Rest of columns contain values for phi and efficiency eta.

Table 1: Example turbine map data

	phi	**eta**		
PR\N_T	100	93	100	105
1.38	6.56E-05	0.685635	0.604117	0.525753
1.46	7.45E-05	0.760898	0.67808	0.636025
1.54	8.27E-05	0.74844	0.703058	0.696143
1.62	9.16E-05	0.826715	0.832461	0.620616
1.7	0.000102	0.832485	0.668396	0.761143
1.78	0.000109	0.749839	0.684681	0.708649
1.86	0.000117	0.834025	0.739318	0.776331
1.94	0.000124	0.811232	0.775089	0.790356
2.02	0.000131	0.702071	0.722723	0.735312

For compressors, the maps work with the assistance of beta lines to avoid singularity. Beta lines are arbitrary and used to match the three 2D tables used in the compressor model. This way the correct combination of the three values is always chosen from the same line. These tables supply the model with flow number phi, efficiency eta, and pressure ratio PR as a function of beta and percentage of the design rotational speed N_T in their respectable tables. Example data in combined table Table 2.

Table 2: Example compressor map data

	Phi	**PR**			**Eta**		
Beta\N_T	100	84.5	100	104.2	84.5	100	104.2
1	6.96796E-05	1.986879	2.208885	2.28386	0.873485431	0.833255	0.732323
2	9.97531E-05	1.884272	2.245644	2.027181	0.699700082	0.817809	0.893452
3	0.000100092	1.613174	2.087385	2.052542	0.810882902	0.834333	0.684615
4	0.000104633	1.570336	2.233341	2.058833	0.929566527	0.738524	0.816362
5	0.000117735	1.700152	2.204144	2.24852	0.810337595	0.729909	0.88518
6	0.000115413	1.496371	1.842877	2.158549	0.941114948	0.754691	0.818569
7	0.000124834	1.475836	2.115701	2.295724	0.771182483	0.800119	0.82407
8	0.000124955	1.635226	2.168829	2.02385	0.714187956	0.816129	0.748609
9	0.000132149	1.555539	2.107155	2.315092	0.681869409	0.88984	0.761194
10	0.000134385	1.439684	1.726368	1.915488	0.790306286	0.760341	0.81087
11	0.000124754	1.413695	2.25282	2.158107	0.645730743	0.814952	0.746407
12	0.000129544	1.385421	1.976684	2.172847	0.600211205	0.758598	0.904795
13	0.000150338	1.428343	1.965474	2.033178	0.612069578	0.712441	0.919121
14	0.000176295	1.239793	1.630675	2.028471	0.599739179	0.663748	0.84734
15	0.000164025	1.168037	1.58979	1.801595	0.362883223	0.745169	0.833516
16	0.000182665	1.171509	1.788816	1.968227	0.249630772	0.74709	0.845974
17	0.000189829	0.897057	1.524511	1.735298	0.021165917	0.55419	0.723251

After the maps were tabulated in the form required, the maps had to be validated against design values. This was done with a simple Modelica model using pressure sources and sinks and a constant rotational speed component to check results on mass flow, and vice versa to check results on pressure difference over the component for compressors. For turbines, the process was similar, but shaft power was also validated using an inertial load to dump the power into to avoid shaft speeds from rising and the model escaping the boundaries of the maps. The results were within a few percent of testing data for these components when within the map’s validity area. Outside of the mapped area, extrapolation causes impossible results, which is expected.

5.1.2 Combustion chamber model

The combustion chamber model included in ThermoPower library is versatile and highly functional as is. A simple unit test was performed with a mass flow source at design state before the chamber and a pressure sink after. The model was found to be working as expected from the start, and further development was not required for this component at this stage. The model calculates heat rate and increases the temperature of exiting gas according to specific heat of the mixture of gases, and the results from this were the same as a conventional Excel calculation when the model is at steady state concerning heat accumulation in chamber walls. However, further development direction was identified, as Aurelia® A400 is a system flexible with fuels, an extended version of the combustion chamber that estimates the heating value of the fuel from its composition and adjusts accordingly could be developed for future use. The model calculates flue gas composition as is but doesn't estimate for heating value out of the box.

5.1.3 Shaft speed controlling generators

At the point where all components were functioning properly in steady state with a constant load, load adjustment functionality was needed. A custom generator model had to be created to be able to control the shaft speed by varying the load. This was done by adding a PID-controller for generator load. The resulting generator model takes the speed setpoint as an input and adjusts the load internally to achieve said setpoint. The basic model diagram for controlled generator is shown below in Figure 4.

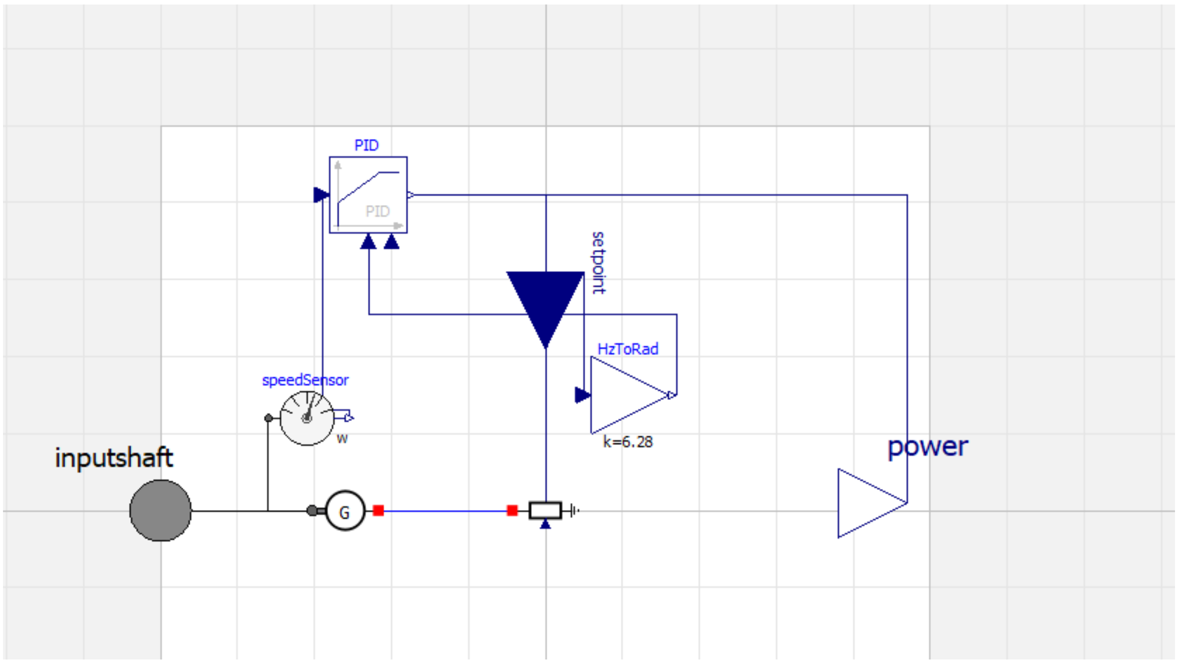

Figure 4:Controlled generator

It includes unit conversion, PID-controller, generator and an electrical load that is adjusted by the PID. The generator power is set up as an output for effortless referring later. The generator model used allows for negative power also, which means this same component is functional for simulating cold runs and spooling up the shafts before ignition.

5.1.4 Intercooler

Neither of the heat exchangers are the focus for this simulation, which means they can be greatly simplified as long as they're within specification. For the intercooler, a model was written where the compressed air's output temperature is set as a parameter, and the model gives estimates for coolant flow based on a simple LMTD calculation. The model has overall heat transfer coefficient, heat transfer area and cooling water inlet temperature as parameters to give an estimate of the required mass flow. An upper limit could be set up for the mass flow to simulate conditions where the intercooler is at its limits and sometimes fails to achieve the set temperature for air.

5.1.5 Recuperator

The recuperator model is a lot more complex, as it transfers heat from the process to the process. The first functioning simplified version was a simple counter current heat exchanger with dimensions, heat transfer models and flow characteristics specified with parameters to function within specifications at design point. Further improvements could be implemented if deemed necessary for off design situations where the simple model loses accuracy. The pressure loss at the recuperator has been implemented as a simple flow dependent pressure loss for the compressed air side. For the flue gas side, the pressure loss is omitted inside the recuperator component and instead a similar resistance is applied in the pressure sink at flue gas exit. Graphical representation in Figure 5: Recuperator

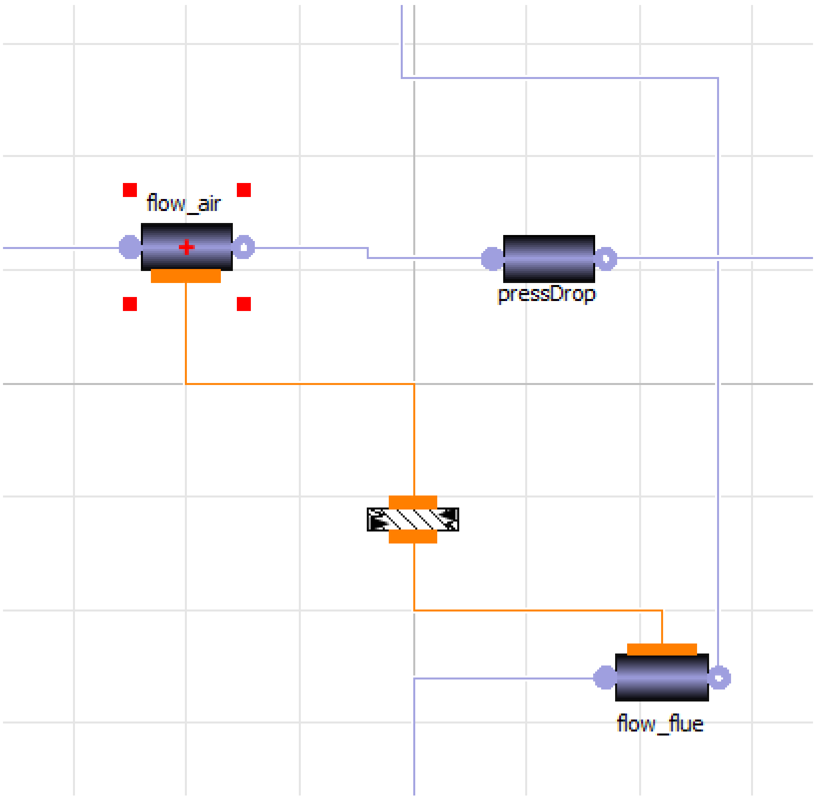

Figure 5: Recuperator

5.2 Integration testing for the model

At the stage where the most important components for Aurelia® A400 were tested and validated individually, their functionality together had to be tested. This phase began with setting up a test simulation with the two turbines in series between a mass flow source and a pressure sink. The simulation ran without major issues. Further testing was done by adding the combustion chamber into this model. Initialisation was rough for the beginning. The newton solver found a valid solution to the model's equation group, but the first 10 seconds of simulation have fluctuating values to find the correct steady-state balance. Tendency to find a numerically correct, but not desired solution was a problem at this stage.

For the compressors, a similar test was conducted, but the compressors proved to be more difficult than the turbines. This was due to an error in the calculation of flow numbers that caused the beta lines to extrapolate into negative region out of the map, even while giving reasonable values for pressure and mass flow. This was corrected, and the two-compressor test model started to give expected results.

The next logical step was to combine the two sides of the system. This proved to be more time consuming than the sides working independently, as enthalpy calculations caused issues where the solver constantly ran into issues. Careful and patient selection and setting of start values and initial equations lead to a desired result. At this point, the model had no intercooler and no recuperator as heat exchanger models were designed after this point, so the band where the model functioned was narrow, and rotational speeds tended to either drop or rise sharply out of the designed operation band and cause the turbo maps to be extrapolated. Extrapolation was however necessary at this point, as without an intercooler, the high-pressure compressor could not function properly, and occasional extrapolation outside the map avoided a crash. Mass flow was determined by a mass flow sink at this point.

Later in the development process, the models for intercooler and recuperator were added, and generator models were added. These latter parts were trivial to integrate, as they already had placeholder components with constant values.

5.3 The full model and its development path

The complete model needed a model for the heat exchangers. The first implementation of the heat exchangers was to use a pipe with a defined wattage of heat extracted/inserted into the flow, like the SimScape example way of implementing a combustion chamber. Proper dynamic models were to be developed later to replace constant value placeholders. This was a successful effort, and the model could be setup to use pressure sinks and sources at both ends. This step was significant, as before this point in development, the model couldn't determine its mass flow correctly without external definition. This is significant, because changes in mass flow have significant effect on turbomachine behaviour, and with a defined mass flow, there is no absolute need for the components and their mapping to work together.

The next significant development version had generators and electrical loads implemented. This marks the first point in development where the model outputs meaningful results, as running the model at self-consumption fuel rate, where no power was generated. This idle state does not give a chance to validate temperatures and pressures at various states, and these are what are needed for control software development. The values at this point were not accurate, but the dynamic behaviour resembled the real machine. Adjusting ambient conditions and the amount of fuel causes similar changes in values as with the real machine, effects are in the correct direction and close in magnitude. From this point onwards, the necessary valves and pressure losses had to be implemented to aim towards an integration test with TwinCat to control the physics model with the old automation software.

The next two remarkable development versions included a simplified intercooler with air exit temperature set at the desired value and controlled generators to control shaft speeds. Temperature and pressure values could be validated at this point, and the results were reasonably close to actual considering the model complexity at that stage. Temperatures were off by a maximum of 50 K for nominal point, pressures had varying error, mostly due to pressure losses not implemented. Pressures were within 0.4 bar from the actual values at points where a significant pressure loss was not omitted at this point. The following major step was the inclusion of a recuperator model that functioned within specification. At this point the model is ready for integration with TwinCat to test their interfacing. Model diagram for the full model is shown in Figure 6.

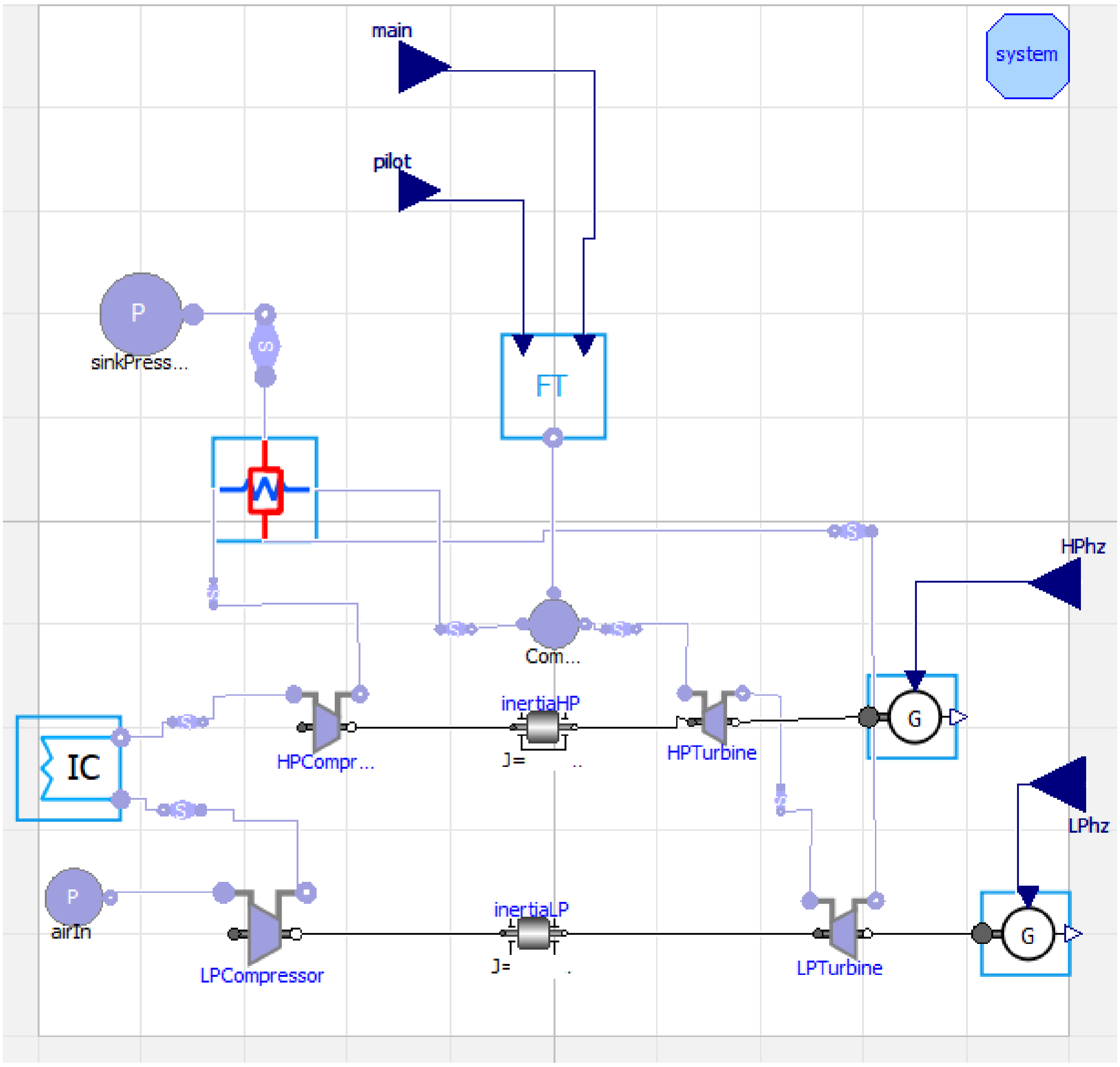

Figure 6: Full model diagraml

Eventually, a complete model with a clean initialisation, proper inputs, simulated fuel train and good numerical stability was formed from iterative development against the automation software. Improvements on stability became gradually when something the automation did caused a crash, and the previously unknown issue was then corrected by altering parameters, start values or functionality to better sustain the occasionally abrupt input changes. The parameters that were the most adjusted, were the PID parameters in controlled generators and flow characteristics in the recuperator. This process also included adding damping to avoid sudden value changes. The model's initialisation also improved from these adjustments to allow for more starting points that reach steady state reasonably fast without risk of a simulation runaway caused by a sharp input change. A runaway in this context means a situation where for example the LP turbine unit's shaft speed escapes control as the controlled generator is not fast enough to increase load sufficiently. The last development version of the model within this thesis is very usable for the initial use of automation testing, and in addition, meaningful process engineering tasks can be done. The model is equipped with two alternate fuel train options, that can be swapped by setting a boolean value to either false or true before starting the simulation.

6 TwinCAT communication

Basic functionality required for the automation testing is as follows:

- Four control inputs from TwinCAT: main fuel valve, pilot fuel valve, and shaft speed control for both shafts. A bleed valve will be added later.
- Enough numerical stability to not crash or get stuck with inputs controlled by the old automation software solution.
- Enough performance and stability to be able to real-time simulate with reasonable timesteps.
- Connectivity to swap numerical values multiple times a second between TwinCAT and simulation.

6.1 Options for communication implementation

Two equally reasonable options for implementing the communication were considered. The first option was to use Beckhoff's Target for FMI addon for TwinCat. Target for FMI is used to generate a TcCom object that can be executed within the TwinCat runtime. The benefits of this option are simple transferability and easy usage for testing purposes, as variable linking and updating values once per cycle is a lot easier with an inbuilt solution compared to the second option, which was creating middleware to handle the initialization and communication between the two systems.

Initial choice was the Target for FMI. However, due to unforeseen compatibility issues, a backup plan of writing a Python program to handle this task was activated in the meantime. Initially the purpose of the simplified Python solution was to figure out what variables in the old automation need to be linked and to research fault states and how to avoid triggering a fault in the automation code. During the researching process, the Python solution evolved into a capable piece of simulator software, that enables an extremely high amount of control over simulation and communication and the possibility to use various solvers with Model Exchange FMUs.

6.1.1 Target for FMI

Target for FMI works by using the source code from an FMU and compiling an object with its internal build instructions. This way, the model's behaviour is encapsulated into an object, that can be ran even on an actual PLC if required. (Beckhoff Automation GmbH, 2022)For a full model, running on a PLC makes little sense, but for predictive purposes, a future use for partial models does exist. This conversion into a TcCom-object is delicate, and multiple issues were found when implementing a simple turbine model. The modelling tool OpenModelica and Beckhoff's object converter were not readily compatible in certain situations, and figuring out a tool chain that works completely as expected was not achieved within a reasonable timeframe. During this research phase, the alternative branch of development with Python gradually became the main branch.

6.1.2 Python with FMPy and pyads

Using Python with pyads and an FMI library was tested in the very early phases of the project, and it was found to be possible but work intensive compared to direct Target for FMI implementation. Pyads is a wrapper library for Beckhoff's ADS commands from their DLL-file. In a similar fashion, FMPy is mostly a wrapper for C-functions described in the FMI standard bundled with some solvers and practical features. Documentation for these libraries is available online (Lehmann, 2022) (FMPy, 2023). Essentially the Python-based implementation is running two C-wrappers and handling data between them. The estimated work intensiveness comes from the fact, that timing and synchronising must be done with Python instead of nearly automatic solution that Target for FMI has. One major consideration against using Python for this project was performance, as efficient automation requires fast communication without delays. However, gradual performance improvements and optimisation proved that very sufficient performance is reachable. The model and communication can be run on relatively weak hardware while still maintaining real-time simulation.

6.1.3 Further information on Beckhoff ADS

Beckhoff Automation Device Specification (ADS) is the communication protocol used by TwinCat. ADS can be used to exchange data between TwinCAT runtime and external systems, and it provides access into the automation system. Beckhoff offers ADS communication tools for common programming languages like C/C++, Java and .NET, of which the C/C++ implementation is used in this thesis via a Python wrapper. In addition, communication interfaces are also provided for various external software options like LabView, MATLAB and Office. (Beckhoff Automation GmbH, 2023)

6.2 Development of the middleware TriCat

The middleware solution got named TriCat, for its task of connecting a cat-named development version of the physics model with TwinCat. The base principle for the basic communicator middleware is the following:

1. Load the Functional Mock-up Unit.
   1.1. Unpack the FMU, fetch model description for variable information.
   1.2. Initialize the FMU with start value parameters to a point where first timestep can be calculated.
   1.3. Pass the prepared FMU object and a dictionary of variables forward.
2. Connect to TwinCat.
   2.1. Initialize a pyads connection to the desired PLC with AMS Net ID of the local runtime.
   2.2. Pass the connection object forward.
3. Variable setup.
   3.1. Use a dictionary with TwinCatName:FmuName key-value pairs to establish which variable corresponds to which. The input dictionary is used to form a list of value

references to fetch from the FMU during communication step. List of names and value references are passed forward.

3.2. This setup is done every run, because swapping the FMU might alter the value references when variables are added. Setting this up automatically by variable name instead of saved value references makes it effortless to update the underlying FMU.

4. Initialize automation.

4.1. Use an initialization dictionary to set variables in PLC to a suitable state where automated control is possible without the automation or the FMU faulting.

4.2. This includes setting faultbits, runbits and certain variables that are not simulated in the model, for example gas line pressures and safety variables.

5. Simulation loop.

5.1. Run timestep.

5.1.1. Fetches values and derivatives from the FMU and calculates the next state.

5.1.2. Repeat for suitable number of times for a reasonable communication interval without falling behind real-time.

5.2. Exchange data.

5.2.1. Form a dictionary to write to PLC from lists from Initialize Automation

5.2.2. Add other values that need to be updated but are not direct transfers from FMU into the dictionary. Do preprocessing and write the dictionary to PLC. Good point to print variables like simtime and difference to real time to monitor simulation speed.

5.2.3. Read the control inputs from PLC.

5.2.4. Write the control inputs from PLC to the FMU.

The above simple structure for a TwinCat communicator is visualised in Figure 7.

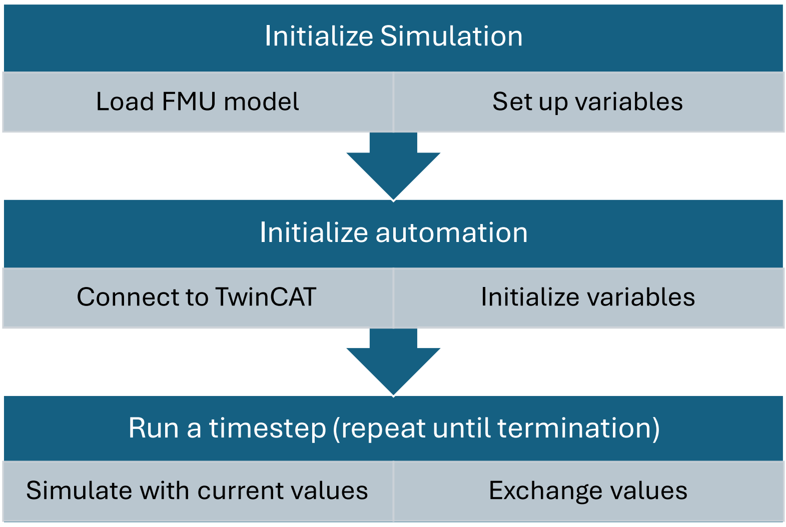

Figure 7: Communicator flowchart

As is visible from Figure 7, this simple implementation must wait for data exchange to get new inputs to calculate the next timestep. Practical testing showed that multiple timesteps of the simulation must be ran with the same input data to stay in real-time. In essence, communication is not fast enough to interchange data in the same process for each timestep. A numerically stable timestep is too small to stay in real-time and communicate at every step. To reduce this instability caused by this, a ramping function was written to gradually adjust the input value step by step between communication intervals. The ramping function works by checking if the automation input is within allowable single step adjustment, and if not, it adjusts the value used in FMU by the maximal allowed adjustment per step. Simple implementation like this with simple forward Euler as the solver can communicate with TwinCat without major issues and serves as a sufficient proof of concept for implementing communication using Python. Solution at this stage can’t reliably handle the abrupt input changes that the automation outputs due to too long communication interval and Euler not being the ideal solver for numerically stiff situations.

6.2.1 Selection for a better solver

Initially, Forward Euler was used for solving the FMU, but a better solver was needed eventually. Forward Euler is a simple and efficient way to calculate new states for the FMU, but it has major drawbacks due to its simplicity. Forward Euler works by adding derivative multiplied by timestep to the value of a variable and then the state is saved for next step.

$$y_1 = y_0 + dt * y_0' \tag{2}$$

where:	y_1	value after timestep
	y_0	value before timestep
	dt	timestep
	y_0'	derivative of y before timestep

For this reason, the simulation is prone to enter a non-solvable state if something changes fast and the timestep is large enough. For gas turbine modelling, using Euler solving is viable for steady-state and slow adjustments, but the limitations are met when fast adjustment to main fuel valve position is done by automation. Having extremely small timesteps and frequent communication interval improves the capability of this solver, but computational expense gets unreasonable before reliable usability is achieved.

Both Co-simulation and Model Exchange FMUs are usable with this approach. Simple forward Euler is available as a solver in both. For this project, Model Exchange was a better approach, as having an external solver facilitates greater step by step control, when with a Co-simulation FMU the solver resides inside the FMU and control is limited by what the FMU generating tool has implemented. Especially this application, where information is exchanged between steps, benefits from the added precise control achieved by using an external solver. Co-simulation is easier to use, as setting up an external solver more complicated than Euler takes some time and trial and error.

After the initial phase with using Euler for integration, a need for a more sophisticated solver was identified, as stability in stiff situations was less than needed. For consistency between running the FMU and the model which it is generated from, CVODE was chosen as the solver. CVODE is short for C-language Variable-coefficients ODE solver, and it is a part of SUNDIALS (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers)(Hindmarsh *et al.*, 2005, 2024; Gardner *et al.*, 2022). CVODE uses multiple kinds of solving methods depending on situation and it is very sophisticated compared to using Euler method. OpenModelica has a CVODE implementation and using the same solver in there and with FMU enables similar results, similar stability, similar adjustability in both implementations. This enables testing a model in OpenModelica with similar performance compared to using an FMU. Correcting numerically difficult situations is straightforward when testing can be done mostly in the modelling environment. OpenModelica uses DASSL as its default solver, and it is slightly more capable than CVODE for this application, but setting up DASSL was deemed not desirable due to FMPy having a ready-made CVODE implementation. The readily available solver worked extremely well for running the gas turbine simulation for defined number seconds and plotting the results afterwards. Working with it stepwise requires a deeper dive into the lower-level control of the solver and the FMPy library to achieve proper initialization and proper step by step simulation. Using lower-level methods within the library gives absolute control over the simulation with the sacrifice of simplicity and ease of coding. These include running the simulation step at a time and setting input values in a custom way.

CVODE is fast, even with fixed step where it doesn't have the ability to adjust step size based on numerical stiffness of the equation group to be solved. Normally CVODE can adjust its step size as needed. This was disabled for easier development. TwinCat communication with both FMU reads and writes is not fast, as reading and writing locks the FMU object for the duration of the transaction, and the simulation is then waiting for the communication to finish before doing the next step. During the time used for communication and reads and writes, the simulation could advance multiple steps. Having both communication and simulation in the same loop and on the same thread causes major performance loss due to waiting.

6.2.2 Restructuring TriCat for performance

Communication interval of 100 ms was the lowest reachable without having either an unstable long stepped simulation or slowdowns below real-time during difficult situations. The need to wait for communication step and having the FMU-object locked during a communication step was a major performance drawback. Restructuring the program to have classes instead of functions and setting them up in Python’s multiprocessing led to a remarkable performance improvement. The parallelised structure is as follows:

1. Simulation process
    1.1. Load the FMU and send variable dictionary to a shared dictionary.
    1.2. Read the shared FMU value reference list, read the variables in reference list, and save them into a shared output data list.
    1.3. Read the shared input data list, do a timestep and update the shared output data list.
    1.4. Repeat 1.3 as fast as possible but wait if ahead of real-time.
2. Communication process
    2.1. Establish a connection to the PLC and use the shared dictionary to setup variables, save FMU value references to a shared list and Initialize the PLC.
    2.2. Read the shared output data list and write the data to the PLC. Read input data from PLC and save it in shared input data list.
    2.3. Repeat 2.2 as fast as possible.

The two processes both have their own task to work with, and PLC connection object and FMU objects reside in their respective processes. The processes are run in parallel, and each process updates their own shared list to avoid writing race conditions. After restructuring, the program is capable of 60 communication cycles and upwards of 200 simulation time steps per second compared to 5-10 communication cycles and 100 steps per seconds that was reachable with a single thread. This approach also enables fresher data to be always used. The improved program structure is visualised below in Figure 8, running left to right and brackets to visualise looping until termination. A larger version of this figure is included as an appendix. Thicker arrows are used to differentiate between instantiating and starting and simple data exchange which is a thinner line.

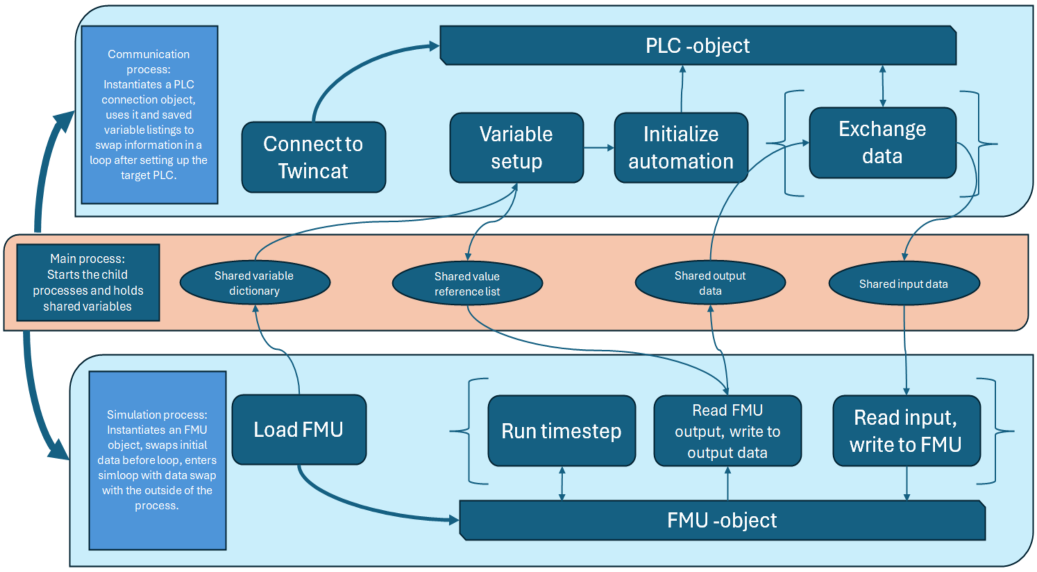

Figure 8: TriCat basic structure

During the addition of multiprocessing, a simple GUI was also added for monitoring and controlling the simulation. A simple early GUI is shown in Figure 9.

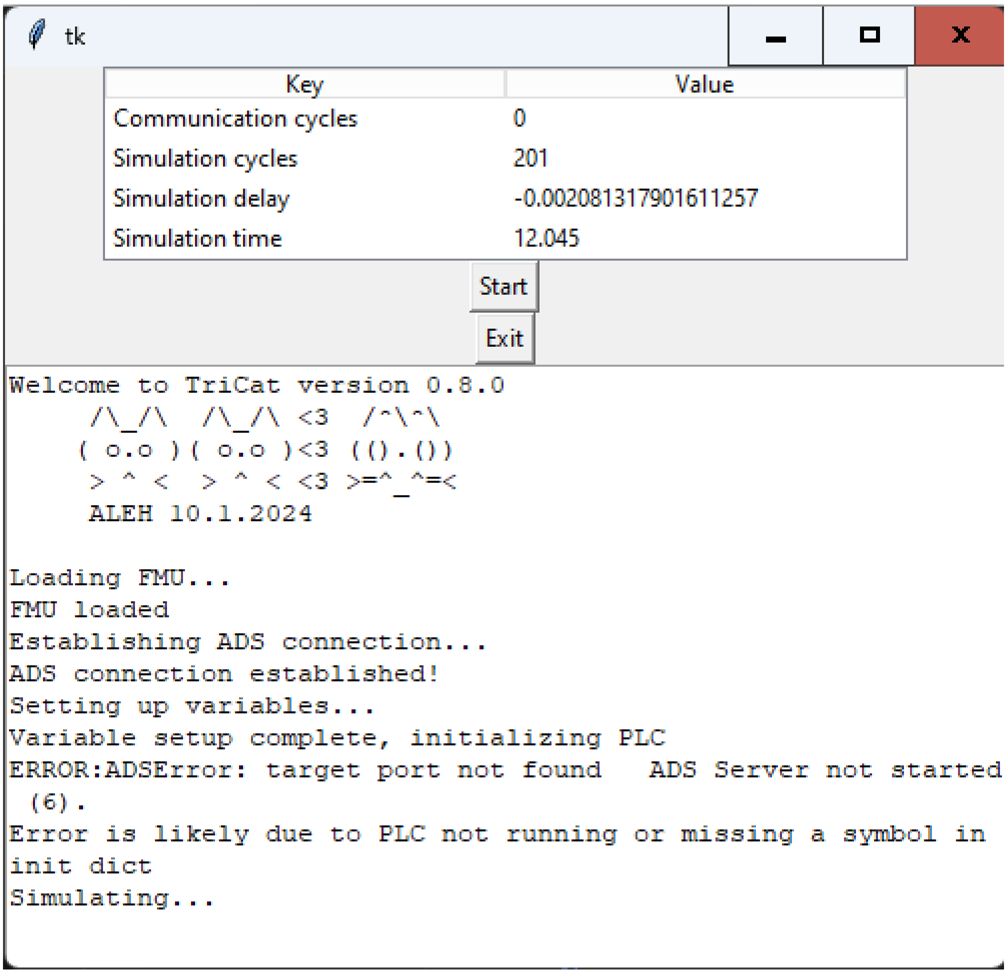

Figure 9: Graphical user interface

The most important performance values are shown in the upper section, and text output is shown in the lower section. The screenshot is from a situation where the target PLC is not running, and therefore the communication process runs into an error. Simulation process is not critically tied to communication and is running at 201 steps per second and slightly ahead of real time with default start input values. Negative delay means that the software must wait to advance to the next step to stay in real-time. Simulation can be run without communication if so desired, and if one of the processes crashes, the other will not crash because of it.

6.3 Final solution and its features

The final implementation of TriCat within the scope of this thesis is structured to stay running while the simulation crashes. Encapsulating the error prone simulation part into its own process allows the graphical user interface and communication to run independently. This choice was made because restarting the simulation after a crash or a solver failure is essential for a pleasant user experience. TriCat encapsulates the simulation within a separate simulation process that can be restarted at will from the graphical user interface. A simulation process crash is detected by the main process as the simulation and communication cycle speeds are also used to determine if the subprocess is still alive. The program is easy to use. Setting the correct AMS Net ID address for ADS and starting the program are enough to get simulated output into TwinCAT from the model. User can choose between different sets of starting values and adjust input ramping parameters. Starting values also contain the choice of fuel train option.

7 Capabilities of the model and future advancements

Digital twin project started in this thesis is in the following state at thesis completion:

- Functional accuracy of the model is reasonably good, actual values for temperatures and pressures behave as intended, but are not accurately representative of an actual machine. This is due to mapping inaccuracy and differences between various configurations of the actual machine.
- Model performance and stability are more than sufficient near the operating point, and a 15% of rotational speed range is tolerated in the higher end and results are sensible. An experimental version with wider band and less accuracy was developed near the end of thesis writing.
- Interlinking between TwinCat and the physics model is usable without knowledge about the inner workings of the model or TriCat.
- Prepared for ML implementation: TriCat has access to turbomachine maps used by the FMU. A test data analysis script that gives an average mapping from test data was written during test implementing expanded turbo maps. A machine learning solution could be inserted here instead of manually adjusting CFD based maps to represent actual data.

The combination of the model and the communication software have various limitations. The most meaningful limitations of the model are limited rotational speed range and lack of thermal inertia and heat soakage. Heat soakage is heat soaking into the machines metal parts during transient states affecting performance (GSP, 2023). The thermal features are not simulated, and for a functioning warm-up and startup sequence these are essential. The more severe limitation of limited rotational speed range has a known cause of conflict in maps when extrapolating, and a plan is devised to increase the range, ideally to a situation where the gas turbine can be simulated from a standstill to 10% overspeed.

The communication program TriCat is limited on its logging capability, as logging per step with a lot of information that the FMU outputs lowers simulation speed below real time or timestep must be increased. Implementing logging features with reasonable and required information is a future target to monitor performance and identify situations that cause slowdown. Currently real-time monitoring of performance values shows that occasional slowdowns happen, and logging performance values would enable problem identification when combined with automation logs.

The model is usable for its initial most immediate use case. Future development directions to be researched that are outside the scope of this thesis are implementing better transient simulation with heat soakage calculation, wider rotational speed range, machine learning for mapping, and general improvements on how the model is implemented.

A major future feature for the model is the capability to have fuel composition as an input and adjust heating value accordingly. This feature can then be used to simulate varying fuel composition and its effect on machine behaviour to enhance multifuel features of the actual machine. Functionality to have fuel composition affect the flue gas composition is useful for excess air calculations, sulphur calculations and many other aspects of fuel flexibility. Simulated fuel flexibility enables fast research on machine behaviour in these situations and enables studying hypothetical situations effectively.

Further configuration options in addition to the choice of fuel train could be implemented, for example different variations of intercooler and recuperator could be developed with varying simulation complexity. For testing recuperator behaviour in different conditions, the current model is not representative enough.

8 Conclusion

Developing a model to be used as a base for future digital twin endeavours and automation design is a complicated process with many variables, 3000 to be exact. To achieve a model and integration with TwinCat that both work as expected requires a considerable amount of trial and error and solving and circumventing unforeseen issues in creative ways.

Developing a dynamic model for a new kind of complicated machine that has never been modelled in this manner shows a myriad of interesting numerical difficulties, that are not present in simpler gas turbine models. Gas turbines in general have been modelled before, but this particular design has specific challenges due to a complicated process and high adjustability. Composing four different turbomachine maps that agree on mass flow and performance with the strong fluid dynamical link between stages and strong mechanical link between compressor and turbine sides is a difficult task in itself, and for further accuracy and usability, the maps modified from readily available mapping data must be expanded towards lower rotational speeds by using test data to get sufficient start-up and transient simulation.

Finding a way to exchange data from the physics model to the PLC in a controlled, reliable, fast and easy to use manner proved to be a complicated, as a readily made solution is not available. Self-made interfacing program TriCat was initially developed for simple steady-state testing of automation behaviour and solving automation solution specific issues while waiting on TwinCAT technical support to advance using FMUs directly within TwinCAT. TriCat eventually became the main research branch as the program gradually improved far beyond expectations, and performance required for real-time simulation and data exchange was exceeded. The interfacing might be switched to a more integrated Target for FMI solution in the future, but the TriCat approach is more than sufficient for the scope of thesis and current needs.

The goal of this thesis was to develop a model that can be used as an aid for automation development with future development towards other uses in mind. This main goal was achieved, and new automation features and adjustments can be preliminarily tested on the model.

If machine learning and other AI technologies are successfully applied to this model and communication infrastructure, the result could be significant for digital twins and AI-assisted models in the field of gas turbines. Especially in a situation where a lot of identical machines exist in various operating conditions, using data-driven approach could detect abnormalities that are missed by conventional alarms and detection methods. This enables maximal reliability and performance. The special nature of the Aurelia® A400 as a small and flexible unit combined with the power of AI technology could lead to efficient predictive control in a situation where both the need of electricity and quality of the fuel are both fluctuating.

References

Altair (2023) *Altair Twin Activate*, *Default*. Available at: https://altair.com/twin-activate (Accessed: 10 October 2023).

Ansys (2021) *Ansys Twin Builder Data Sheet*. Available at: https://www.ansys.com/content/dam/product/digital-twin/twin-builder/ansys-twin-builder-technical-datasheet.pdf (Accessed: 10 October 2023).

Attaran, M. and Celik, B.G. (2023) 'Digital Twin: Benefits, use cases, challenges, and opportunities', *Decision Analytics Journal*, 6, p. 100165. Available at: https://doi.org/10.1016/j.dajour.2023.100165.

Aurelia Turbines (2023) *Aurelia A400 Datasheet*. Available at: https://aureliaturbines.com/files/sites/775/230828-datasheet-a400-iso-eu-370ccbc8-71b3-4f6a-88e1-f637ed0f26f6.pdf (Accessed: 10 October 2023).

Beckhoff Automation GmbH (2022) *TwinCAT 3 Target for FMI*, *Beckhoff Automation*. Available at: https://www.beckhoff.com/fi-fi/products/automation/twincat/texxxx-twincat-3-engineering/te1420.html (Accessed: 18 January 2024).

Beckhoff Automation GmbH (2023) *TwinCAT 3 ADS*, *Beckhoff Automation*. Available at: https://www.beckhoff.com/fi-fi/products/automation/twincat/tc1xxx-twincat-3-base/tc1000.html (Accessed: 5 March 2024).

Casella, F. (2014a) *ThermoPower*. Available at: https://casella.github.io/ThermoPower/ (Accessed: 23 January 2024).

Casella, F. (2014b) *ThermoPower Documentation*, *ThermoPower Documentation*. Available at: https://build.openmodelica.org/Documentation/ThermoPower.html.

Collimator (2024) *Home | Collimator Docs*. Available at: https://docs.collimator.ai (Accessed: 8 April 2024).

Dassault Systemes (2023a) *Dymola*, *Dassault Systèmes*. Available at: https://www.3ds.com/products/catia/dymola (Accessed: 8 April 2024).

Dassault Systemes (2023b) *Open Source | Scilab*. Available at: https://www.scilab.org/about/scilab-open-source-software (Accessed: 10 October 2023).

Elmqvist, H., Boudaud, F., Broenink, J., Brück, D., Ernst, T., Fritzson, P., Jeandel, A., Juslin, K., Klose, M., Mattsson, S.E., Otter, M., Sahlin, P., Tummescheit, H. and Vangheluwe, H. (1997) 'ModelicaTM - A Unified Object-Oriented Language for Physical Systems Modeling'.

FMPy (2023) *FMPy documentation*. Available at: https://fmpy.readthedocs.io/en/latest/ (Accessed: 18 January 2024).

Gardner, D.J., Reynolds, D.R., Woodward, C.S. and Balos, C.J. (2022) 'Enabling New Flexibility in the SUNDIALS Suite of Nonlinear and Differential/Algebraic Equation Solvers', *ACM Transactions on Mathematical Software*, 48(3), pp. 1–24. Available at: https://doi.org/10.1145/3539801.

GE Digital (2022) 'Autonomous Tuning for Aeroderivative Turbines'. Available at: https://www.ge.com/digital/sites/default/files/download_assets/autonomous-tuning-gas-turbine-from-ge-digital-datasheet.pdf.

GSP (2023) *Heat soakage effects*. Available at: https://www.gspteam.com/GSPsupport/OnlineHelp/heat_soakage_effects.htm (Accessed: 6 November 2023).

Hindmarsh, A.C., Brown, P.N., Grant, K.E., Lee, S.L., Serban, R., Shumaker, D.E. and Woodward, C.S. (2005) 'SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers', *ACM Transactions on Mathematical Software*, 31(3), pp. 363–396. Available at: https://doi.org/10.1145/1089014.1089020.

Hindmarsh, A.C., Serban, R., Balos, C.J., Gardner, D.J., Reynolds, D.R. and Woodward, C.S. (2024) 'User Documentation for CVODE'. Available at: https://sundials.readthedocs.io/en/latest/cvode.

Idebrant, A. and Näs, L. (2003) 'Gas Turbine Applications using ThermoFluid', in. Available at: https://api.semanticscholar.org/CorpusID:14709042.

Lehmann, S. (2022) *pyads 3.3.9 documentation*. Available at: https://pyads.readthedocs.io/en/latest/ (Accessed: 18 January 2024).

Lim, J., Perullo, C.A., Milton, J., Whitacre, R., Jackson, C., Griffin, C., Noble, D., Boche, L., Seachman, S., Angello, L., Maley, S. and Lieuwen, T.C. (2021) 'The EPRI Gas Turbine Digital Twin – a Platform for Operator Focused Integrated Diagnostics and Performance Forecasting', in *Volume 4: Controls, Diagnostics, and Instrumentation; Cycle Innovations; Cycle Innovations: Energy Storage; Education; Electric Power*. *ASME Turbo Expo 2021: Turbomachinery Technical Conference and Exposition*, Virtual, Online: American Society of Mechanical Engineers, p. V004T09A009. Available at: https://doi.org/10.1115/GT2021-59572.

Link, K., Gall, L., Bonifay, J. and Buggert, M. (2014) 'Testing Power Plant Control Systems in Modelica', in. *the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, pp. 1067–1072. Available at: https://doi.org/10.3384/ecp140961067.

Modelica Association (2022) *Release of FMI 3.0 - Functional Mock-up Interface*. Available at: https://fmi-standard.org/news/2022-05-10-fmi-3.0-release/ (Accessed: 12 January 2024).

Modelica Association (2023) *Tools - Functional Mock-up Interface*. Available at: https://fmi-standard.org/tools/ (Accessed: 10 October 2023).

Modelica Association Project FMI (2022) *Functional Mock-up Interface Specification*. Available at: https://fmi-standard.org/docs/3.0/ (Accessed: 12 January 2024).

Open Source Modelica Consortium (2023) *Open Source Modelica Consortium*. Available at: https://openmodelica.org/home/consortium/ (Accessed: 25 October 2023).

OSMC (2023) *OpenModelica User's Guide — OpenModelica User's Guide v1.23.0-dev-83-ga073984273 documentation*, *OpenModelica User Guide*. Available at: https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/index.html (Accessed: 25 October 2023).

Panov, V. and Cruz-Manzo, S. (2020) 'Gas Turbine Performance Digital Twin for Real-Time Embedded Systems', in *Volume 5: Controls, Diagnostics, and Instrumentation; Cycle Innovations; Cycle Innovations: Energy Storage*. *ASME Turbo Expo 2020: Turbomachinery Technical Conference and Exposition*, Virtual, Online: American Society of Mechanical Engineers, p. V005T05A010. Available at: https://doi.org/10.1115/GT2020-14664.

Robb, D. (2024) 'Digital Twins in the Real World', *Turbomachinery Magazine*, 65(1). Available at: https://www.turbomachinerymag.com/view/digital-twins-in-the-real-world (Accessed: 7 March 2024).

Singh, M., Fuenmayor, E., Hinchy, E.P., Qiao, Y., Murray, N. and Devine, D. (2021) 'Digital Twin: Origin to Future', *Applied System Innovation*, 4(2), p. 36. Available at: https://doi.org/10.3390/asi4020036.

Yu, Y., Chen, L., Sun, F. and Wu, C. (2005) 'Matlab/Simulink-based simulation for digital-control system of marine three-shaft gas-turbine', *Applied Energy*, 80(1), pp. 1–10. Available at: https://doi.org/10.1016/j.apenergy.2004.03.006.

Appendix 1: Enlarged Figure 8