LARGE LANGUAGE MODEL IN AIRCRAFT SYSTEM DESIGN

Petter Krus

Department of Management and Engineering, Linköping University
58183, Linköping, Sweden

Abstract

The introduction of large language models in the form of ChatGPT at the end of 2022, marked the beginning of the widespread democratization of artificial intelligence. This is spawning a entirely new area of research into application of this technology. This paper presents a novel approach to generate configuration rules, and design concepts using ChatGPT. The objective is to demonstrate how large language models models can aid in automating the engineering design process. In this paper the configuration of aircraft hybrid propulsion systems is studied as an example. It is shown how a system configuration can be generated and presented in the form of UML component diagrams. Large Language Models usually have an element of randomness inserted in their response. This makes them inherently non-deterministic, and the result is not always correct. Therefore, statistical properties are studied such that the probability of getting a correct result can be estimated, and prompts can be tweaked to provide the best result. This is particularly useful when a prompt is separated into a general part that can be reused for similar tasks, and one specific part.

It is also shown how these models can used in consecutive step to formulate a simulation model and execute it within the framework of ChatGPT-4, or be exported to more dedicated simulation models for more advanced simulations. Also optimization can be included. The study shows promising results, and it shows that there is an immense potential for AI in engineering system design, and we are just in the beginning of applying this tools.

Keywords: Large language models, aircraft design, aircraft systems, propulsion systems, modelling and simulation

1. Introduction

Large Language Models (LLMs) have shown a great capability as few-shot or even single shot learner [3]. This has opened new avenues in various fields, including engineering system design [4]. This paper explores the integration of LLMs in the architectural design of fluid power systems used in construction machinery. The primary contribution of this paper is demonstrating how LLMs can be employed to transform textual inputs into formal, architectural definitions for system design.

Design rules are fundamental in engineering disciplines, serving as the backbone in Knowledge based engineering (KBE) for creating complex systems. While traditional methods for generating design rules are effective, they can be time-consuming and labor-intensive to set up. This paper aims to explore the use of ChatGPT for automating this process.

Configuration rules are fundamental in engineering disciplines, serving as the backbone for creating complex systems that are both functional and reliable. Automating the process of generating these rules can significantly streamline the design phase, reducing the potential for human error.

The introduction of configuration rules means that the design space can be drastically restricted by removing the unwanted parts, thereby making the conceptualization much more efficient, by automatically suggest a concept from a set of requirements. For this, tools like ChatGPT offer promising potential for such automation.

ChatGPT excels in understanding natural language queries and generating human-like text responses. It has a strong ability to interpret complex queries and provide detailed, coherent answers. ChatGPT

can generate and understand code in various programming languages, including Python. This is particularly relevant in this paper where there is a focus on generating UML diagrams and Python code for system designs. ChatGPT can learn from examples and refine its outputs based on user feedback. This means that we can have pre-defined text that can be used as part of a more specific prompt. It is also possible to upload files to customized GPT for specific purposes.
Here, we explore the feasibility and methodology of using ChatGPT to automatically generate design rules, with an aircraft propulsion system as a case study. Through this example, we can go through the process of defining configuration rules, translating them into UML diagrams, and ultimately generating Python code that can produce generic design diagrams for varying system configurations.

2. Generating System Architecture

In the literature a range of concepts can be found. Here we used [6], [5], [1], [2]. These form the basis to create a set of instructions for the Large Language Model, in this case ChatGPT-4o. Here is and example for generating a system architecture for a hybrid propulsion system. First we can just ask ChatGPT-4o to generate code for UML component diagram for PlantUML using the following prompt.

User:"Generate a hybrid electric aircraft system with a fuel cell. Show the result as code for a UML-component diagram in PlantUML" " PlantUML have a straightforward textual format that can be conveniently visualized.

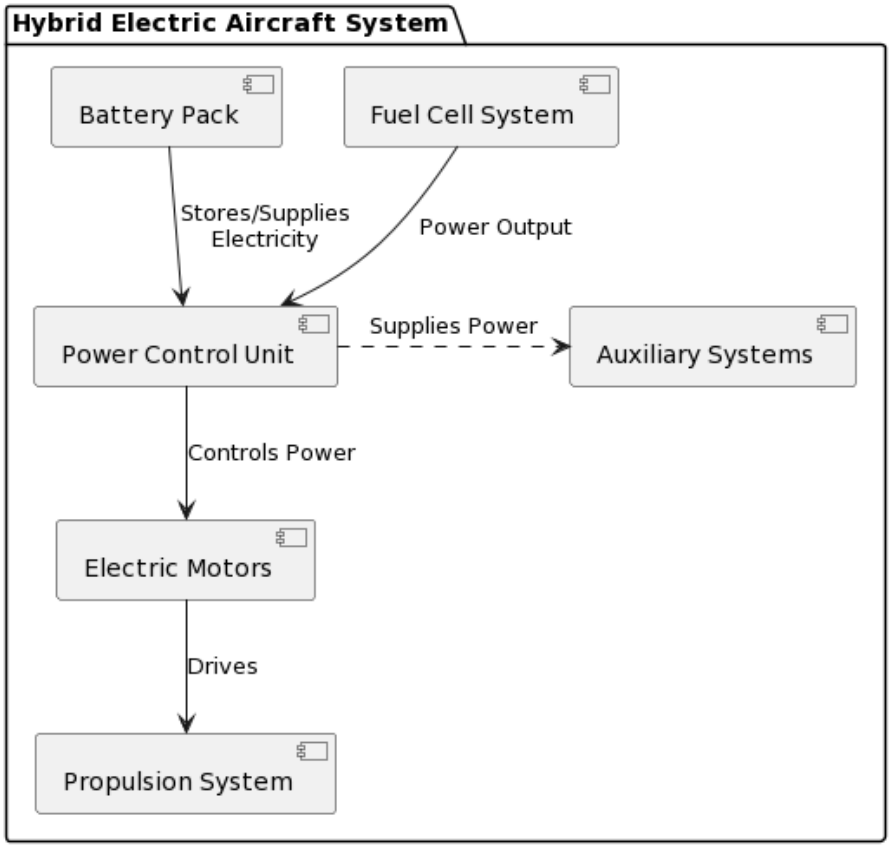

Figure 1 – Hybrid propulsion system with fuel cell. This diagram is generated with a single prompt in ChatGPT-4

Even though it is impressive that it provide a solution there are some things we would like to make more well defined. E.g. what type of information that is transferred between the component. Also there are different styles that the diagram can be generated in, and they may not look the same from time to time. Furthermore, there might be more serious errors, components missing or not connected. E.g., In this case there is no hydrogen tank connected to the fuel cell. In order to make the process more reliable a pre-defined reusable prompt can be uploaded first. This prompt can also be part of a

Custom GPT, which is a facility provided as part of ChatGPT-4 subscription. This is a way in which specific information can be uploaded that make the Custom GPT more qualified to solve specific tasks within a field.
An example of such a prompt is shown in Appendix I.

2.1 Prompt Engineering

Here is an example of a prompting to generate a system definition as UML code (for PlantUML), after the prompt in Appendix I has been uploaded. A key feature in this prompt is the use of micro-templates that shows examples of how components can be defined. Then this prompt is added:

User: "Generate a hybrid aircraft concept with fuel cell. Use the uploaded prompt and represent the system as UML code."

The resulting diagram is shown in Fig. 2

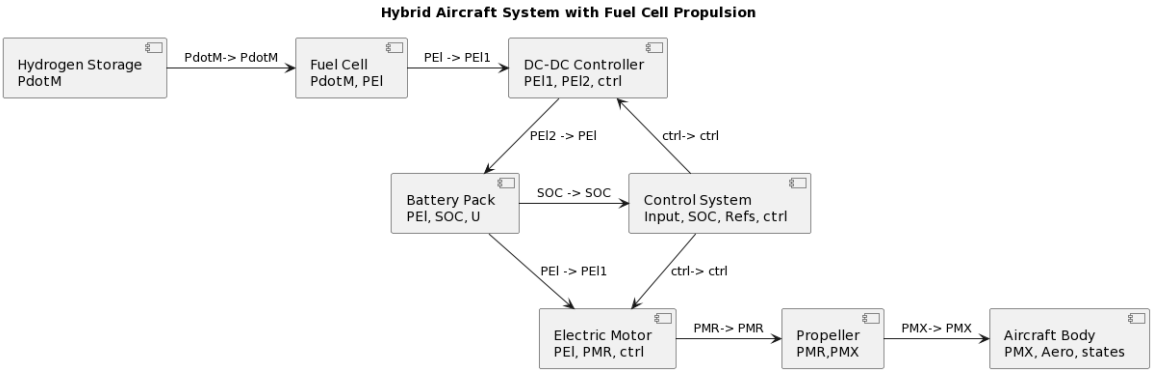

Figure 2 – Hybrid electric aircraft propulsion system with fuel cell

Modifying the system to have two motors we simply ad this prompt:

User:"Modify the system to have two motors and batteries"

The resulting diagram is shown in Fig. 3

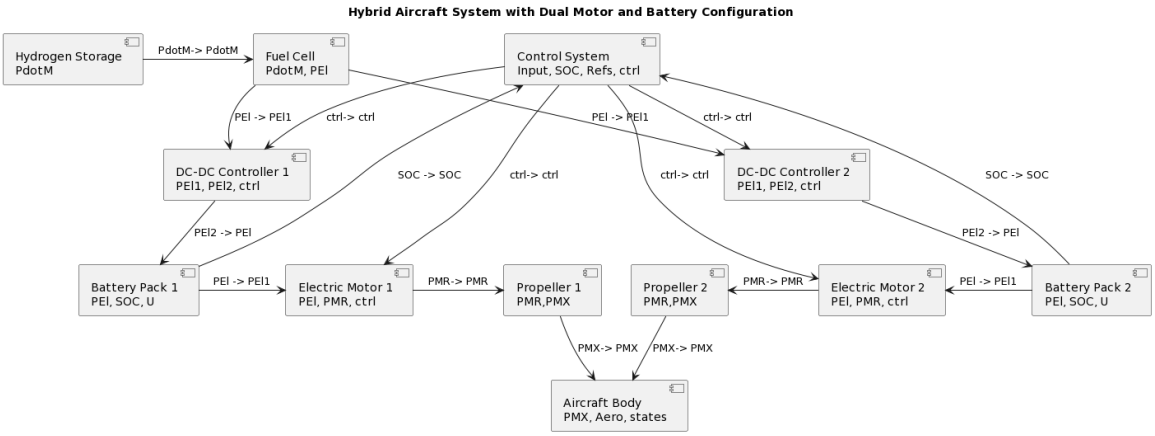

Figure 3 – Aircraft propulsion system with two electric motors.

Of course the Language model is not restricted to the information it has been fed with in the prompt. We can also ask it to ad auxiliary systems to the fuel cell such as "Air Supply" and "Cooling System", and it incorporate it in the right place, see Fig. 4.

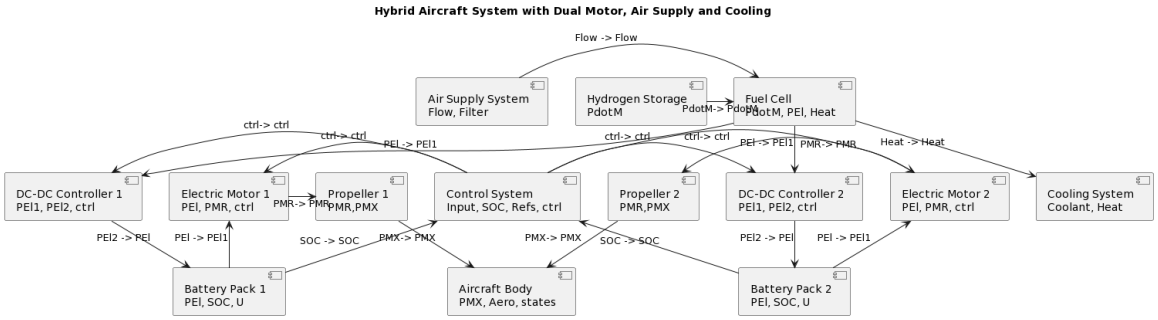

Figure 4 – Hybrid electric aircraft propulsion system with fuel cell with auxiliary systems, air supply and cooling system.

Then we could try to make a system with a gas-turbine driven generator instead of a fuel cell. This will yield in the diagram in Fig. 5

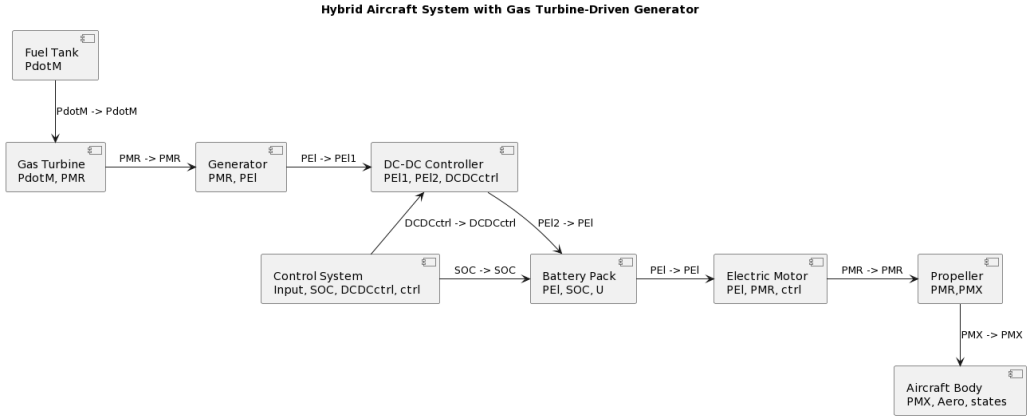

Figure 5 – A hybrid propulsion system with gas-turbine driven generator.

2.2 From Configuration Rules to Python Code: A Process Outline

Another way to further increase the reliability of the configuration is to have Python code generated that can generate the different possible systems from some input variables. This has been shown in [7].Interestingly ChatGPT-4 is quite good at generalizing from a few examples and once the code has been tested and validated, it can be used as a reliable way to generate new concepts.
The following steps outline the process that culminates in the generation of configuration program in Python that implements the design rules for a specific type of systems.

1. **Define Configuration Rules:**
   The initial step involves defining a set of configuration rules that guide the architecture and functionalities within the system. For instance, in an aircraft propulsion system, these rules might detail the types and functions of various components like batteries, fuel cells, motors etc, as well as their interactions. Also some small examples in the form of micro-templates are included to show how the result should be presented (here code for UML (Unified Modeling Language) component diagrams in PlantUML). For the aircraft actuation system it would include example of components, and rules how they should be interconnected.

2. **Validate Rules with UML Diagram:**
   After the configuration rules have been set, they are validated using a UML diagram generated by ChatGPT, This serves as a graphical representation of the system and aids in manual validation of the prompt for the configuration rules. The prompt may have to be reiterated until the result is satisfactory.

3. **Use ChatGPT to generate Python configuration program:**
   By instructing ChatGPT with a prompt to understand the design rules and their logical implementation, it can then produce Python configuration code that adheres to these rules and generates UML-code, with a specified sets of degrees of freedom.

4. **Testing and Validation:**
   The Python code generated undergoes testing to ensure its adherence to the configuration rules. This validation includes using different inputs to generate different system configuration to be represented as UML diagrams. These may lead to reiteration of the earlier steps.

5. **Finalization:**
   Upon successful testing and validation, the Python configuration code is finalized. It now serves as an automated tool for implementing the design rules, thus streamlining what would typically be a manual design process.

The process is indicated in the Sequence diagram in Fig. 6.

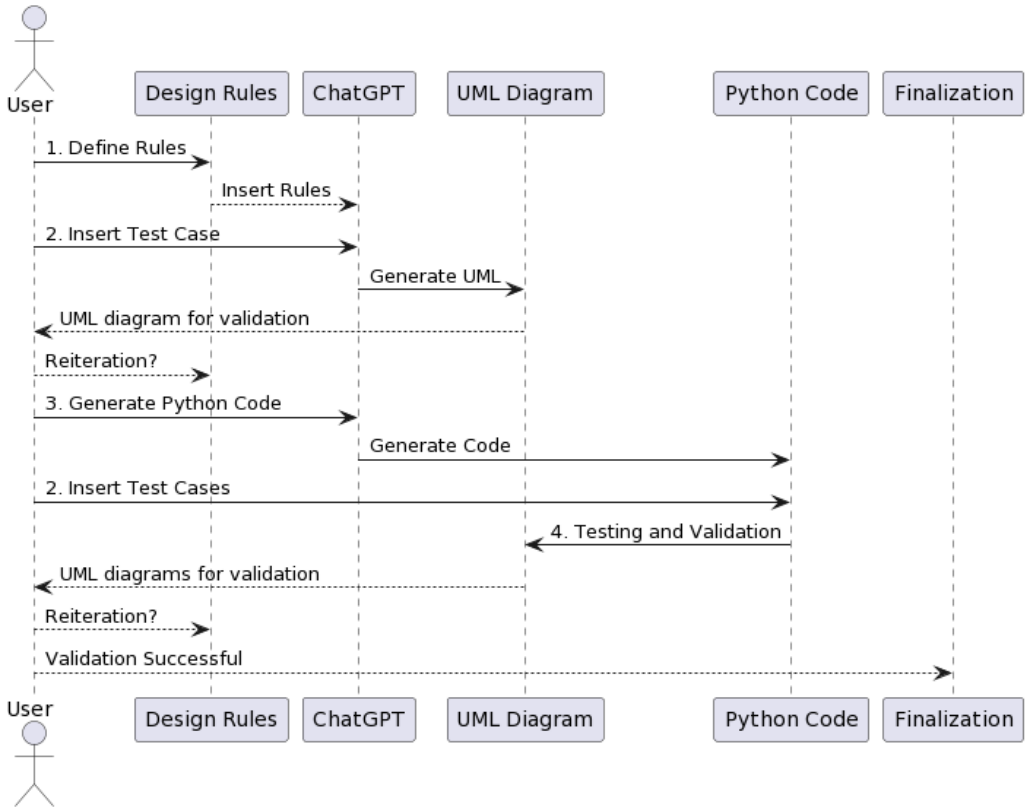

Figure 6 – UML Sequence Diagram illustrating the interaction process from configuration rules to Python configuration code.

The result of this process when applied to the aircraft propulsion system is a Python configuration program that can generate concepts from the resulting design space. The usage is exemplified below

```
    # Example: Create a gas turbine aircraft system with FAR25 certification
system = create_system("Gas Turbine Aircraft System", primary_fuels=["gas_turbine"
print(system.generate_plantuml())
```

3. Example: Aircraft Hydraulic Actuation System

Another Example is the configuration of aircraft actuation system for e.g. the primary flight control system. This involves actuation of ailerons, elevators and rudder. in order to ensure redundancy there are two parallel circuits connected to the functions, so that, e.g., an elevator has two actuator connected to it in case of failure in one of them.

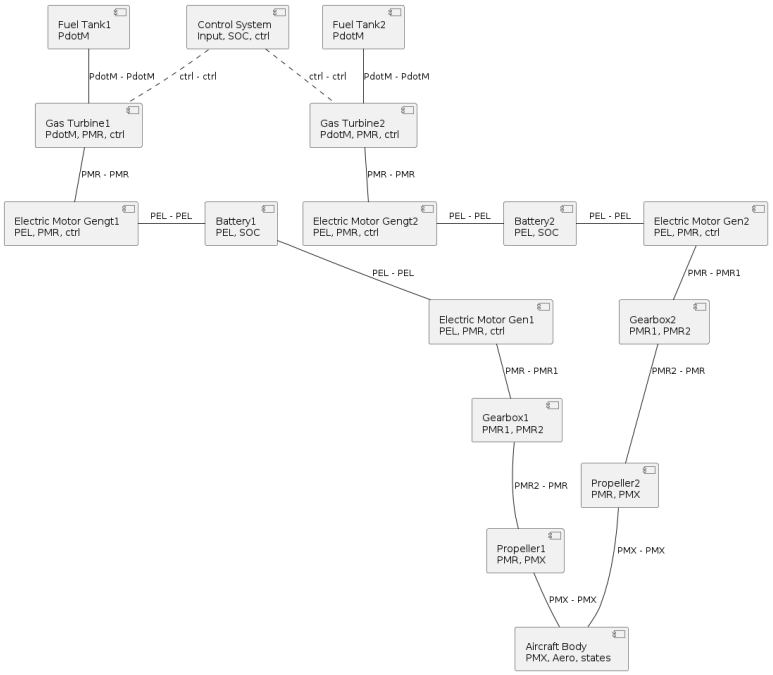

Figure 7 – Series hybrid generated from the example, by the generated Python configuration code

When a satisfactory program has been generated it can be used to create .i.e. a lager system than in the training example. Below is an example of the usage of the code. Here also inboard ailerons have been added.

```
# Example usage:
functions = ["Aileron Left", "Aileron Right","Inboard Aileron Left",
             "Inboard Aileron Right", "Elevator Left", "Elevator Right",
             "Rudder"]
circuits = ["Circuit 1", "Circuit 2"]

plantuml_code = generate_plantuml(functions, circuits)
```

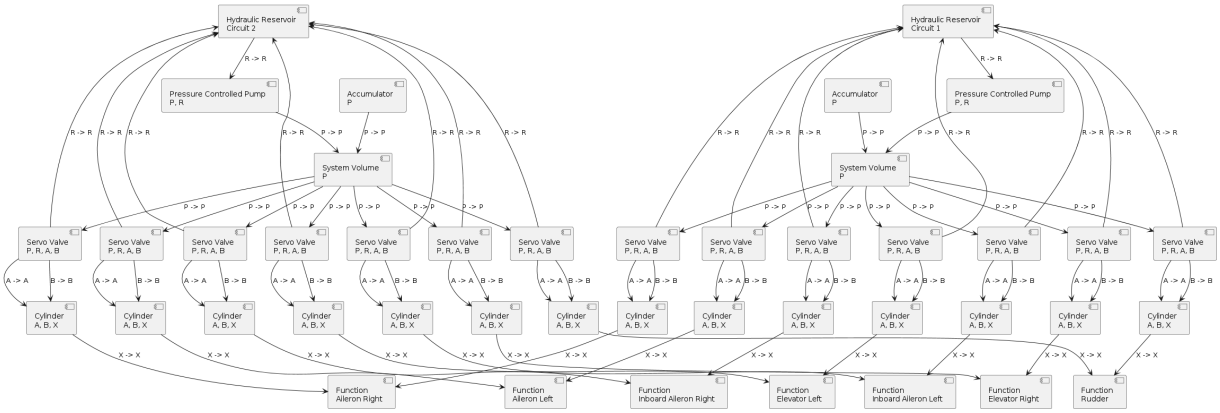

Figure 8 – Actuation system generated from generated configuration program in Python

3.1 Expanding a Configuration Code

We can then continue to ad additional functionality to the Python program step by step. E.g. we can ad the possibility to have additional functions by prompting

Prompt: "Also ad the possibility to have additional actuators (like flaps and landing gear) that are using a subset of the circuits".

```
# Example usage:
functions = ["Aileron Left", "Aileron Right", "Elevator Left", "Elevator Right", "
additional_actuators = {
    "Flaps Left": [1],
    "Flaps Right": [1],
    "Landing Gear Nose": [2],
    "Landing Gear Left": [2],
    "Landing Gear Right": [2]
}
circuits = ["Circuit 1", "Circuit 2"]
```

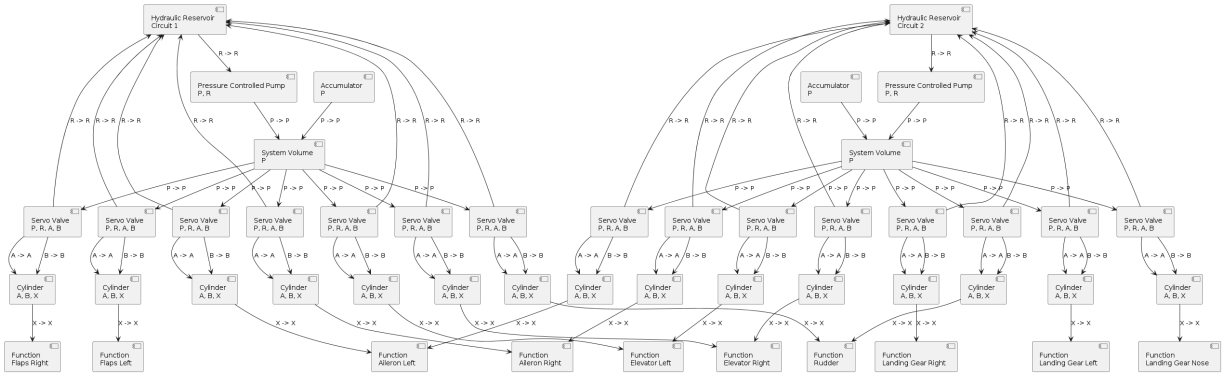

Figure 9 – Actuation system with extra functions (flaps and landing gear) added

4. Simulation

The generated UML code can be used as input to define a simulation model. This is just a matter of mapping the components to the corresponding component in a simulation library of the simulation software, and to map the connections between the components. In this example, a notation has been forced on ChatGPT so that components have ports. Ports can have more than one variable going through them and they can be bi-directional. This is supported in modern simulation software such as Dymola, Simscape, OpenModelica and Hopsan. Of course more information has to be added such as parameter values for the component and setting up and defining the simulation case. No doubt, these tools are likely to adopt AI as part of the user interface, but there is also another possibility. Since ChatGP-4 can generate and execute Python code directly, it is also possible to generate simple simulation models directly inside ChatGPT. Here we could try to just simulate the propulsion system in cruise condition, and with just battery electric propulsion.
To generate this the following prompt can be used.

User: Make UML code for a simulation model of this system. It should also include a simulation manager that controls the simulation and connects to all the component models. Also ad a speed control system that regulates the motor to let the aircraft follow a reference speed with a feedback from the aircraft. (We also need to add a little more detail prompt for the aerodynamics model).

The resulting diagram is shown in Fig. 10
Simulation Manager is the central component responsible for controlling and managing the simulation. It has connections to all other components to manage their states and control signals. Aircraft Body, Battery, Electric Motor, and Propeller are represented as individual components, each with their specific ports and attributes. The Simulation Manager has control and management links to all the components, indicating its role in the simulation process.

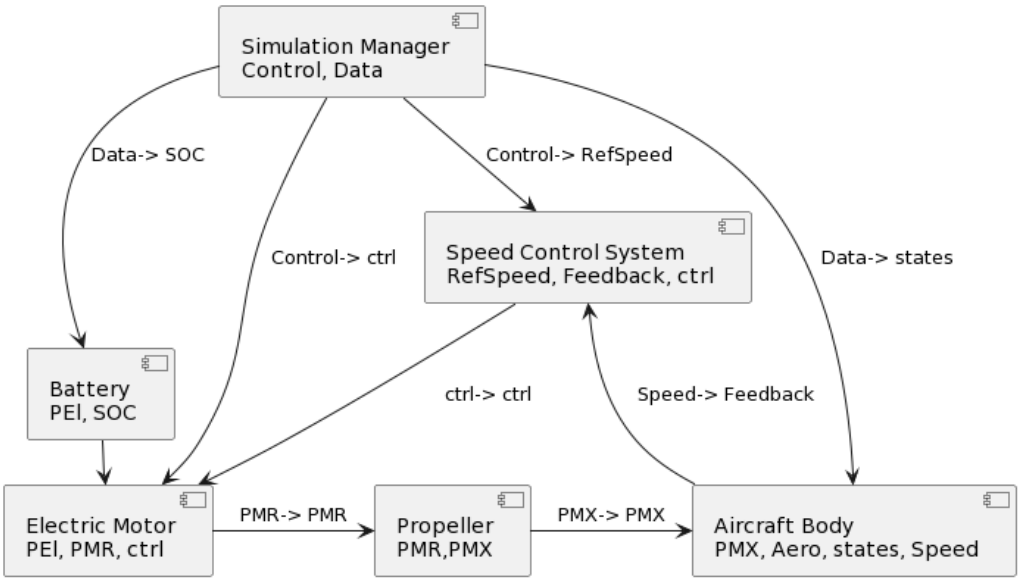

Figure 10 – Simulation model for a battery electric aircraft.

This UML diagram effectively illustrates the structure of a simulation model for a battery electric aircraft, highlighting the interaction between the simulation manager and the component models.
Finally we need to set up the simulation in Python and to run it. This is done by instructing ChatGPT what equations to use. We can actually do much of this in a dialogue with ChatGPT. However, we can also load a pre-defined prompt that is shown in Appendix II.
With this we can ask ChatGPT-4 to set up a simulation model. Admittingly this takes a few correcting prompt to get everything satisfactory, not least regarding parameter values where we can start with asking ChatGPT for reasonable values to start with. Anyway after some iterations we can arrive to the simulation results in Fig. 11. Obviously we can also ask ChatGPT also to do optimization. This shows that conceptual design to a great deal can be carried out inside ChatGPT.

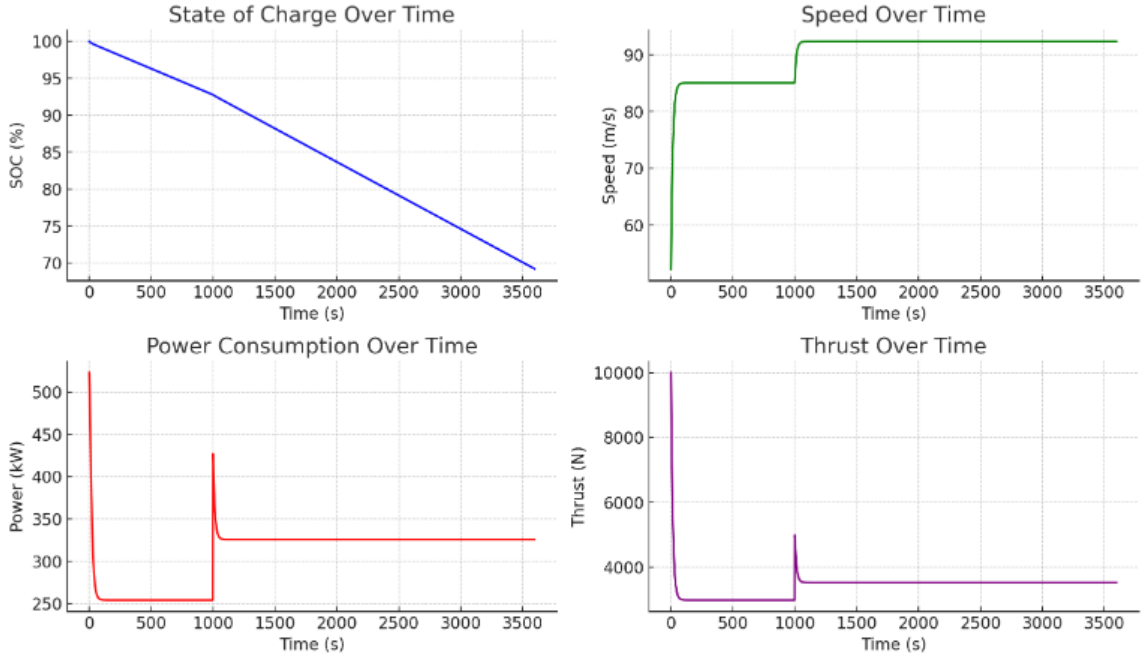

Figure 11

5. Discussion

The results indicate that large language models like ChatGPT have the potential to significantly contribute the process of conceptual design. However, there are limitations, e.g. the capability to input very complex rules. This can be mitigated by working iteratively and gradually increase the complexity. In particular, generating Python code for the design rules and then use ChatGPT to build on the code generated in the previous step for another step, where additional rules are added seems a very useful strategy. Here the output generated was code for UML component diagrams in PlantUML. This is a very simple format that was selected for it minimalist format and ease of use. however, other format can of course als obe used.

The fact that a system architecture can be defined in this way meant that it can be manipulated and and be handed over e.g., to perform system simulation. There is a fairly straightforward process of mapping components in the UML diagram to components in the library of a simulation package. However, very intriguing is also the possibility to make simpler simulations inside ChatGPT-4o that can already be done for simple system, and this capability is only likely to improve.

5.1 Conclusions

In this paper the use of GPT-4o for generating configuration rules coded in software is demonstrated. The examples are hybrid aircraft propulsion system and an aircraft actuation systems. The method involves defining design rules, validating them with UML diagrams, and coding them in a configuration program in Python. It was also shown how a simulation simulation model (of an electric aircraft) could be used to simulated inside ChatGPT using the built in Python environment. It shows that Large language models can be expected to have a profound impact on system design, and at this point we are only in the beginning.

6. Copyright Statement

References

[1] K. Abu Salem, V. Cipolla, G. Palaia, V. Binante, and D. Zanetti. Conceptual Study of Hybrid-Electric Box-Wing Aircraft Towards the Reduction of Aviation Effects on Local Air Quality and Climate Change. In *33rd Congress of the International Council of the Aeronautical Sciences, ICAS 2022*, volume 2, pages 787–811, 2022.

[2] A. Batra, R. Raute, and R. Camilleri. Series or Parallel Hybrid-Electric Aircraft Propulsion Systems? Case Studies of the Atr42 and Atr72. In *33rd Congress of the International Council of the Aeronautical Sciences, ICAS 2022*, volume 2, pages 812–827, 2022.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020-Decem, 2020.

[4] K. Ma, D. Grandi, C. Mccomb, and K. Goucher-Lambert. Conceptual design generation using large language models. *arXiv*, abs/2306.0:1–12, 2023.

[5] V. Marciello, M. Ruocco, F. Nicolosi, and M. Di Stasio. Market Analysis, Tlars Selection and Preliminary Design Investigations for a Regional Hybrid-Electric Aircraft. In *33rd Congress of the International Council of the Aeronautical Sciences, ICAS 2022*, volume 1, pages 587–606, 2022.

[6] F. Nicolosi, V. Marciello, V. Cusati, and F. Orefice. Technology Roadmap and Conceptual Design of Hybrid and Electric Configurations in the Commuter Class. In *33rd Congress of the International Council of the Aeronautical Sciences, ICAS 2022*, volume 2, pages 1298–1313, 2022.

[7] M. Pradas, Alejandro, Krus, Petter, Panarotto and O. Isaksson. Large Language Models in Complex System Design. In *Design 2024*, 2024.

Appendix

This is the user pre-defined prompt used as background information for the generation of UML-component code for aircraft propulsion system.

```
# Aircraft Propulsion System Configuration

This section details the standard configuration for connecting components
in an aircraft propulsion system, focusing on hybrid systems that
incorporate gas turbines, fuel cells, batteries, and electric machines.

## Component Descriptions

- **Gas Turbine**: Converts chemical power (\( \dot{P}_M \)) from a fuel
  source (Fuel Tank or Hydrogen Storage) into rotational power (MR). The
  MR port can only be connected to one component.

- **Fuel Cell**: Converts chemical power (\( \dot{P}_M \)) from Hydrogen
  Storage into electric power (EL).

- **Propeller**: Converts rotational power into linear power, i.e., thrust.

- **Gearbox**: Connects one or more rotational power sources (MR) to a
  single output rotational power (MR).

- **Electric Motor/Generator**: Converts electric power (EL) from a Battery
  into mechanical rotational power (MR) or vice versa. The MR port can only
  be connected to one component.

- **Battery Pack**: A storage unit that provides electric power (EL). It
  can have multiple connections.

## Component Connection Principles

- **Port Matching**: Ensure that ports on each component are connected
  to ports of a similar type on other components. For example, electric
  power ports (EL) should connect to other EL ports, while mass flow
  ports (\( \dot{P}_M \)) should connect to other \( \dot{P}_M \) ports.
  Only include components that are used.

- **Dashed Connections**: When using dashed lines ("-[dashed]-"), do not
  specify direction (e.g., up, down, right, or left).

## Energy Storage Requirements

- At least one form of energy storage is mandatory in the system. This
  could be Hydrogen Storage, a Fuel Tank, or a Battery Pack.
- If there is a gas turbine, it must be connected to a chemical energy
  source from a Fuel Tank or Hydrogen Storage.
- Ensure there is a maximum of one energy source, in addition to the battery.

## Use Requirements

- Ensure there is at least one path from an energy storage component to
  the propeller through power ports, to the Aircraft Body.
```

```
- Ensure sufficient redundancy in the case of more than one propeller.

## Power and Control Separation

- Clearly differentiate between power distribution components (like
  Electric Motor Generators) and energy storage (Battery).
- The Controller regulates components (using ctrl) such as Electric Motor
  Generator, Gas Turbine, and Fuel Cell, and monitors e.g., SOC in Battery.
  There are also external input Refs.

## Micro Templates

Here are some micro templates that show the style of the UML diagram.

### Propeller Connected to an Aircraft

\'''plantuml
@startuml
' Define the components with ports
component "Aircraft Body\nPMX, Aero, states" as AircraftBody
component "Propeller\nPMR,PMX" as Propeller
' Connections
Propeller -down- AircraftBody: PMX- PMX
' Diagram Title
title Aircraft System with Aircraft Body and Propeller
@enduml
\'''

### Connection Between a Battery and a Motor

\'''plantuml
@startuml
' Define the components with ports
component "Battery\nPEL, SOC" as Battery
component "Electric Motor Gen\nPEL, PMR, ctrl" as ElMotorGen
component "Propeller\nPMR,PMX" as Propeller
component "Gearbox\nPMR1,PMR2" as Gearbox
' Connections
Battery -right- ElMotorGen: PEL- PEL1
ElMotorGen -down- Gearbox: PMR- PMR1
Gearbox -right- Propeller: PMR- PMR
' Diagram Title
title Aircraft System with Electric Motor and Propeller
@enduml
\'''

### Turbo Prop Connection

\'''plantuml
@startuml
' Define the components with ports
component "Propeller\nPMR,F" as Propeller
component "Gearbox\nPMR1,PMR2" as Gearbox
```

```
component "Fuel Tank\nPdotM" as FuelTank
component "Gas Turbine\nPdotM, PMR, ctrl" as GasTurbine
' Connections
FuelTank -right- GasTurbine: PdotM - PdotM
GasTurbine -down- Gearbox: PMR- PMR1
Gearbox -down- Propeller: PMR2- PMR
' Diagram Title
title Aircraft System with Gas-Turbine and Propeller
@enduml
\```

### Fuel Cell Example

\```plantuml
@startuml
' Define the components with ports
component "Hydrogen Storage\nPDotM" as HydrogenStorage
component "Battery\nPEL, SOC" as Battery
component "Fuel Cell System\nPdotM, PEL,ctrl" as FuelCell
component "Control System\nInput, SOC,ctrl" as ControlSystem
' Define connections
HydrogenStorage -right- FuelCell: PdotM-> PdotM
FuelCell -right- Battery: PEL- PEL1
Battery -[dashed]-> ControlSystem: SOC- SOC
ControlSystem -[dashed]-> FuelCell: ctrl- ctrl
' Add a title for the diagram
title Fuel Cell Connected to a Battery
@enduml
\```

### Gas-Turbine Generator Example

\```plantuml
@startuml
' Define the components with ports
component "Fuel Tank\nPdotM" as FuelTank
component "Gas Turbine\nPdotM, PMR, ctrl" as GasTurbine
component "Generator System\nPMR, PEL,ctrl" as Generator
component "Battery\nPEL, SOC" as Battery
component "Control System\nInput, SOC,ctrl" as ControlSystem
' Define connections
FuelTank -right- GasTurbine: PdotM- PdotM
GasTurbine -right- Generator: PMR- PMR
Generator -right- Battery: PEL2- PEL1
Battery -[dashed]-> ControlSystem: SOC- SOC
ControlSystem -[dashed]-> GasTurbine: ctrl- ctrl
ControlSystem -[dashed]-> Generator: ctrl- ctrl
' Add a title for the diagram
title Gas-Turbine Series Hybrid
@enduml
\```

## System Outline and Validation
```

```
1. **Select maximum one non-electric energy storage.**
2. **Ensure inclusion and connection of the Aircraft Body.**
3. **Include one battery for each electric motor generator.**
4. **Ensure gas turbines have a chemical energy source from a Fuel Tank
  or Hydrogen Storage.**

## Final Solution

1. Outline the system and check it against the instructions.
2. Validate the architecture against the instructions.
3. Write the final solution as code for a component UML diagram in PlantUML.
```