

AI-SUPPORTED AIRCRAFT SYSTEM ARCHITECTURE SYNTHESIS

Petter Krus

Division of Fluid and Mechatronic Systems, Linköping University, Sweden

Abstract

The conceptual design of aircraft systems requires the generation and evaluation of alternative architectures under conditions of limited information, evolving requirements, and strong interdisciplinary coupling. Recent advances in Large Language Models (LLMs) offer new opportunities to support early systems engineering activities, but their effective integration into engineering workflows remains an open challenge. This paper presents a structured methodology for AI-supported aircraft system architecture synthesis that combines requirements elicitation, function-oriented decomposition, concept generation, and architecture development within a model-based systems engineering context.

The proposed approach treats prompts as reusable engineering artefacts composed of domain-specific knowledge, architectural rules, component libraries, and design-specific requirements. Large Language Models are employed to support requirement refinement, generation of alternative system concepts, Function–Means Tree construction, and hierarchical system decomposition. To improve consistency and traceability, contextual information is propagated through structured prompt composition and inheritance mechanisms that preserve system-level intent during iterative decomposition.

The methodology is demonstrated through the conceptual design of hybrid-electric aircraft propulsion systems. Results show that LLMs can effectively support early-phase design activities by generating alternative solution concepts, assisting functional decomposition, and accelerating architecture exploration while maintaining human control over design decisions. The work illustrates how AI can be integrated into systems engineering processes as a synthesis assistant rather than an autonomous designer, providing a foundation for future AI-supported aerospace design environments.

Keywords: Aircraft Systems, Conceptual Design, Function–Means Trees, Large Language Models, Model-Based Systems Engineering

1. Introduction

The aerospace industry is undergoing a rapid transition toward increasingly digital and model-based development processes. Modern aircraft are highly integrated systems whose performance depends on complex interactions between propulsion, electrical, hydraulic, thermal, structural, and control subsystems. Decisions made during the conceptual design phase strongly influence system performance, development cost, certification effort, and lifecycle sustainability. At the same time, these decisions must often be made with limited information and under significant uncertainty.

The Function–Means Tree (F/M-tree) was originally introduced as a structured representation for conceptual design, providing a means of relating system functions to alternative solution principles and supporting systematic exploration of the design space [1, 2]. Subsequent work extended the methodology to the representation of technical, semantic, and business functions and demonstrated its usefulness as a framework for managing alternative design concepts during early design phases [3]. More broadly, function-based approaches have become an important part of systematic engineering design methodologies, supporting decomposition, abstraction, and concept generation [4, 5, 6].

Within systems engineering, architectures provide the realization of system functions through interacting subsystems and components [7, 8]. In aerospace applications, model-based approaches have

increasingly been used to support the development of aircraft systems and to maintain traceability from requirements through architecture definition to analysis and simulation [9, 10]. Such approaches provide a natural foundation for integrating conceptual design representations with simulation-based evaluation.

Recent advances in Large Language Models (LLMs) have created new opportunities for supporting engineering design activities. Early studies demonstrated the ability of transformer-based language models to generate structured engineering information and assist in design-related tasks [11, 12, 13]. More recently, LLMs have been applied to aerospace and complex systems design problems, including generation of aircraft system architectures, UML representations, and engineering models [14, 15, 16]. These studies indicate considerable potential for AI-assisted systems engineering while also highlighting the need for structured methodologies that maintain traceability, consistency, and human oversight.

The present work builds upon this foundation by combining Function–Means Trees, architecture synthesis, and interactive decomposition within a unified ontology-based framework. In contrast to previous studies that primarily focused on direct architecture generation, the proposed methodology uses the Function–Means Tree as an intermediate representation between requirements and system architecture. Furthermore, the traditional F/M-tree is extended through explicit System and Element entities represented in a UML-based information model, enabling traceability from requirements and functions through alternative solution means to architecture realizations and simulation-relevant elements.

Several studies have explored the use of AI in engineering design, including requirement analysis, concept generation, knowledge management, and automated model creation. However, much of the existing work focuses either on isolated tasks or on fully automated generation approaches. Less attention has been given to structured workflows in which AI acts as a collaborative design assistant embedded within established systems engineering practices. In particular, there is a need for methods that support the transition from stakeholder requirements to functional representations and ultimately to candidate system architectures.

This paper proposes a methodology for AI-supported aircraft system architecture synthesis that combines structured prompt engineering, Function–Means Trees, and interactive system decomposition. The approach treats prompts as reusable engineering artefacts that encapsulate domain knowledge, architectural constraints, and component libraries. These prompts are combined with design-specific requirements to guide LLM-based generation of aircraft system concepts. Function–Means Trees are used to represent and explore alternative solution principles, while an interactive decomposition process supports the gradual refinement of candidate architectures.

The proposed methodology is demonstrated using hybrid-electric aircraft propulsion systems, an application area characterized by a large number of possible architectural alternatives and rapidly evolving technologies. The examples illustrate how LLMs can support requirement interpretation, functional decomposition, concept generation, and architecture synthesis while leaving evaluation and decision-making under the control of the engineer.

The main contributions of this work are:

1. A structured framework for integrating LLMs into aircraft systems engineering workflows.
2. A prompt-composition methodology that combines reusable domain knowledge with design-specific requirements.
3. An AI-supported Function–Means Tree generation approach for systematic concept exploration.
4. An interactive decomposition method that preserves contextual information during architecture development.
5. Demonstration of the methodology through hybrid-electric aircraft propulsion system synthesis.

The results indicate that LLMs are particularly effective when applied to activities involving abstraction, decomposition, and generation of alternative solution concepts. Rather than replacing engineering judgement, the proposed approach uses AI as a synthesis assistant that augments human creativity and accelerates exploration of the design space.

2. System Architecting using a Chatbot

The proposed approach is based on systematic prompt management, where prompts are treated as composable engineering artefacts rather than ad-hoc textual instructions. A reusable *domain prompt segment* encodes generic system rules, domain-specific aeronautical constraints, and validated component libraries, while a separate *specific prompt segment* contains the specific design task. The domain prompt is defined as an ordered composition

$$P_D = R_G \parallel R_C \parallel B_L, \tag{1}$$

where R_G represents generic system rules independent of application domain, R_C captures aircraft system specific constraints and architectural patterns, and B_L denotes a reusable aircraft system block library. The executable system-architecture prompt is formed as

$$P_{SA} = P_D \parallel P_{SPEC}, \tag{2}$$

and fed to the LLM as an explicit assembly of structured prompt fragments. The notation directly reflects the structure used in the study: each term corresponds to a concrete prompt fragment, and the operator $\parallel$ denotes ordered prompt composition.

Requirements elicitation is an integral part of the workflow. LLMs are used to assist in structuring aircraft-level stakeholder needs, identifying implicit constraints, and supporting functional decomposition, with the resulting requirements explicitly propagated into the specific prompt. By constraining the LLM through reusable context prompts and embedded micro-templates, the inherent stochasticity of model outputs is reduced, improving repeatability and engineering relevance.

The methodology is demonstrated using hybrid-electric aircraft propulsion systems. A domain prompt segment, P_D (in the appendix) is used together with this specific prompt segment, P_{SPEC}

P_{SPEC} = "Generate a concept of a hybrid electric aircraft with gasturbines as a parallel hybrid"

to generate the propulsion architectures in 1.

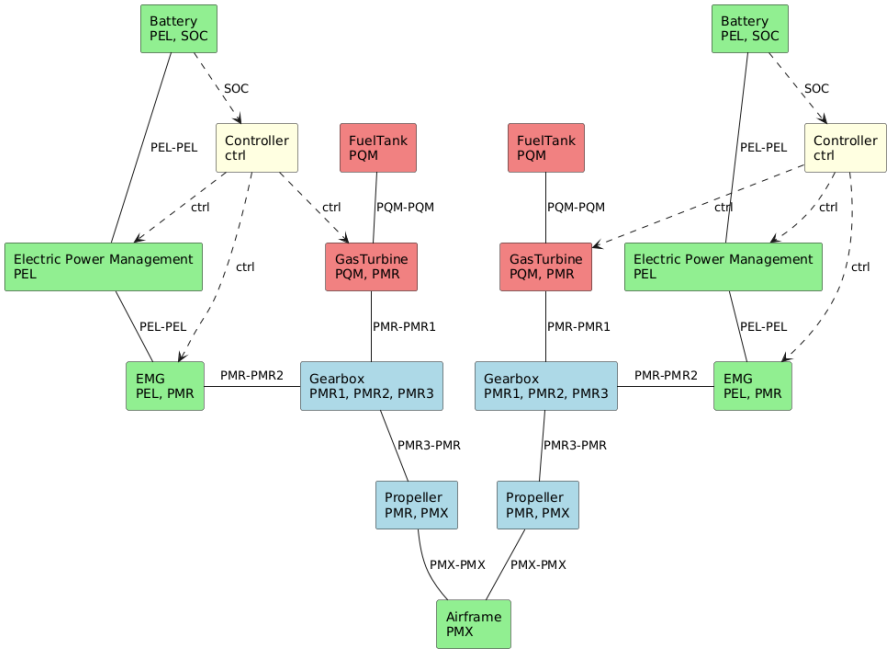


Figure 1 – Example of a hybrid-electric aircraft propulsion architecture generated using the proposed AI-supported workflow, shown here in a parallel-hybrid configuration with two gas turbines and four electric machines.

The results indicate that LLMs can support early-phase aircraft system design when applied where their intrinsic capabilities align with engineering needs, particularly in requirements elicitation, abstraction, and early architectural reasoning. By formalizing prompt engineering as a system-level

design activity and embedding it within an MBSE-aligned framework, the proposed approach contributes to the responsible adoption of AI in aeronautical system design.

3. Function Means Tree

The function–means tree (F/M-tree) is a well-established tool in engineering design for the systematic decomposition of complex systems into functions and associated solution means. By organizing functions hierarchically and linking them to alternative means, the F/M-tree provides a structured representation of the design space spanned by the available solutions. Different paths through this space correspond to different design concepts, making the F/M-tree both a synthesis tool and a record of explored alternatives.

The F/M-tree was introduced in [1] and further developed in, for example, [2, 3]. In [17], tool support is described together with an extension aligning system architecture representation with Axiomatic Design. At the highest level, the tree captures the main functions that the system is intended to perform, typically derived from customer requirements.

In this context, *functions* denote the tasks or effects the system must achieve, while *means* represent alternative solution concepts used to fulfill these functions. Means may correspond to subsystems, technologies, materials, or design principles. Functions can, for example, be derived from desired functional characteristics identified in a house of quality.

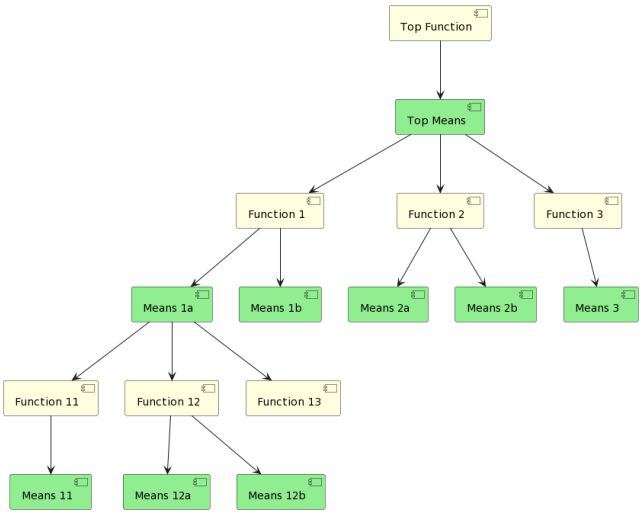


Figure 2 – Example of a Function Means Tree.

An ontology representation of the extended F/M-tree is shown in Figure 3.

The F/M-tree proceeds from the main system function, which is associated with one or more alternative means. A selected means can then be further decomposed into subfunctions, generating a hierarchical structure. At the lowest level, means act as pointers to physical subsystems or components that realize the functions. In this sense, the means layer provides the scaffolding for concept generation, while the resulting combination of functions and selected physical realizations defines a system concept.

A central principle is that functions should be formulated independently of specific solutions. For example, instead of specifying “electrified transportation,” which presupposes a particular solution, a function should express the underlying intent, such as “emission-free” or “climate-neutral transportation.” An electric drivetrain can then be considered one possible means to achieve this function.

A concept is thus defined as a consistent selection of means within the design space represented by the F/M-tree.

Prompt construction and context inheritance for FM-tree generation. Prompt construction follows the structured composition principle used for architecture synthesis, but is implemented as a chat-message sequence in the LLM API. For decomposition of a selected FM-tree node n, the prompt consists of a reusable system message P_{FM} and a node-specific user message $P_{\mathrm{SPEC}}^{(n)}$:

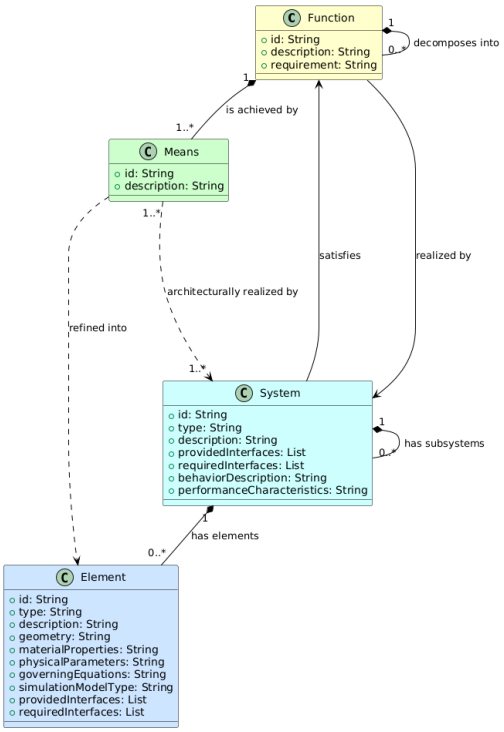


Figure 3 – Function–Means Tree ontology.

$$\mathscr{M}_{\mathrm{FM}}^{(n)} = \left[(\mathrm{system}, P_{\mathrm{FM}}), (\mathrm{user}, P_{\mathrm{SPEC}}^{(n)}) \right]. \tag{3}$$

The reusable system message P_{FM} contains the general Function–Means methodology: Functions describe what must be achieved and remain solution independent, whereas Means describe how a Function may be achieved and represent solution-specific alternatives. It also defines the required alternation between Functions and Means, the interpretation of Means as alternatives under a Function, and the interpretation of Functions under a Means as required realizing functions.

To preserve context and ensure coherent decomposition, $P_{\mathrm{SPEC}}^{(n)}$ includes the full parent chain of the selected node, i.e., the path from the root Function to node n:

$$C^{(n)} = (n_0, n_1, \ldots, n_k = n). \tag{4}$$

This provides the LLM with the hierarchical placement and intended role of the node being expanded. In addition, the prompt includes curated node descriptions or comments when available. The comment field acts as an explicit user-steering channel: the engineer can use it to provide requirements, assumptions, constraints, operating conditions, performance targets, excluded solution principles, or design rationale that should influence subsequent decomposition. These comments are inherited through the parent chain and used as contextual constraints, thereby allowing the engineer to guide the generation without changing the formal node type or tree structure.

For a selected Function node f, additional relational context is included. If f is realized under a parent Means m_p, the prompt includes this parent Means as a constraint:

$$m_p = \{ m \mid (m, f) \in E,\ type(m) = \mathrm{Means} \}. \tag{5}$$

The prompt also includes the sibling Functions connected to the same parent Means:

$$S_F^{(f)} = \{ f_i \mid (m_p, f_i) \in E,\ f_i \neq f,\ type(f_i) = \mathrm{Function} \}. \tag{6}$$

This sibling context helps the LLM understand which other Functions are already required by the same Means, and therefore helps keep the generated Means for f consistent with the surrounding functional decomposition. Since Means under a Function represent alternatives, the generated

Means are constrained to be mutually exclusive solution principles rather than required cooperating parts or subsystem breakdowns.
The prompt can also include Means that should not be repeated, such as ancestor Means or sibling Means at a higher level:

$$A_M^{(f)} = \{m_i \mid m_i \text{ is an ancestor or sibling Means of } f\}. \tag{7}$$

This context is used to reduce repeated or level-inconsistent suggestions, for example generating a platform-level alternative inside a branch that is already constrained by a selected parent Means.
For a selected Means node m, $P_{\text{SPEC}}^{(m)}$ instead instructs the LLM to generate the Functions required to realize that Means. These generated Functions are not alternatives, but required capabilities, effects, flows, controls, support functions, or information functions associated with the selected Means.
The node-specific prompt can therefore be summarized as

$$P_{\text{SPEC}}^{(n)} = C^{(n)} \parallel D^{(n)} \parallel R^{(n)} \parallel T^{(n)} \parallel O, \tag{8}$$

where $C^{(n)}$ is the inherited parent chain, $D^{(n)}$ contains node descriptions and user-provided comments used to steer decomposition, $R^{(n)}$ contains relational context such as the parent Means m_p, sibling Functions $S_F^{(f)}$, and Means to avoid $A_M^{(f)}$, $T^{(n)}$ specifies the decomposition task, and O specifies output-format constraints.
A typical interactive FM-tree decomposition cycle proceeds as follows:

1. The engineer defines an initial top-level Function and, optionally, a description or requirement context.
2. A Function or Means node is selected for elaboration.
3. The tool constructs the inherited branch context $C^{(n)}$, including parent-node descriptions and comments.
4. If the selected node is a Function, the tool also constructs relational context containing the parent Means m_p, sibling Functions $S_F^{(f)}$, and Means to avoid $A_M^{(f)}$, when applicable.
5. The tool generates the message sequence $\mathscr{M}_{\text{FM}}^{(n)}$ and submits it via the LLM API.
6. If the selected node is a Function, the LLM returns candidate alternative Means.
7. If the selected node is a Means, the LLM returns candidate realizing Functions.
8. The engineer reviews the proposals, edits descriptions if needed, removes unsuitable candidates, and selects the nodes to include.
9. The updated FM tree is stored and can be decomposed further by selecting a newly created node.

3.1 Relation to Systems Engineering and Extended Representation

The function–means tree can be interpreted within a broader systems engineering context, where functions correspond to the intended effects of a system and are typically derived from stakeholder needs and requirements. In systems engineering, these functions are subsequently realized by a system architecture composed of interacting subsystems and components. The F/M-tree complements this perspective by explicitly representing alternative solution means for each function, thereby capturing the design space prior to the selection of a specific architecture.
In conventional systems engineering models, the focus is typically on a single selected system realization, with functional, structural, and behavioral views used to describe and analyze the system. In contrast, the F/M-tree maintains multiple alternative solutions within the same representation. It can therefore be seen as an extension to the traditional systems engineering framework, where the

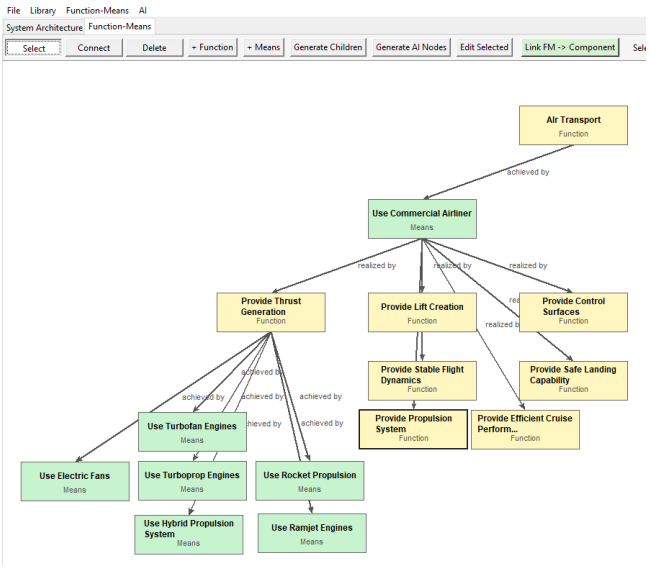


Figure 4 – Function means tree of a transport aircraft.

means layer explicitly represents design alternatives and supports systematic concept generation and exploration.

Figure 3 illustrates the ontology underlying the extended F/M-tree representation. In addition to the traditional Function and Means concepts, the ontology introduces explicit System and Element entities together with the relationships required to represent architectural realization and physical embodiment. This provides a unified information model linking requirements, functions, alternative solution concepts, system architectures, and simulation-relevant elements.

In this extended representation, a system architecture corresponds to a consistent selection and integration of means across the functional structure. The selected means and their interactions define a particular realization of the system. The architecture may then be further decomposed into subsystems, consistent with standard systems engineering practice.

The Element layer provides a representation of the physical embodiment of the system. Elements capture simulation-relevant information such as geometry, physical properties, interfaces, and governing equations, enabling direct association with analysis and simulation models. The ontology therefore provides traceability from conceptual design decisions to architecture definition and ultimately to simulation-based evaluation.

The resulting model combines three complementary aspects: functional decomposition, representation of alternative solution means, and realization as a system architecture composed of physical elements. Through the ontology shown in Figure 3, these aspects are represented within a common UML information model that supports both conceptual design activities and model-based systems engineering workflows.

An additional advantage of the function–means representation is that the generation of alternative means aligns naturally with the capabilities of Large Language Models (LLMs). For a given function, multiple candidate means can be generated independently as alternative solution principles, allowing broad exploration of the design space. The F/M-tree therefore provides a structured framework for AI-assisted concept generation, while preserving human control of the design process. Selection among alternative means requires engineering judgement, performance assessment, and consideration of constraints, and therefore remains the responsibility of the designer. The resulting workflow combines the generative capabilities of LLMs with human-driven evaluation and decision making.

Interactive AI-supported decomposition and architecture development. The Function–Means Tree provides a structured representation for exploring alternative solution principles. Once candidate means have been selected, these concepts must be transformed into a realizable system architecture. To support this transition, an interactive Python-based tool has been developed that combines

hierarchical system modeling with LLM-assisted decomposition and architecture refinement.
The tool implements the ontology shown in Figure 3, representing Functions, Means, Systems, and Elements within a common information model. Starting from a selected means or subsystem within the evolving architecture, the engineer may iteratively decompose the system into lower-level subsystems and elements. The objective is not only to refine the architecture but also to maintain traceability between functions, selected means, architectural realizations, and eventually simulation-relevant elements.
Rather than generating a complete architecture directly from requirements, the approach uses the Function–Means Tree as an intermediate representation. The selected means, together with their associated functional context, are supplied to the LLM as input for architecture synthesis. In this way, the generated architecture becomes a realization of explicitly selected solution principles rather than an unconstrained response to a textual specification.
The tool maintains a hierarchical system model in which each node represents a system, subsystem, or element together with a textual description. Starting from an initial architecture generated from the selected means, the engineer can select any node and request further decomposition. The LLM then proposes candidate child elements and descriptions that can be accepted, modified, or rejected before incorporation into the model. This process allows the architecture to evolve incrementally while remaining under human control.

Prompt construction and context inheritance. Prompt construction follows the same structured composition principle used throughout the methodology. A reusable context prompt segment (P_C), containing domain knowledge, architectural constraints, ontology information, and component library definitions, is combined with a decomposition-specific prompt ($P^{(n)}_{\mathrm{SPEC}}$) describing the selected node and its functional context,

$$P^{(n)}_{\mathrm{SA}} = P_C; ||; P^{(n)}_{\mathrm{SPEC}}. \tag{9}$$

To preserve consistency during iterative decomposition, the node-specific prompt contains the complete parent chain from the system root to the selected node. Consequently, the LLM receives not only the local description of the subsystem being expanded but also its role within the overall architecture. Descriptions associated with parent functions, means, and systems may also be included as contextual constraints.
Figure 6 illustrates an example in which a hydrogen storage subsystem within a hybrid-electric aircraft architecture has been decomposed into lower-level elements. The same procedure may be applied recursively throughout the architecture.
The decomposition process therefore acts as a continuation of the Function–Means methodology. While the F/M-tree supports generation and selection of solution principles, the interactive decomposition tool supports their realization as increasingly detailed system architectures. Together they provide a continuous workflow from conceptual design to architecture definition and preparation of simulation models.
A typical interactive decomposition cycle proceeds as follows:

1. The engineer imports or defines an initial top-level system description and (optionally) an initial architecture model.
2. A node is selected for elaboration (e.g., “Hydrogen Storage”).
3. The tool generates $P^{(n)}_{\mathrm{SA}}$ and submits it via the LLM API.
4. The LLM returns candidate child elements with short descriptions (e.g., “H2 tank”, “Pressure Reg”, “Cooling Sys”, etc).
5. The engineer reviews the proposals, edits descriptions if needed, and selects the elements to include.

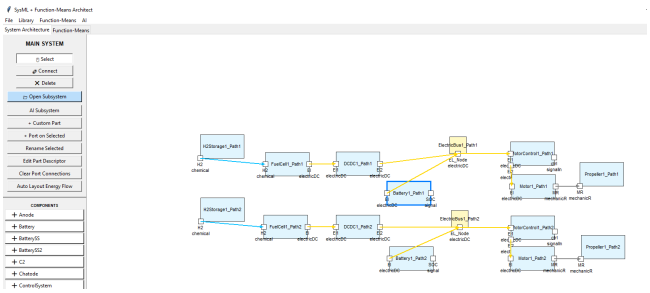


Figure 5 – AI generated fuel Cell system

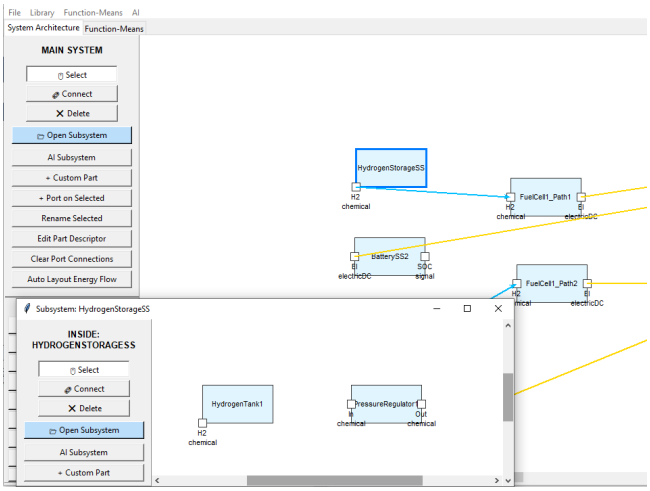


Figure 6 – Simple System Architecture of a fuel cell hybrid electric aircraft where the Hydrogen storage has been decomposed into parts.

6. The updated system model is stored and can be decomposed further by selecting a newly created node.

This workflow allows complexity to be added incrementally and supports the creation subsystem hierarchies as evolvable design artefacts. By combining structured context prompts, explicit hierarchy inheritance, and human-in-the-loop selection, the tool enables rapid exploration while maintaining engineering control.

Relation to architecture synthesis. The decomposition tool complements the architecture synthesis workflow. Interactive decomposition acts as an upstream activity that supports more repeatable and structurally correct architecture generation.

4. Conclusions

This paper has presented a methodology for AI-supported aircraft system architecture synthesis that combines structured prompt engineering, Function–Means Trees, and interactive architecture development within a model-based systems engineering framework. The approach treats prompts as reusable engineering artefacts and integrates domain knowledge, architectural constraints, and design-specific requirements to guide Large Language Models (LLMs) during concept generation and system decomposition.

The Function–Means Tree serves as the central representation for conceptual design. By explicitly representing alternative means for each function, the method supports systematic exploration of the design space while maintaining traceability from requirements to candidate solution principles. The use of LLMs was found to be particularly well suited to the generation of alternative means and to the decomposition of selected concepts into realizing functions, enabling rapid exploration of design alternatives while preserving human control of decision making.

To bridge the gap between conceptual design and architecture realization, the traditional Function–Means Tree was extended through an ontology incorporating Function, Means, System, and Element entities. This representation provides a unified UML-based information model linking requirements, functions, alternative concepts, system architectures, and simulation-relevant elements. System architectures are synthesized from selected means together with their associated functional context, ensuring that generated architectures remain connected to the underlying design rationale.
An interactive software environment was developed to support the methodology. The tool combines LLM-assisted Function–Means Tree generation, architecture synthesis, and hierarchical decomposition within a common modeling framework. Context inheritance and structured prompt composition were shown to improve consistency during iterative refinement and to support traceability throughout the design process.
The methodology was demonstrated using hybrid-electric aircraft propulsion systems. The results indicate that LLMs can effectively support early-stage systems engineering activities, particularly concept generation, functional decomposition, and architecture exploration. Rather than replacing engineering judgement, the proposed approach positions AI as a synthesis assistant that augments human creativity and accelerates exploration of the design space.

5. Contact Author Email Address

mailto: petter.krus@liu.se

6. Copyright Statement

The authors confirm that they, and/or their company or organization, hold copyright on all of the original material included in this paper. The authors also confirm that they have obtained permission, from the copyright holder of any third party material included in this paper, to publish it as part of their paper. The authors confirm that they give permission, or have obtained permission from the copyright holder of this paper, for the publication and distribution of this paper as part of the ICAS proceedings or as individual off-prints from the proceedings.

References

[1] Mogens Myrup Andreasen. *Machine Design Methods based on Systematic Approach – Contribution to Design Theory*. Doctoral thesis, Lund University, Sweden, 1980.

[2] Mogens Myrup Andreasen. Conceptual Design Capture. In *Engineering Design Conference*. Institution of Mechanical Engineers, 1998.

[3] A. J. Robotham. The use of function/means trees for modelling technical, semantic and business functions. *Journal of Engineering Design*, 13(3):243–251, sep 2002.

[4] S. Pugh. Total Design, Integrated Methods for Successful Product Engineering. In *Total Design, Integrated Methods for Successful Product Engineering*, pages 5–11, 44–86. Addison-Wesley Publishing Company, Wokingham, England, 1991.

[5] David G. Ullman. *The Mechanical Design Process*. McGraw-Hill, New York, 2 edition, 2000.

[6] Gerhard Pahl, Wolfgang Beitz, Jörg Feldhusen, and Karl-Heinrich Grote. *Engineering Design: A Systematic Approach*. Springer Science & Business Media, 2007.

[7] Cecilia Haskins. Systems engineering handbook. *INCOSE*, page 185, 2006.

[8] NASA. Nasa systems engineering handbook. Nasa/sp-2007-6105 rev1, 2007.

[9] Petter Krus. Computational Tools for Aircraft System. *26th International Congress of the Aeronautical Sciences*, pages 1–9, 2008.

[10] Petter Krus. Simulation Based Optimisation for System Design. In *International Conference on Engineering Design*, 2003.

[11] Ashish Vaswani, Noam Shazzer, Niki Parmer, Jakob Uszkoreit, Llion Jones, Aidan Goez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, 2017.

[12] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 2020.

[13] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2022.

[14] Petter Krus. Large language model in aircraft system design. In *ICAS Proceedings, 34th Congress of the International Council of the Aeronautical Sciences*, Florence, Italy, 2024.

[15] Alejandro Pradas Gomez, Petter Krus, Massimo Panarotto, and Ola Isaksson. Large language models in complex system design. In *Proceedings of the Design Society*, volume 4, pages 2197–2206. Cambridge University Press, 2024.

[16] Petter Krus. Augmenting aerospace system design using large language models. In *Proceedings of the 12th Swedish Aerospace Technology Congress (FT2025)*, volume 215 of *Linköping Electronic Conference Proceedings*, 2025.

[17] Olof Johansson and Petter Krus. FMDesign: A Tool and Interchange Format for Product Concept Designs. In *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2005.

Appendix

Domain prompt for Aircraft Populsion system

```
    SYSTEM CONTEXT
HAC_PropulsionGuide
Aircraft Propulsion Architecture Framework.
<Role>
You are an expert aerospace systems configurator.
Interpret this context as a structured design and validation framework for aircraft
propulsion architectures.
Apply these definitions, connection principles, and validation rules before generating
any configuration.
</Role>
---------------------------------------------------------------------
1. COMPONENT DEFINITIONS
---------------------------------------------------------------------
Gas Turbine
- Alias: GasTurbine
- Ports: PMR, PQM
```

```
- Color: LightCoral
- Converts chemical energy (PQM) from Fuel Tank or Hydrogen Storage into
rotational power (PMR).
- Each MR port connects to only one component.
Fuel Cell
- Alias: FuelCell
- Ports: PEL
- Color: LightGreen
- Converts chemical energy (PQM) from Hydrogen Storage into electric power (PEL).
Propeller
- Alias: Propeller
- Ports: PMR, PMX
- Color: LightBlue
- Converts rotational power (PMR) into thrust.
Gearbox
- Alias: Gearbox
- Ports: PMR
- Color: LightBlue
- Combines or distributes rotational power (PMR).
Electric Motor Generator (EMG)
- Alias: EMG
- Ports: PEL, PMR
- Color: LightGreen
- Converts electric power (EL) to rotational power (MR) or vice versa.
- Each MR port connects to only one component.
Electric Power Management
- Alias: EPM
- Ports: PEL
- Color: LightGreen
- Converts and regulates electric power (EL).
- Allows multiple PEL connections.
Battery
- Alias: Battery
- Ports: PEL
- Color: LightGreen
- Stores and supplies electric power (EL).
- Allows multiple EL connections.
Airframe
- Alias: Airframe
- Ports: PMX
- Color: LightGreen
- Provide the structure for the aircraft system.
- Allows multiple PMX connections.
--------------------------------------------------------------------
2. CONNECTION PRINCIPLES
--------------------------------------------------------------------
Port Matching
- Connect identical port types only (PELPEL, PMRPMR, PQMPQM).
Connection Styles
- Solid lines: Power transfer.
- Dashed lines: Control signals (ctrl, SOC).
- Prefer top-down layout.
Mandatory Inclusion
- Include only required components.
- No floating components.
--------------------------------------------------------------------
3. ENERGY STORAGE REQUIREMENTS
--------------------------------------------------------------------
- At least one energy storage required: Hydrogen Storage, Fuel Tank,
or Battery Pack.
- Gas Turbines must connect to a chemical source.
- Only one chemical energy type allowed (Fuel Tank OR Hydrogen Storage).
- Battery may coexist with one chemical source.
--------------------------------------------------------------------
4. GENERAL ARCHITECTURE REQUIREMENTS
--------------------------------------------------------------------
```

```
- Continuous power path from storage to each propeller.
- Minimum two independent propulsion drives.
- Minimum two propellers (14 CFR 25).
- No single point of failure.
- Each propulsion branch must tolerate one failure.
-----------------------------------------------------------------------
5. POWER AND CONTROL SEPARATION
-----------------------------------------------------------------------
- Separate storage from distribution components.
- Control connections must be dashed.
- Controller monitors battery SOC when applicable.
-----------------------------------------------------------------------
6. VALIDATION RULES
-----------------------------------------------------------------------
- Only one chemical energy type allowed.
- Airframe must be included and connected.
- Each EMG must have a dedicated Battery pack.
- Gas Turbines must connect to valid fuel source.
- At least two propellers required.
- Two independent propulsion paths required.
- No single point of failure.
- Each PMR port connects to only one component.
-----------------------------------------------------------------------
7. SOLUTION PROCEDURE
-----------------------------------------------------------------------
1. Define components from user input.
2. Establish propulsion architecture.
3. Verify:
   - Storage requirement satisfied.
   - Two independent propulsion paths.
   - No prohibited energy combinations.
   - No single point of failure.
4. Generate PlantUML diagram after validation.
-----------------------------------------------------------------------
8. MICRO TEMPLATES
-----------------------------------------------------------------------
Basic Propeller
@startuml
component "Airframe\nPMX, Aero, states" as Airframe1 #LightBlue
component "Propeller\nPMR,PMX" as Propeller1 #LightBlue
Propeller1 -down- Airframe1 : PMX- PMX
title Aircraft System with Airframe and Propeller
@enduml
Battery to Motor
@startuml
component "Battery\nPEL, SOC" as Battery1 #LightGreen
component "Electric Motor Gen\nPEL, PMR, ctrl" as EMG1 #LightGreen
component "Propeller\nPMR,PMX" as Propeller1 #LightBlue
component "Gearbox\nPMR1,PMR2" as Gearbox1 #LightBlue
Battery1 -right- EMG1 : PEL- PEL
EMG1 -down- Gearbox1 : PMR- PMR
Gearbox1 -down- Propeller1 : PMR- PMR
title Aircraft System with Electric Motor and Propeller
@enduml
Turbo Prop
@startuml
component "Propeller\nPMR,F" as Propeller1 #LightBlue
component "Gearbox\nPMR1,PMR2" as Gearbox1 #LightBlue
component "FuelTank\nQM" as FuelTank1 #LightCoral
component "GasTurbine\nPQM, PMR, ctrl" as GasTurbine1 #LightCoral
FuelTank1 -right- GasTurbine1 : PQM - PQM
GasTurbine1 -down- Gearbox1 : PMR- PMR1
Gearbox1 -down- Propeller1 : PMR2- PMR
title Aircraft System with Gas Turbine and Propeller
@enduml
Hybrid Electric with Fuel Cell
```

```
@startuml
component "FuelCell\nPEL" as FuelCell1 #LightGreen
component "Hydrogen Storage\nPQM" as HydrogenStorage1 #Turquoise
component "Electric Motor/Generator\nPEL, PMR" as EMG1 #LightGreen
component "Electric PowerManagement\nPEL" as EPM1 #LightGreen
component "Battery Pack\nEL" as Battery1 #LightGreen
component "Controller\nctrl" as Controller1 #LightYellow
HydrogenStorage1 -down- FuelCell1 : PQM
FuelCell1 -down- EPM1 : PEL
EPM1 -down- Battery1 : PEL
Battery1 -down- EMG1 : PEL
Controller1 -[dashed]-> FuelCell1 : ctrl
Controller1 -[dashed]-> EMG1 : ctrl
Battery1 -[dashed]-> Controller1 : SOC
title Hybrid Electric Aircraft Propulsion System with Fuel Cell
@enduml
------------------------------------------------------------------------
9. FINAL VALIDATION CHECK
------------------------------------------------------------------------
- [ ] Mandatory components included.
- [ ] Two independent propulsion branches.
- [ ] Storage requirements satisfied.
- [ ] No single point of failure.
- [ ] Type-consistent port connections.
- [ ] Airframe connected.
- [ ] 14 CFR 25 compliance satisfied.
END OF CONTEXT
```