

Modeling System Architecture Design Spaces with ADORE

ADORE Modeling Workshop | 2026-01-22

Jasper Bussemaker (jasper.bussemaker@dlr.de)

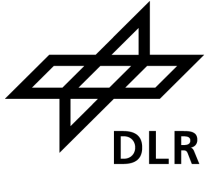
DLR Institute of System Architectures in Aeronautics (SL), Hamburg, Germany



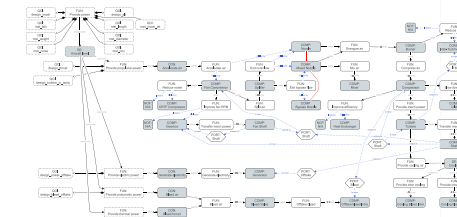
<https://adore.mbse-env.com/docs/>



Workshop Contents



- Modeling the Architecture Design Space
 - What model elements are available?
 - How to formulate architectural choices?
- Defining the Design Problem
 - How to test the design space model?
 - How to formulate design variables, objectives and constraints?



$$\begin{aligned} &\text{minimize} && f_m(\mathbf{x}, \mathbf{y}), && m = 1, 2, \dots, M \\ &\text{where} && \mathbf{x}_i \in \mathbb{R} && i = 1, 2, \dots, n \\ &&& \mathbf{y}_j \in \mathbb{Z} && j = 1, 2, \dots, \mathcal{J} \\ &\text{w.r.t.} && g_k(\mathbf{x}, \mathbf{y}) \leq 0, && k = 1, 2, \dots, \mathcal{K} \\ &&& x_i^{(L)} \leq x_i \leq x_i^{(U)} \end{aligned}$$

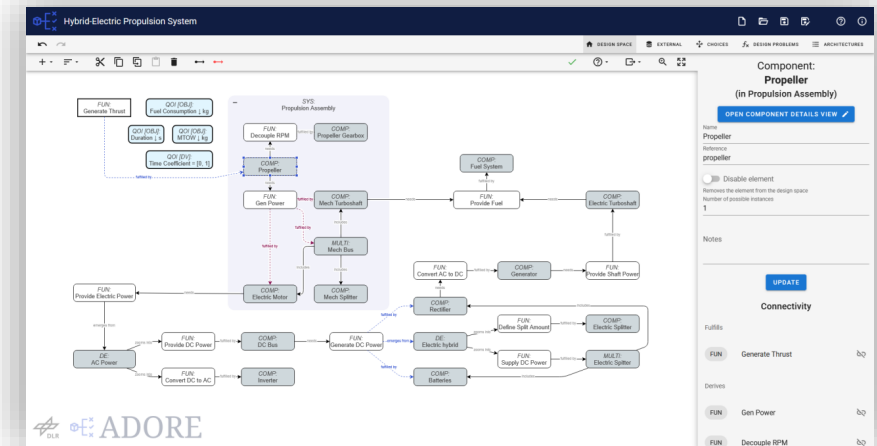


ADORE documentation:

<https://adore.mbse-env.com/docs/>

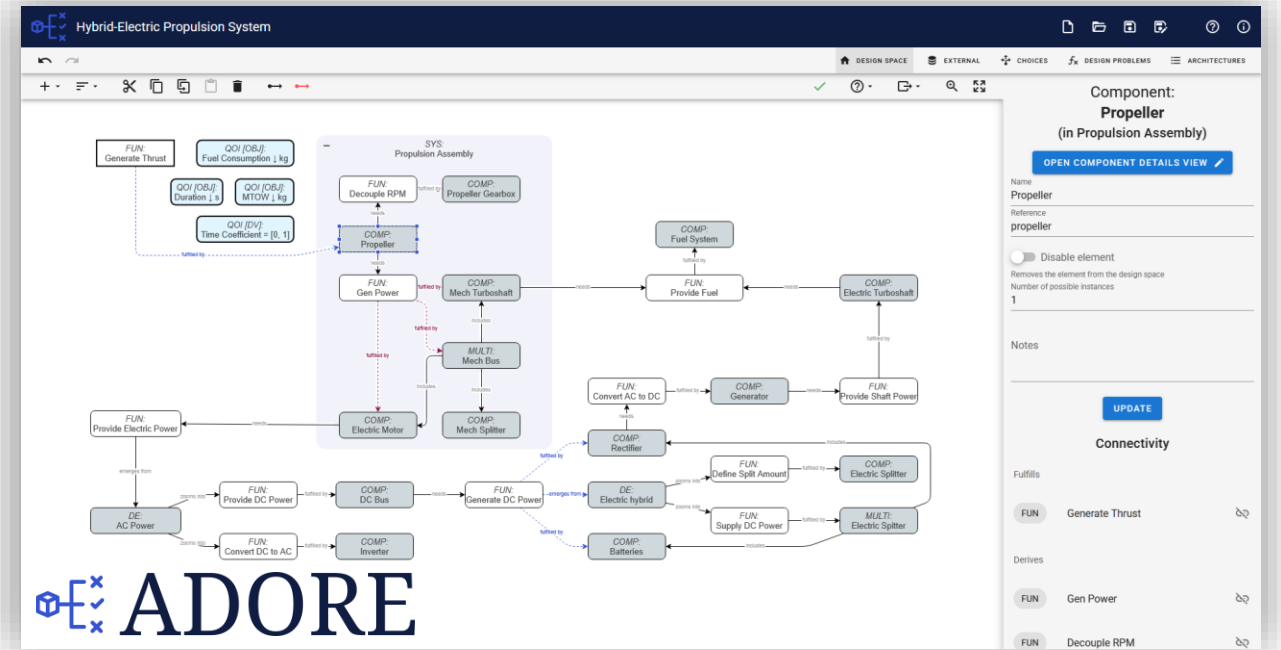
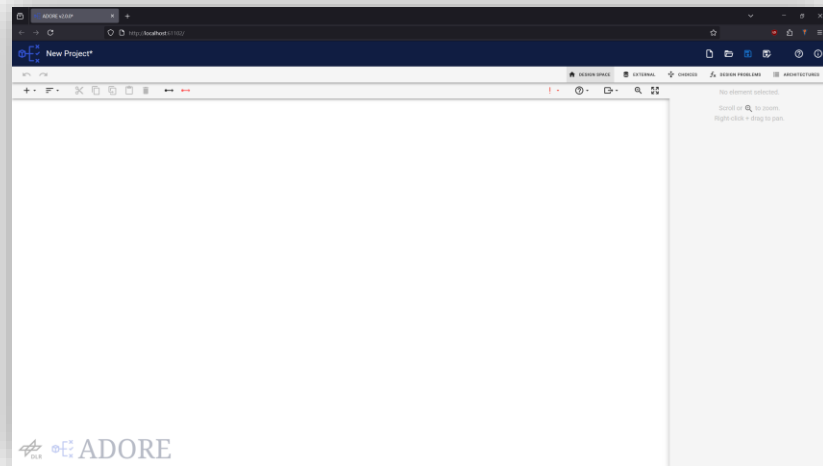
Background and theory:

[System Architecture Optimization: Function-Based Modeling, Optimization Algorithms, and Multidisciplinary Evaluation](https://adore.mbse-env.com/docs/)



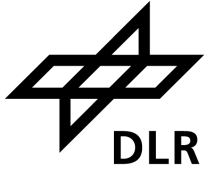
Running ADORE

- Install according to the instructions in the documentation
- Run ADORE



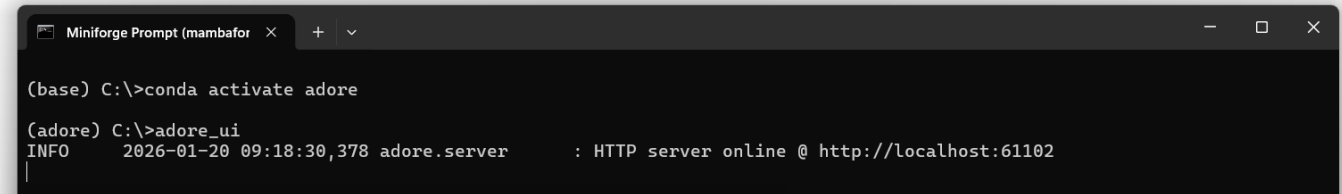
<https://adore.mbse-env.com/docs/>

Installing and Running ADORE Locally

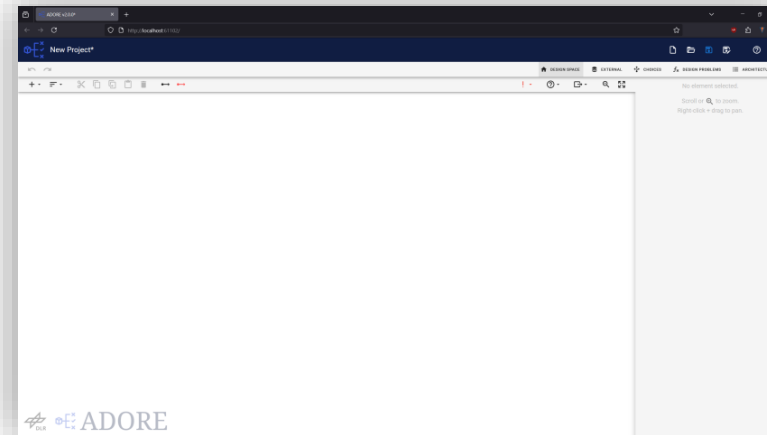


Clone the ADORE repo from Gitlab and install locally

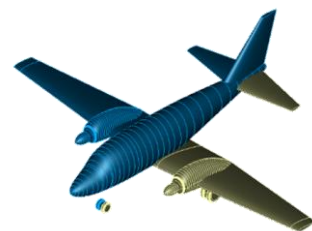
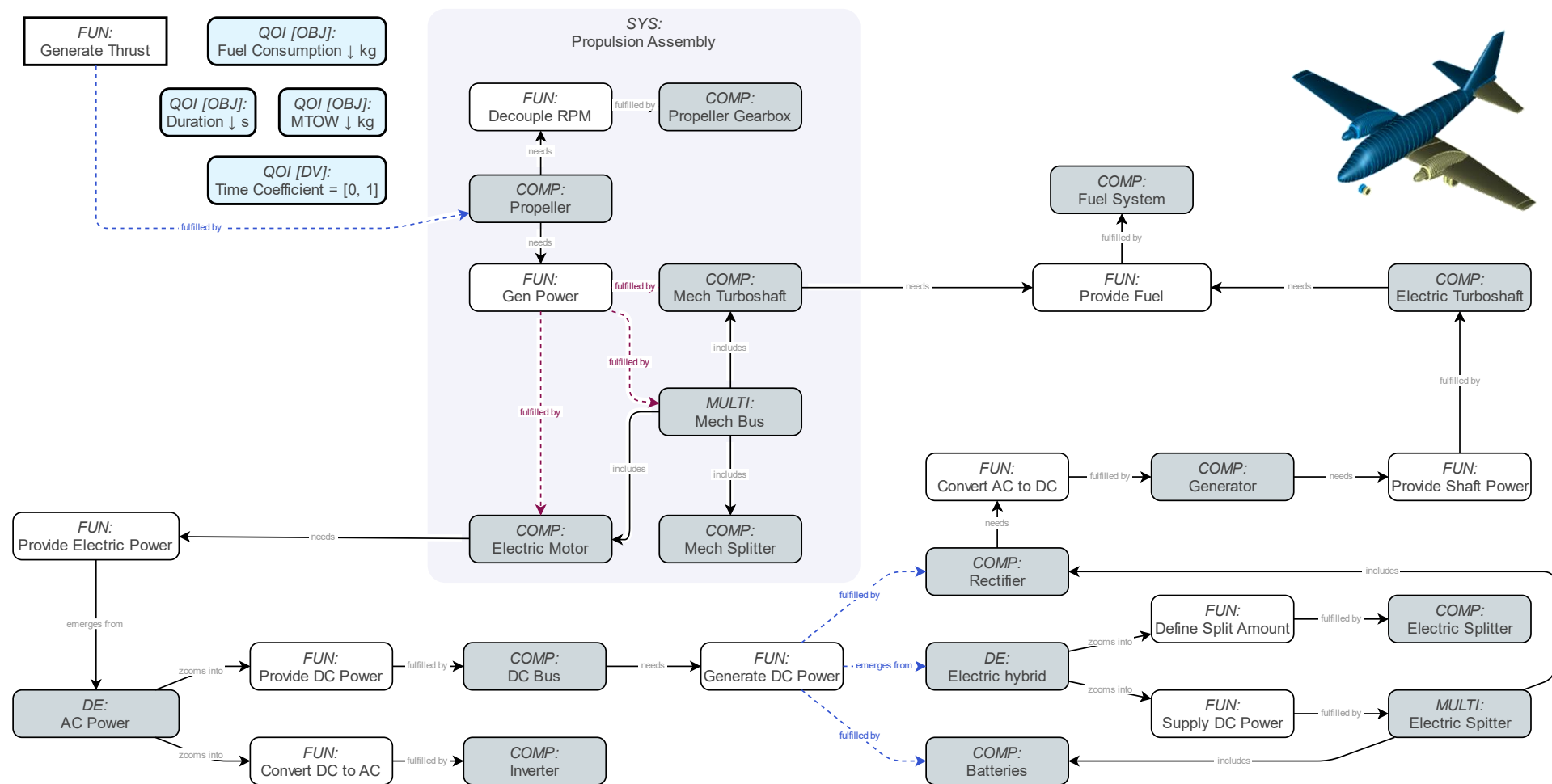
- `git clone https://gitlab.dlr.de/buss\_js/adore.git`
- `cd adore`
- `git fetch --all`
- `git checkout dev`
- `conda create -n adore python=3.12`
- `conda activate adore`
- `pip install .[gui]`
- `adore_ui`

A screenshot of a Windows terminal window titled "Miniforge Prompt (mambafor)". The terminal shows the following commands and output:

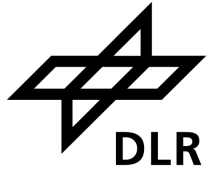
```
(base) C:\>conda activate adore
(adore) C:\>adore_ui
INFO      2026-01-20 09:18:30,378 adore.server      : HTTP server online @ http://localhost:61102
```



Workshop System of Interest: Hybrid-Electric Propulsion System



Main Editor Features

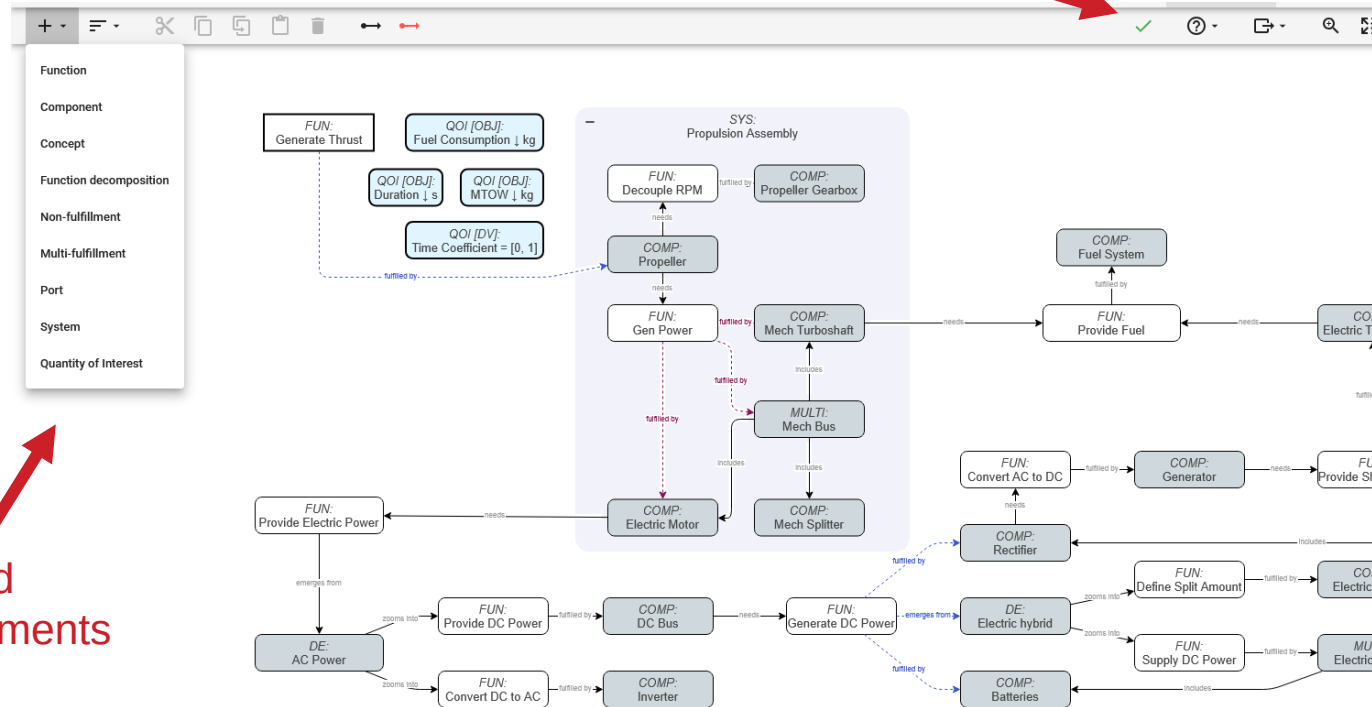


Auto layout

Cut/copy/duplicate/delete selected element

Model status indicator

Element quick reference



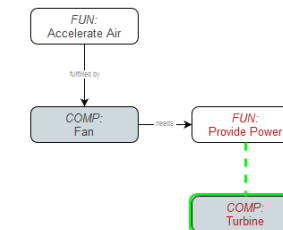
Add elements

Draw connection

Draw incompatibility constraint



Drag from source to target



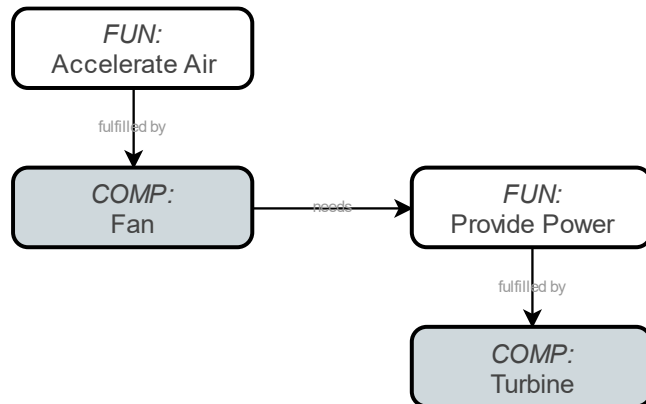
The highlight color indicates if a connection is allowed



System Functions

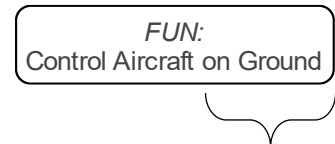
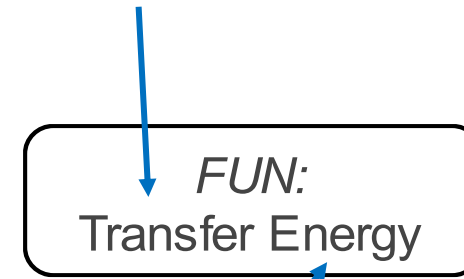
Functions represent **what** the system does.

Functions **are fulfilled** by one or more components, concepts, or decompositions.



Process

The action (verb) associated to the function
("what is happening ...")



Condition (optional)

Operating condition where the function is active

Operand

The object (noun) that the process is applied to
("... to what")

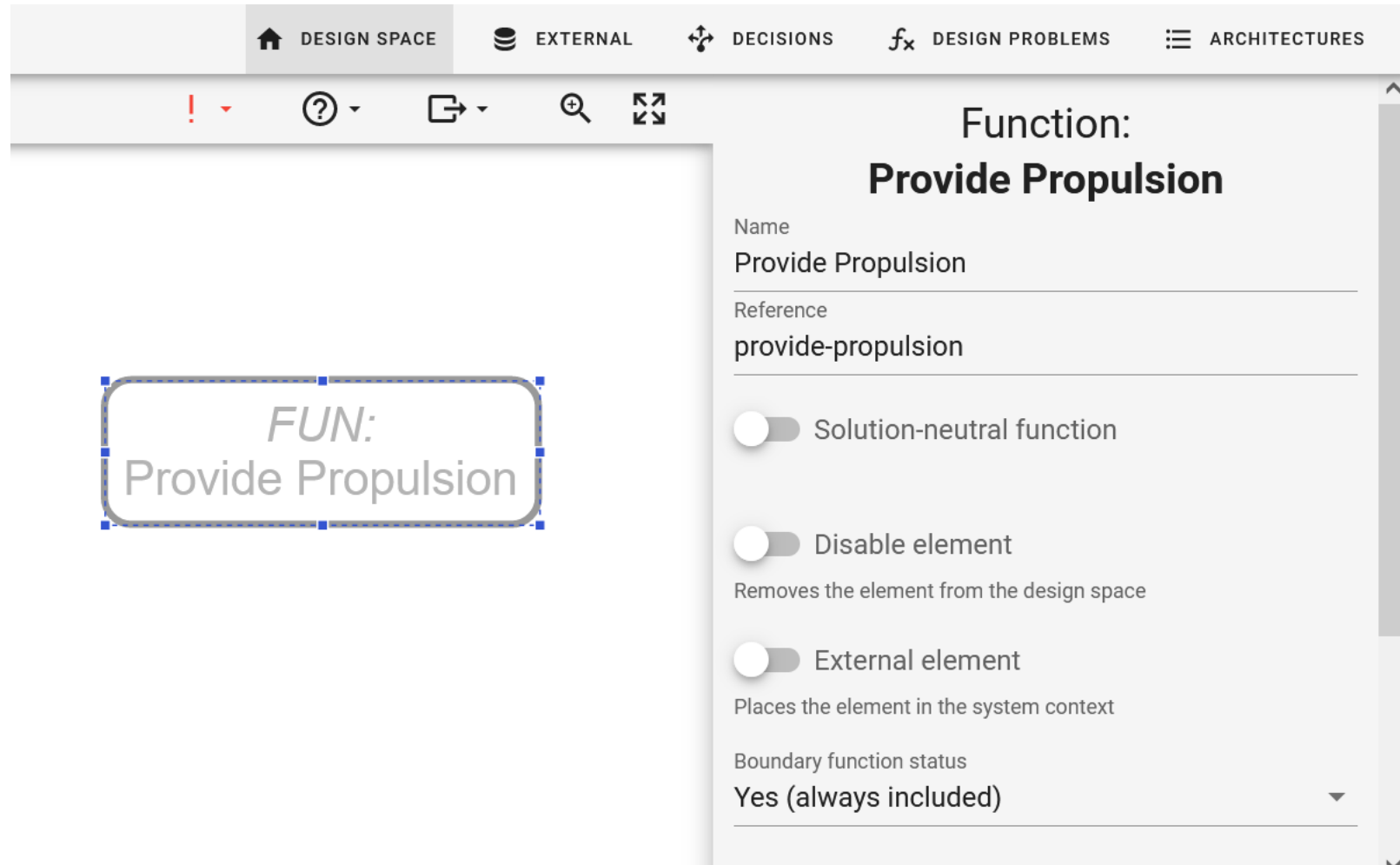
Exercise: Add Boundary Function



- Create a boundary function called “Provide Propulsion”
- Manually define it to be a boundary function
- Remember to press update to save all the changes

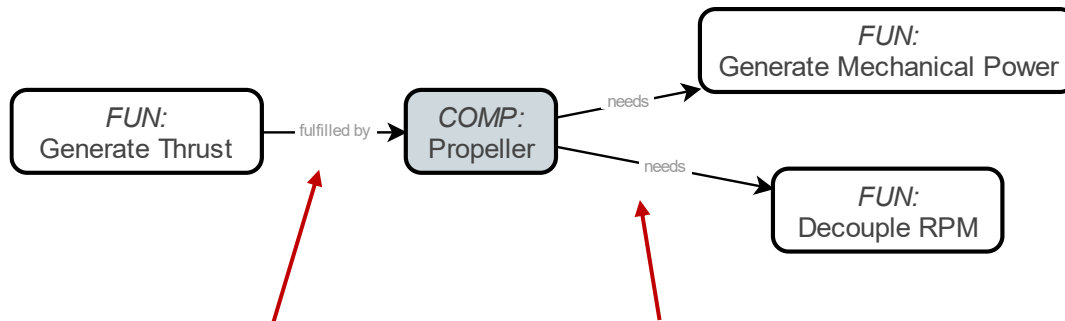
Exercise: Add Boundary Function

- Create a boundary function called “Provide Propulsion”
- Manually define it to be a boundary function
- Remember to press update to save all the changes



Components

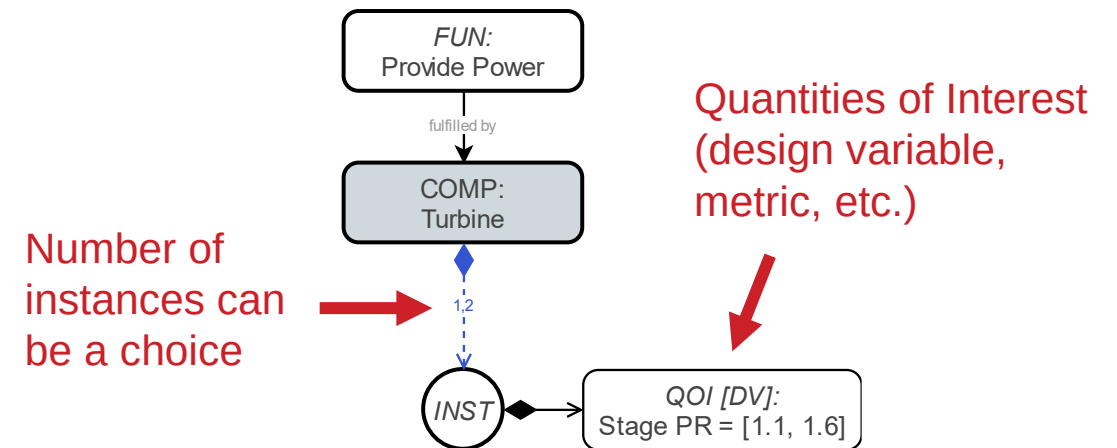
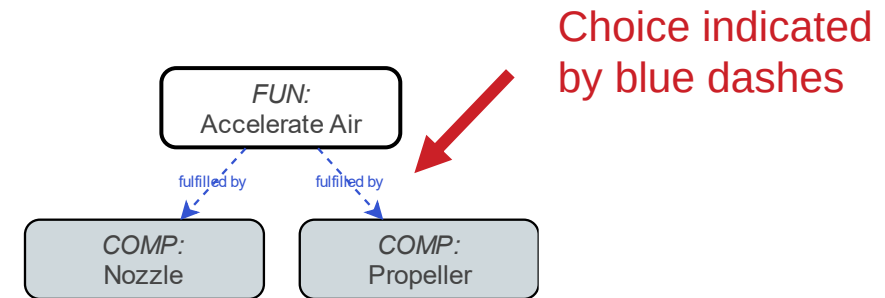
Components represent **how** the system fulfills its functions.
Components **fulfill** functions and may **derive (induce/need)** additional functions.



Function fulfillment
The function is fulfilled by the component.

Function induction
The components needs additional functions to be fulfilled.

Multiple components fulfilling a function leads to an **architecture choice**



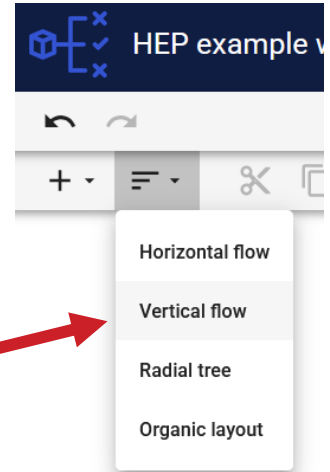
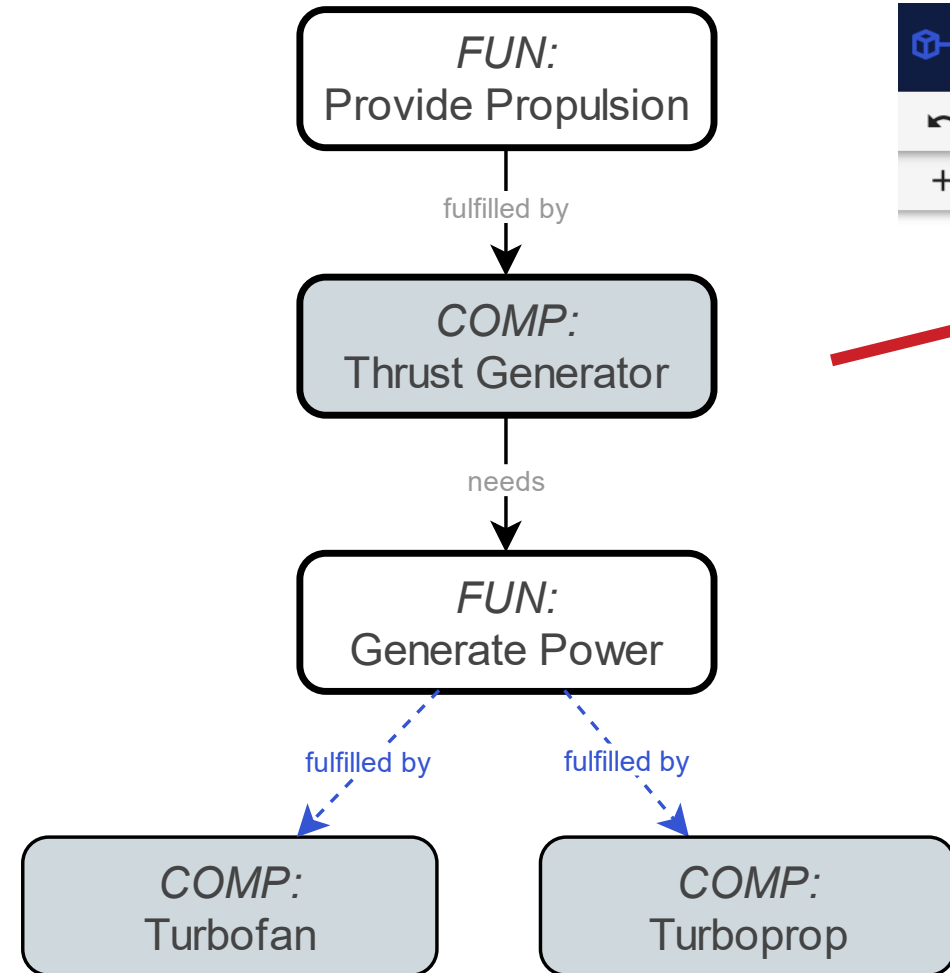
Exercise: Add Main Components



- Add a component called “Thrust Generator” to fulfill the boundary function
- Add an induced function called “Generate Power”
- Add two components that can fulfill this induced function:
 - Turbofan and Turboprop

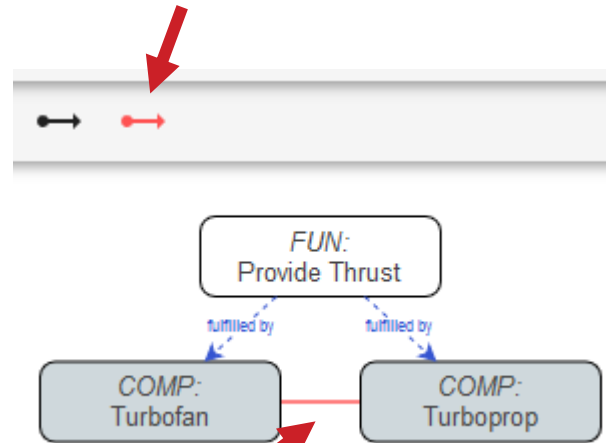
Exercise: Add Main Components

- Add a component called “Thrust Generator” to fulfill the boundary function
- Add an induced function called “Generate Power”
- Add two components that can fulfill this induced function:
 - Turbofan and Turboprop



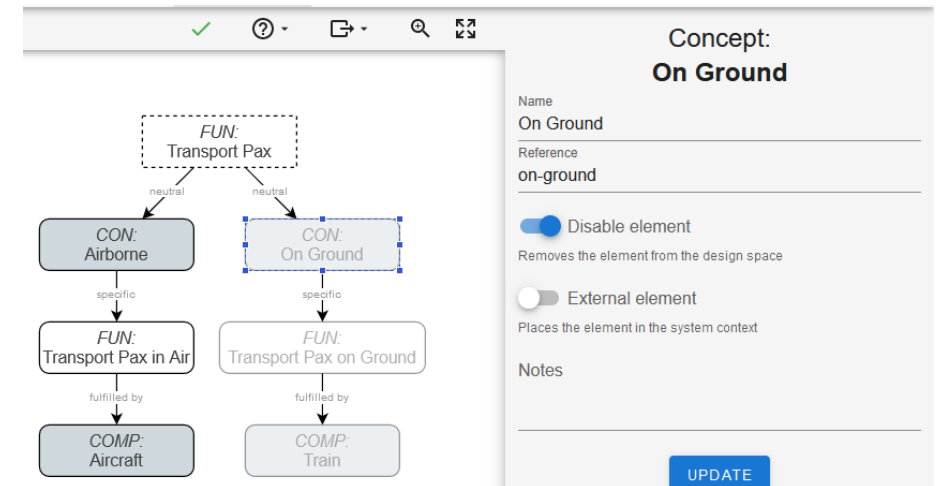
Incompatibility and Disabling Elements

Incompatibility constraints explicitly define that two elements (non transitive) **cannot exist together** in an architecture.



An incompatibility constraint

Elements (and derived elements) can be explicitly **removed** from the design space. This is helpful for documentation.



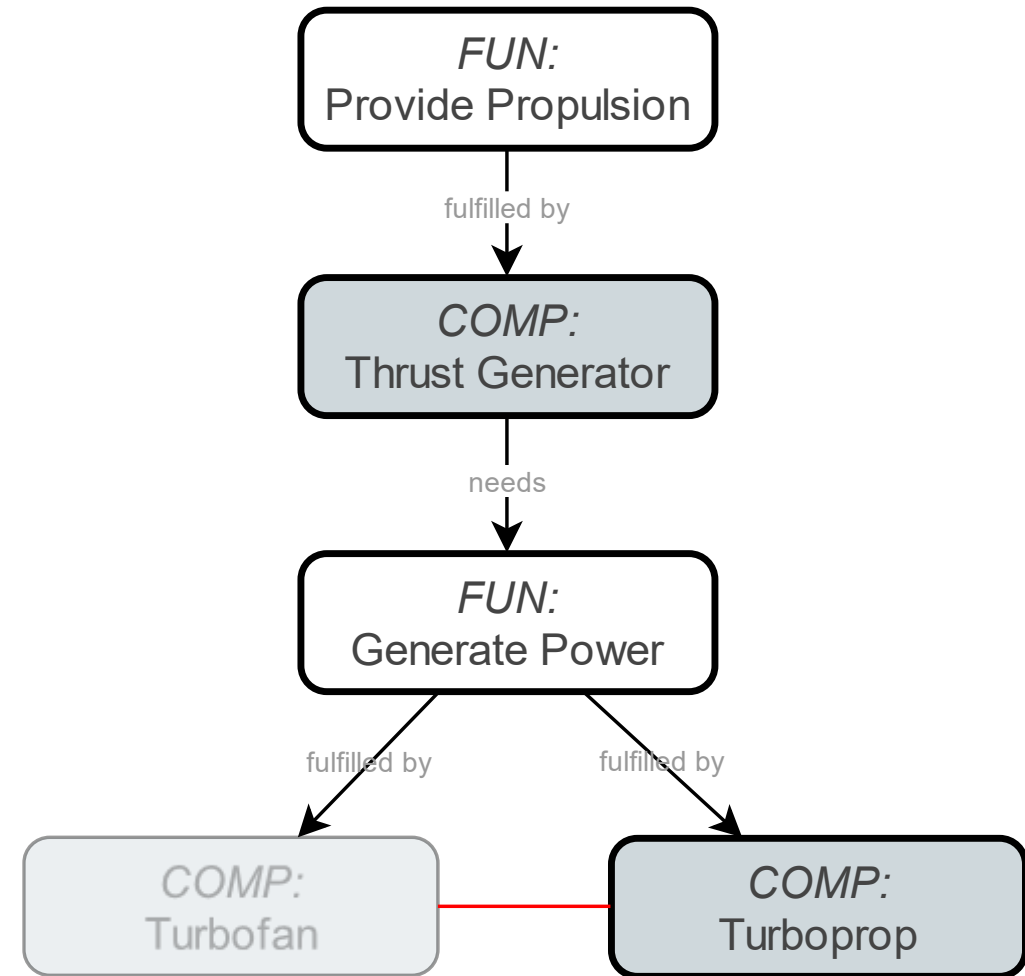
Exercise: Disable and Define Incompatibility



- Define that Turbofan and Turboprop are mutually incompatible
- Disable the Turbofan

Exercise: Disable and Define Incompatibility

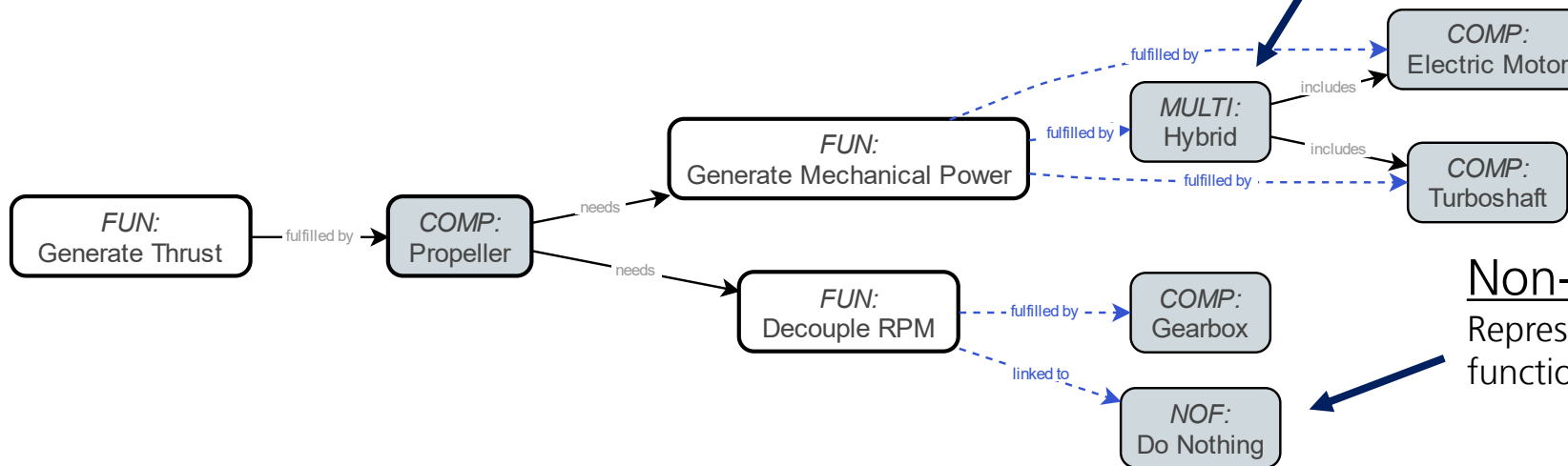
- Define that Turbofan and Turboprop are mutually incompatible
- Disable the Turbofan



Non- and Multi-Fulfillments

Multi-fulfillments

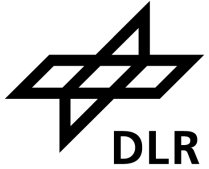
Represent the option for *multiple* components to fulfill a given function.



Non-fulfillments

Represent the option for a function to remain *unfulfilled*.

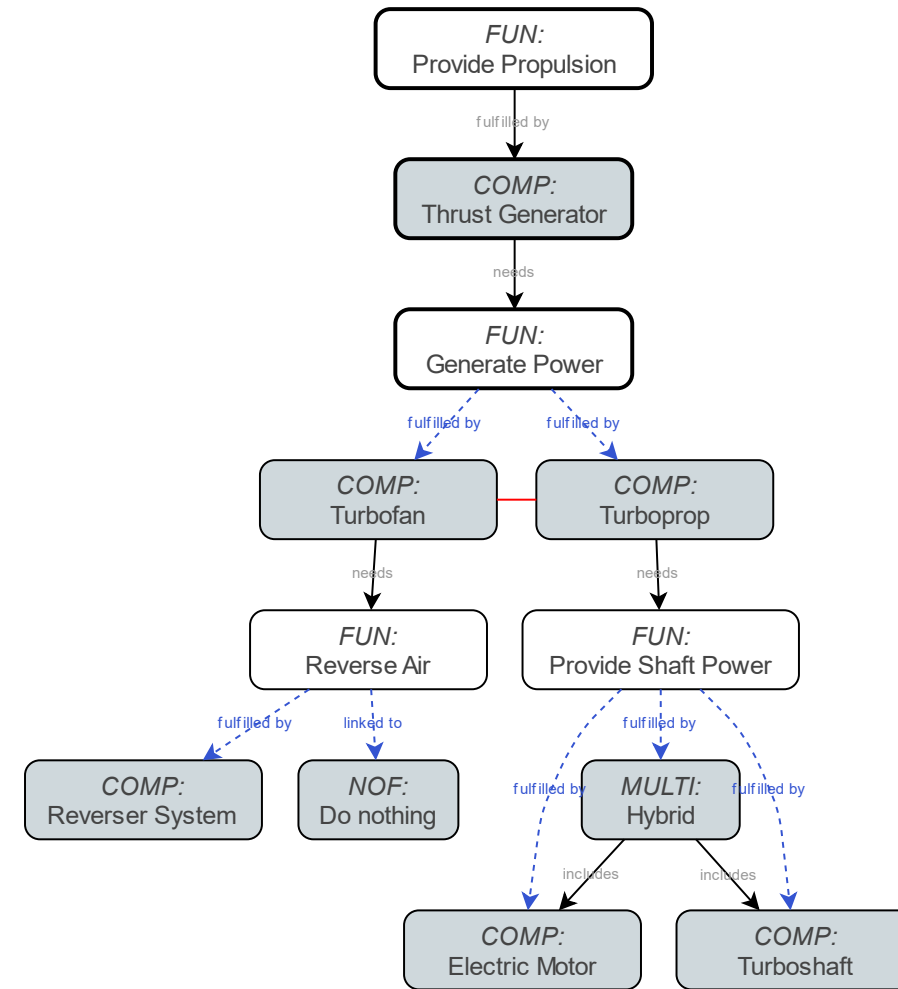
Exercise: Define Optional and Hybrid Choices



- Induce functions:
 - Turbofan → “Reverse Air”
 - Turboprop → “Provide Shaft Power”
- Define “Reverse Air” to be fulfilled either by a “Reverser System” or not
- Define “Provide Shaft Power” to be fulfilled by:
 - An “Electric Motor”
 - A “Turboshaft”
 - Both of the above (“Hybrid”)

Exercise: Define Optional and Hybrid Choices

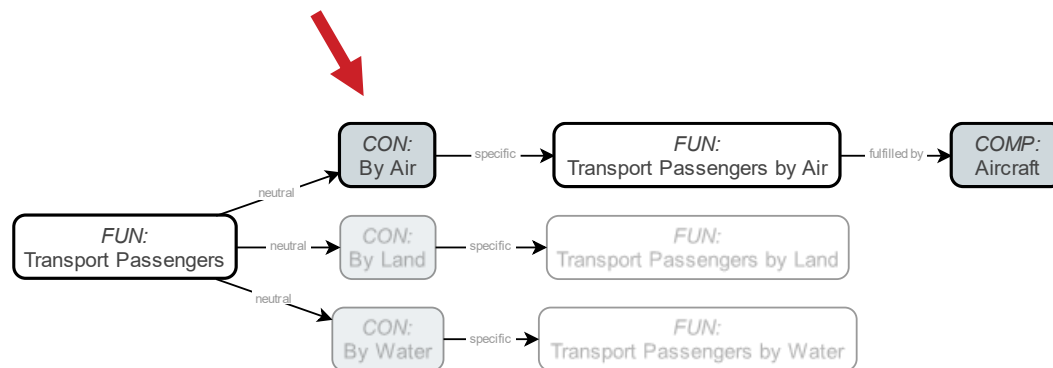
- Induce functions:
 - Turbofan → “Reverse Air”
 - Turboprop → “Provide Shaft Power”
- Define “Reverse Air” to be fulfilled either by a “Reverser System” or not
 - An “Electric Motor”
 - A “Turboshaft”
 - Both of the above (“Hybrid”)



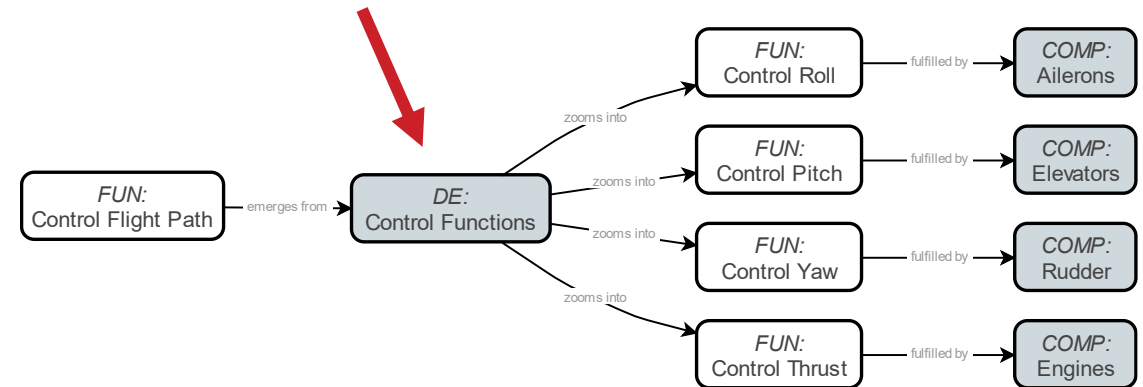
Concepts and Function Decompositions

Concepts and decompositions map functions to functions, and can be used as **complexity management tools**.

A concept mapping a solution-neutral function to a solution-specific functions



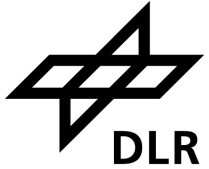
A decomposition mapping one to multiple functions



Only use a decomposition if the following conditions are satisfied:

1. *Parallelism*: the decomposed functions act in **parallel**, or in a **tightly-coupled loop** without a clear starting point.
2. *Solution-neutrality*: the decomposed functions are as **solution-neutral** as the parent function, not implying any technology/solution selection.
3. *Compatibility*: the decomposition does not require major cross-tree **incompatibility constraints** to be added later on.

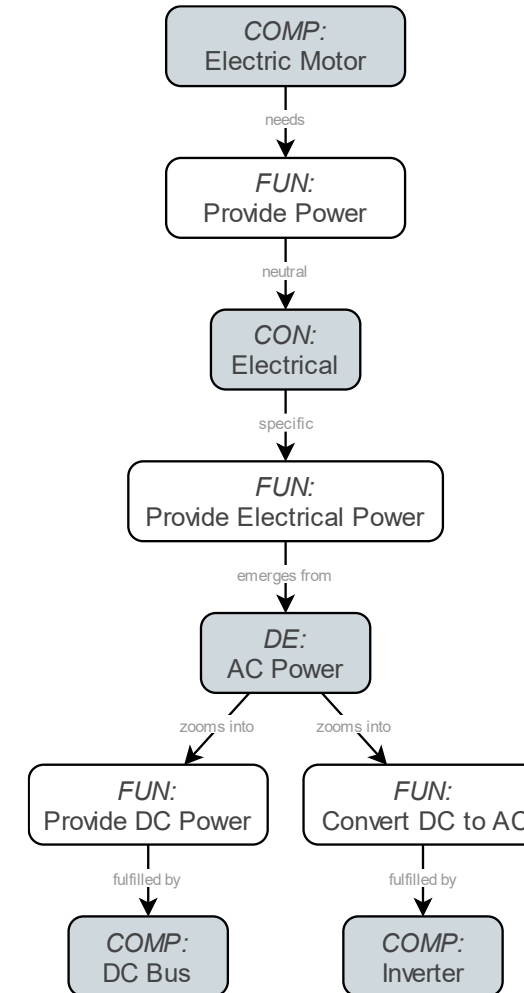
Exercise: Define Concepts and Decompositions



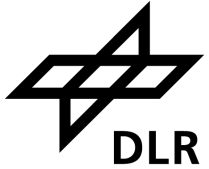
- The Electric Motor induces the “Provide Power” function
- Define a concept that maps “Provide Power” to “Provide Electrical Power”
- Decompose “Provide Electrical Power” into:
 - “Provide DC Power”, fulfilled by “DC Bus”
 - “Convert DC to AC”, fulfilled by “Inverter”

Exercise: Define Concepts and Decompositions

- The Electric Motor induces the “Provide Power” function
- Define a concept that maps “Provide Power” to “Provide Electrical Power”
- Decompose “Provide Electrical Power” into:
 - “Provide DC Power”, fulfilled by “DC Bus”
 - “Convert DC to AC”, fulfilled by “Inverter”



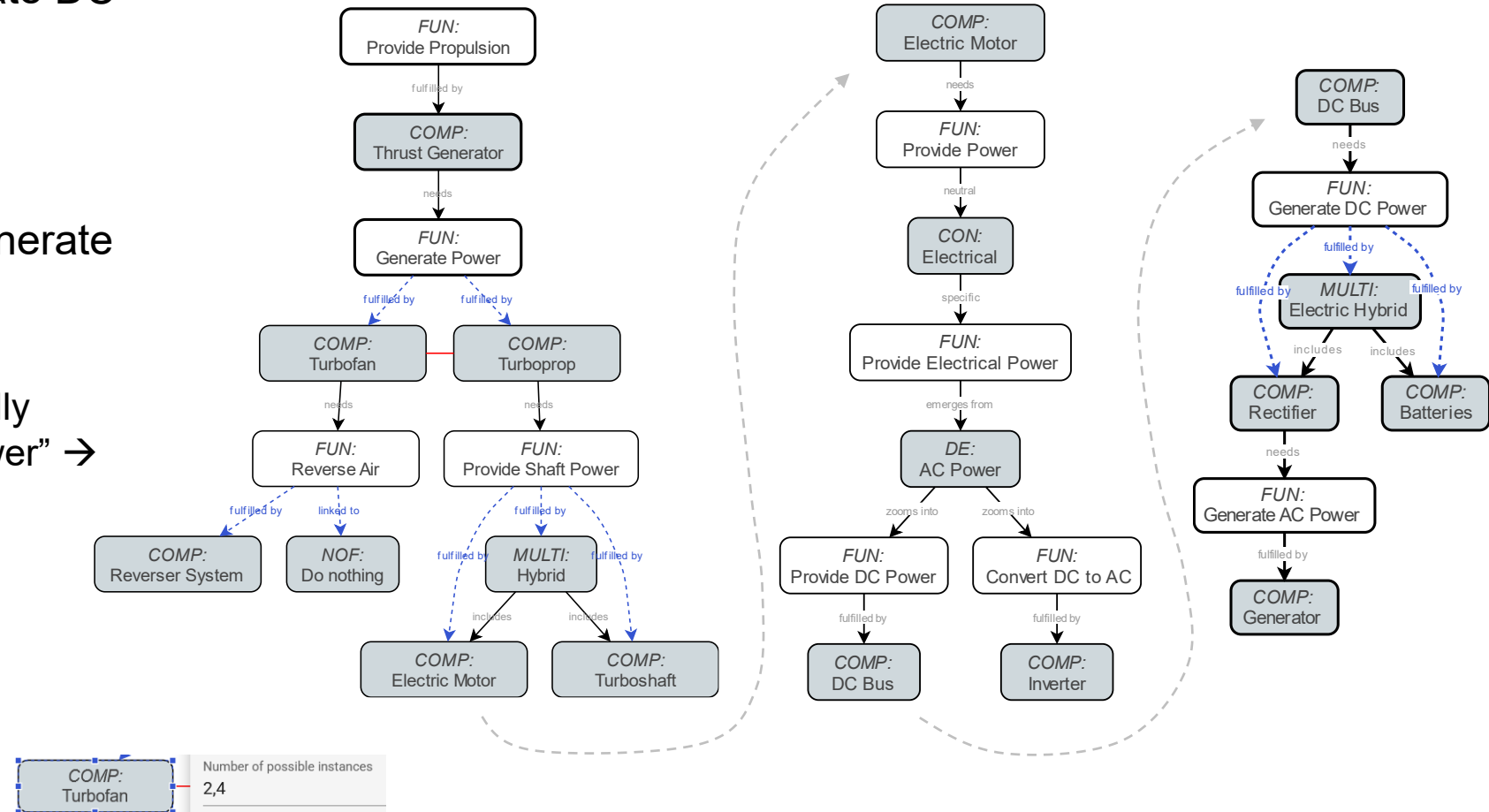
Exercise: Practice



- DC Bus induces the “Generate DC Power” function
- Add another hybrid function fulfillment choice for the “Generate DC Power” function:
 - “Batteries”
 - “Rectifier”, which additionally induces “Generate AC Power” → “Generator”
 - Or both (“Electric Hybrid”)
- Set the number of possible “Turbofans” to be: 2 or 4

Exercise: Practice

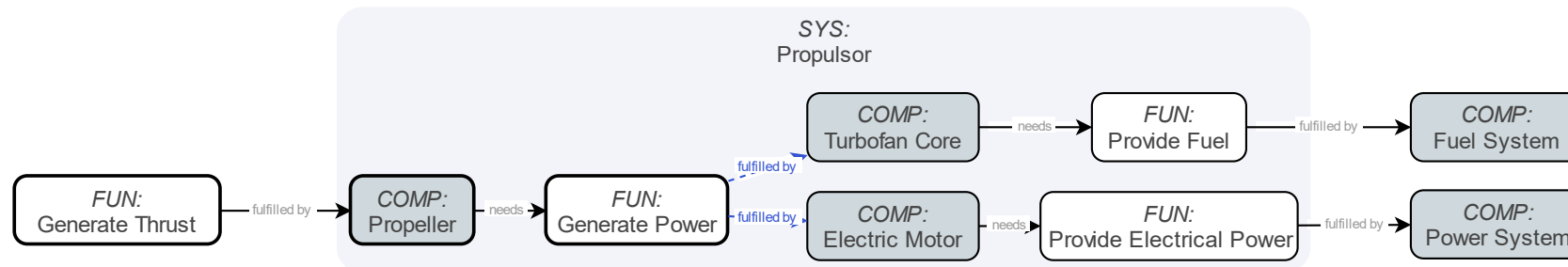
- DC Bus induces the “Generate DC Power” function
- Add another hybrid function fulfillment choice for the “Generate DC Power” function:
 - “Batteries”
 - “Rectifier”, which additionally induces “Generate AC Power” → “Generator”
 - Or both (“Electric Hybrid”)
- Set the number of possible “Turbofans” to be: 2 or 4



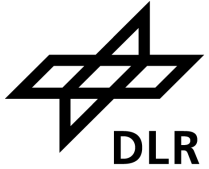
Subsystems

Subsystems represent groups of elements, that can be **instantiated** multiple times. Choices within system instances are independent.

Subsystems can recursively own other subsystems.



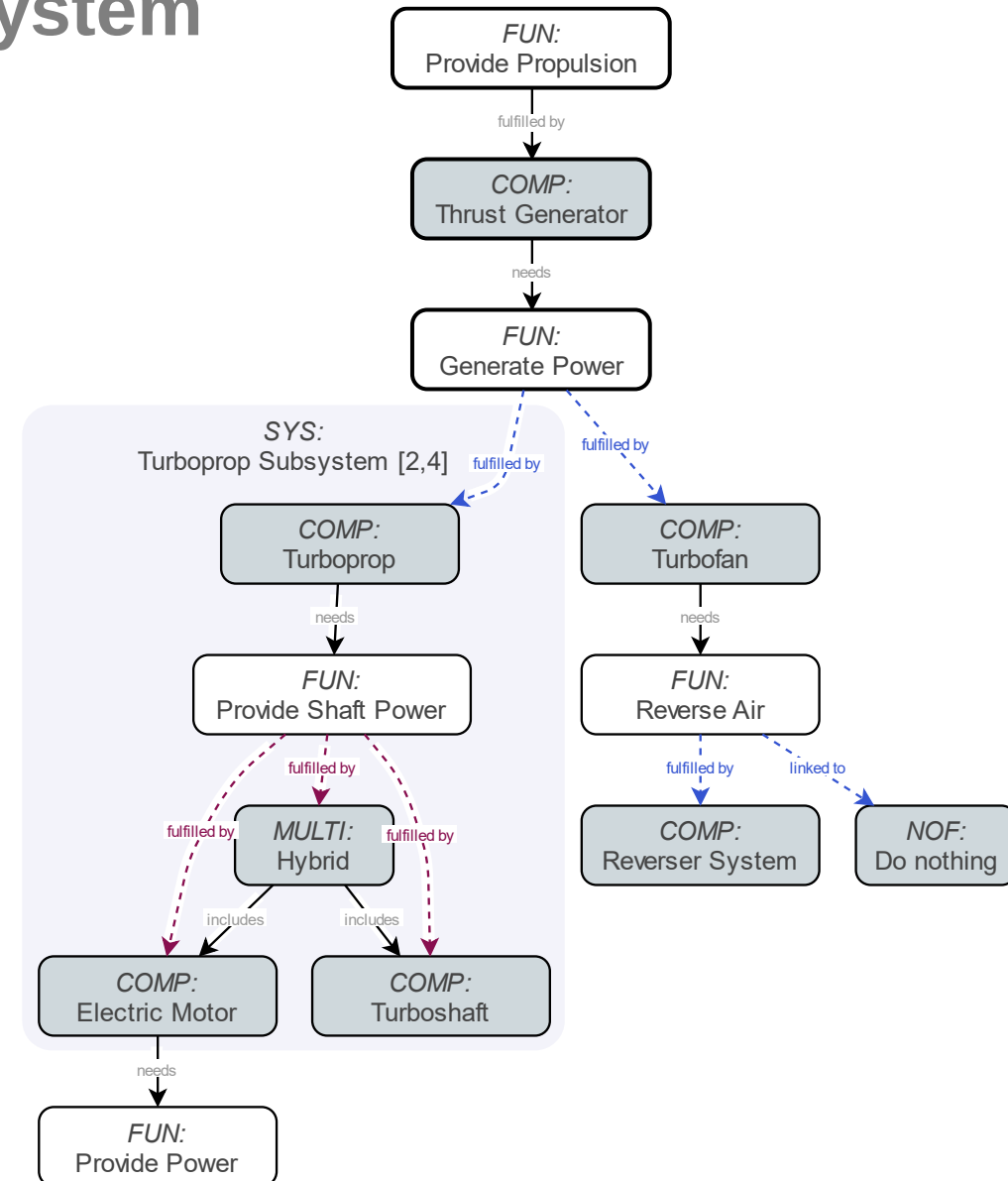
Exercise: Create a Turboprop System



- Create a Turboprop subsystem, include:
 - Turboprop
 - Provide Shaft Power
 - Subsequent fulfillment options
- Tip: *first* create the subsystem block, *then* drag and drop elements inside
- Set the number of possible instances to be 2 or 4

Exercise: Create a Turboprop System

- Create a Turboprop subsystem, include:
 - Turboprop
 - Provide Shaft Power
 - Subsequent fulfillment options
- Tip: *first* create the subsystem block, *then* drag and drop elements inside
- Set the number of possible instances to be 2 or 4



Quantities of Interest (QOIs)

QOIs represent quantifiable inputs or outputs to/from the architecture evaluation.

The type can be changed easily.

QOIs can be owned by the top-level system, subsystems, functions, or components.

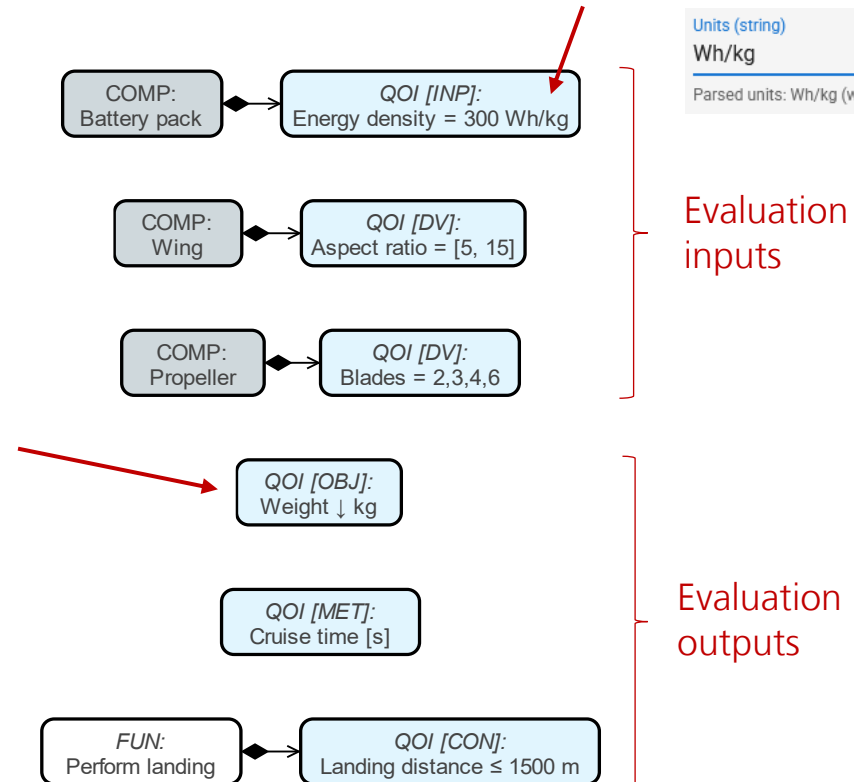
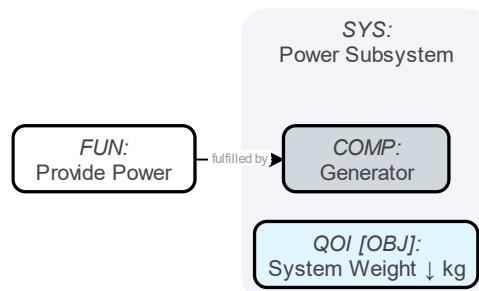
Units

QOIs can have units defined.
Unit conversions are supported.

Units (string)
Wh/kg
Parsed units: Wh/kg (watt_hour/kilogram; [length]²/[time]²)

Objective QOIs

QOIs can only be objectives if they *always* exist, as needed for architecture performance comparisons.



Exercise: Define Quantities of Interest



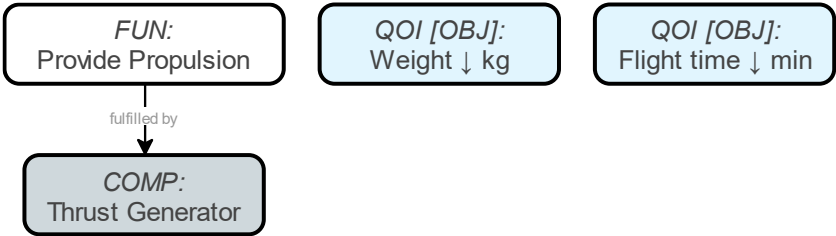
Owner	Name	Type	Value(s)	Units
Top-level system	Weight	Objective	Minimized	kg
	Flight time	Objective	Minimized	min
Batteries	Weight	Design Var	[500,1000]	kg
	Final Load	Constraint	≥ 0	

Tip: to add a QOI to a component, first you need to go into the “component details view”

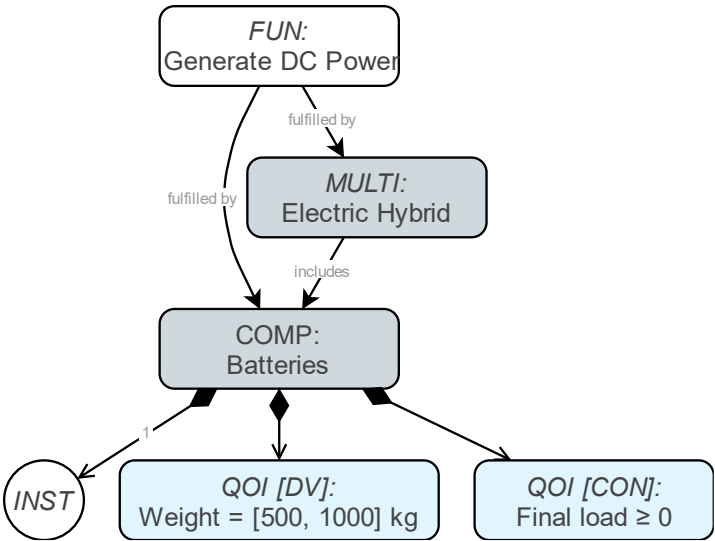
Exercise: Define Quantities of Interest



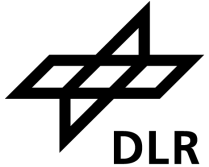
Owner	Name	Type	Value(s)	Units
Top-level system	Weight	Objective	Minimized	kg
	Flight time	Objective	Minimized	min
Batteries	Weight	Design Var	[500,1000]	kg
	Final Load	Constraint	≥ 0	



Tip: to add a QOI to a component, first you need to go into the “component details view”

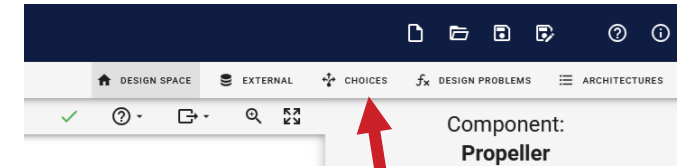


Inspecting Architectural Choices



Gives an overview of defined architectural choices. Also allows defining constraints between choices.

Statistics about the design space can be calculated. The “valid” design space size represents the number of valid (discrete) architectures in the design space.



Click to refresh statistics

Architecture Choices						
Search						
# ↑	Operation	Subject	Component Instance	System Instance	Options	Constraints
1	Fulfill function	Reverse Air			Do nothing, Reverser System	↔
2	Fulfill function	Generate DC Power			Batteries, Electric Hybrid, Rectifier	↔
3	Fulfill function	Provide Shaft Power		1	Electric Motor, Hybrid, Turboshaft	↔
4	Fulfill function	Provide Shaft Power		2	Electric Motor, Hybrid, Turboshaft	↔
5	Fulfill function	Provide Shaft Power		3	Electric Motor, Hybrid, Turboshaft	↔
6	Fulfill function	Provide Shaft Power		4	Electric Motor, Hybrid, Turboshaft	↔
7	Fulfill function	Generate Power			Turboprop, Turboshaft	↔
8	Instantiate system	Turboprop Subsystem				↔
11	Instantiate component	Turbofan				↔
12	Continuous design variable	COMP: Batteries -> Weight				↔

Choice Constraint Types

Choices can be linked by a constraint, which limits the options that can be selected when taking all choices within a constraint into account.

As an example, consider two choices with three options (A, B, C) each:

Constraint	Description	Possible Combinations
None	All combinations of options	AA, AB, AC, BA, BB, BC, CA, CB, CC (9)
Linked Values	All choices get the same option	AA, BB, CC (3)
Permutations	All combinations without repeating values	AB, BA, AC, CA, BC, CB (6)
Unordered Combinations	All combinations when selection order is not relevant	AA, AB, AC, BB, BC, CC (6)
Unordered Non-replacing Combinations	Unordered, without repeating values	AB, AC, BC (3)

Statistics

Statistics	Value
Decisions	12
Valid Design Space (Discrete)	270
Declared Design Space (Discrete)	3 888
Discrete Choices	9
Discrete Imputation Ratio	14.4
Information Index ⓘ	73%
Continuous Dimensions	1
Mean Active Continuous Dimensions	0.7
Continuous Imputation Ratio	1.5
Imputation Ratio ⓘ	22.1
Encoding Algorithm ⓘ	Complete

Exercise: Constrain “Provide Shaft Power”



- Constrain the “Provide Shaft Power” choices to only accept “unordered combinations”

DESIGN SPACE			EXTERNAL	CHOICES	DESIGN PROBLEMS	ARCHITECTURES
			Statistics			
			Decisions			
			12			
			Valid Design Space (Discrete)			
			270			
			Declared Design Space (Discrete)			
			3 888			
			Discrete Choices			
			9			
			Discrete Imputation Ratio			
			14.4			
			Information Index			
			73%			
			Continuous Dimensions			
			1			
			Mean Active Continuous Dimensions			
			0.7			
			Continuous Imputation Ratio			
			1.5			
			Imputation Ratio			
			22.1			
			Encoding Algorithm			
			Complete			

System Instance	Options	Constraints
	Do nothing, Reverser System	
	Batteries, Electric Hybrid, Rectifier	
1	Electric Motor, Hybrid, Turboshaft	
2	Electric Motor, Hybrid, Turboshaft	
3	Electric Motor, Hybrid, Turboshaft	
4	Electric Motor, Hybrid, Turboshaft	
	Turbofan, Turboprop	
	2 or 4 times	
	2 or 4 times	
	Between 500 and 1000 [kg]	



Exercise: Constrain “Provide Shaft Power”



- Constrain the “Provide Shaft Power” choices to only accept “unordered combinations”

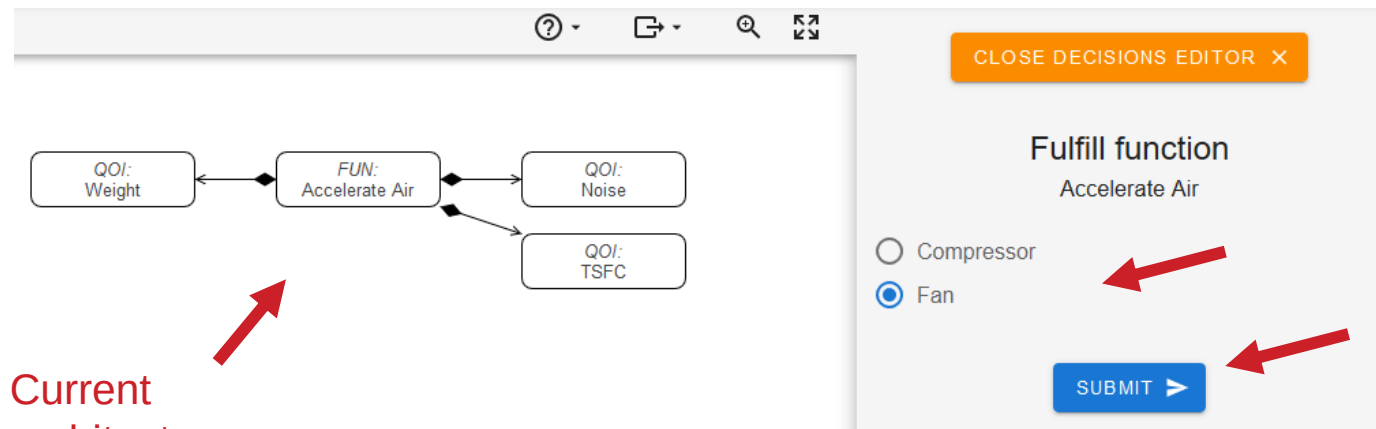
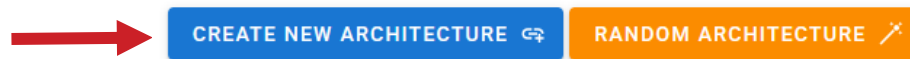
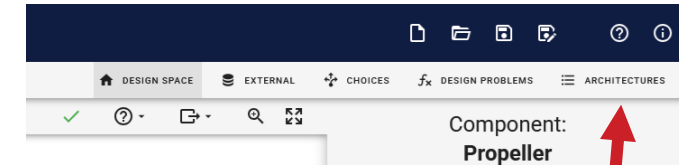
DESIGN SPACE			EXTERNAL	CHOICES	DESIGN PROBLEMS	ARCHITECTURES
			Statistics			
			Decisions			
			12			
			Valid Design Space (Discrete)			
			270			
			Declared Design Space (Discrete)			
			9			
			Discrete Choices			
			14.4			
			Discrete Imputation Ratio			
			73%			
			Information Index			
			1			
			Continuous Dimensions			
			0.7			
			Mean Active Continuous Dimensions			
			1.5			
			Continuous Imputation Ratio			
			22.1			
			Imputation Ratio			
			Complete			
			Encoding Algorithm			

#	Operation	Subject	Component Instance	System Instance	Options	Constraints
1	Fulfill function	Reverse Air			Do nothing, Reverser System	
2	Fulfill function	Generate DC Power			Batteries, Electric Hybrid, Rectifier	
3	Fulfill function	Provide Shaft Power		1	Electric Motor, Hybrid, Turboshaft	
4	Fulfill function	Provide Shaft Power		2	Electric Motor, Hybrid, Turboshaft	
5	Fulfill function	Provide Shaft Power		3	Electric Motor, Hybrid, Turboshaft	
6	Fulfill function	Provide Shaft Power		4	Electric Motor, Hybrid, Turboshaft	
7	Fulfill function	Generate Power			Turbofan, Turboprop	
8	Instantiate system	Turboprop Subsystem			2 or 4 times	
11	Instantiate component	Turbofan			2 or 4 times	
12	Continuous design variable	COMP: Batteries -> Weight			Between 500 and 1000 [kg]	

Statistics	
Decisions	12
Valid Design Space (Discrete)	63

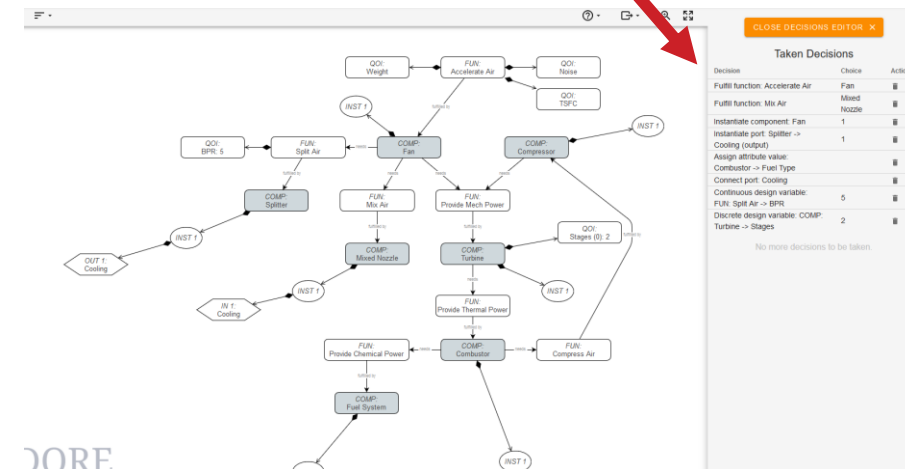
Manually Creating Architecture Instances

Manually creating architecture can be helpful for validating the design space and testing evaluation interfaces.



Current architecture instance

Until all choices have been taken



Exercise: Manually Creating Architectures

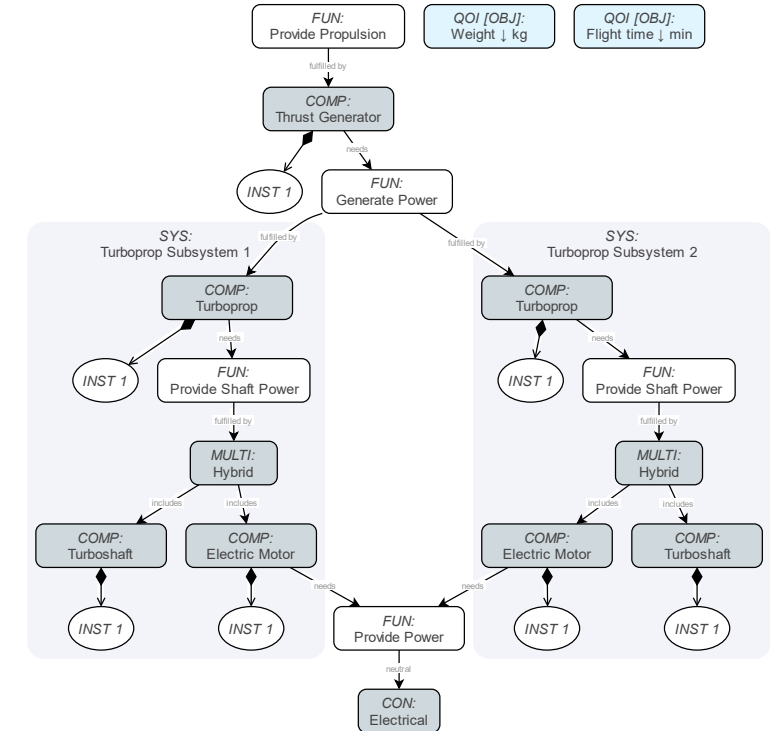
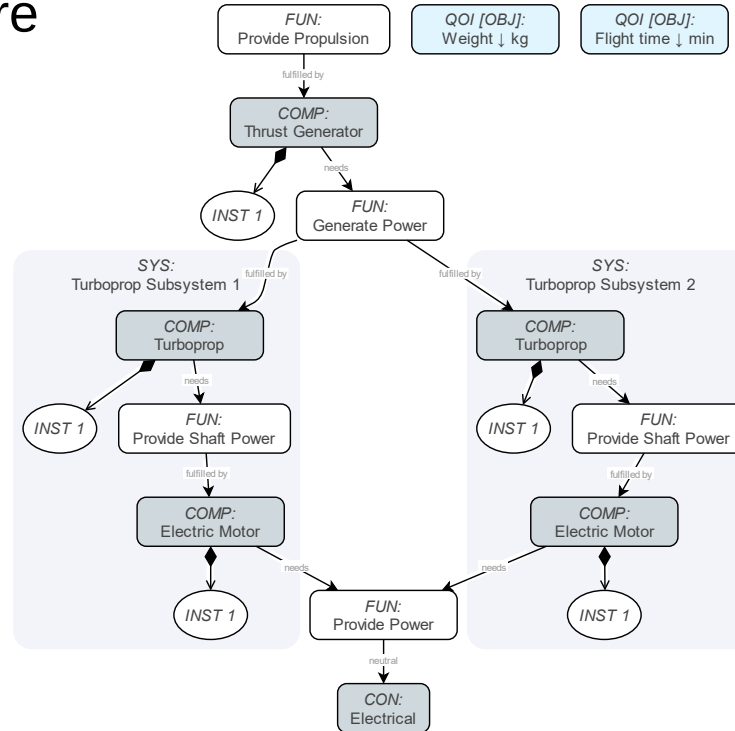


- Create the following architectures:
 - A fully electric architecture
 - A hybrid architecture

Exercise: Manually Creating Architectures

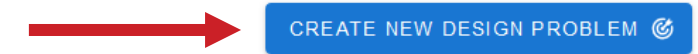
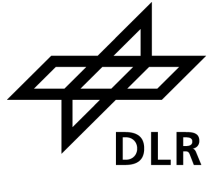
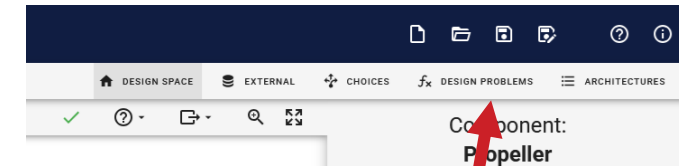
- Create the following architectures:

- A fully electric architecture
- A hybrid architecture



Defining Design Problems

Design problems are a **formalization** of the design space in terms of design variables, objectives, constraints. They are used by optimization algorithms.



Choices and QOIs mapped to design variables

QOIs mapped to objectives and constraints

#	Name	Type	Source	Options	Fixed Value	Actions
1	func_fulfill	Discrete	Decision #1	2	C: FAN	
2	func_fulfill_2	Discrete	Decision #2	2		
3	comp_instance_FAN	Discrete	Decision #3	2		
4	port_needed_COOLING	Discrete	Decision #4	2		
5	comp_atl_COMBUSTOR_FUEL_TYPE_0	Discrete	Decision #5	3		
6	port_COOLING_0	Discrete	Decision #6	5		
7	port_COOLING_1	Discrete	Decision #6	3		
8	BPR_0	Continuous	QOI: BPR	Between 3 and 12	5	
9	STAGES_0	Discrete	QOI: Stages	4		

#	Name	Source	Active	Direction	Actions
1	TSFC	QOI: TSFC	✓	Minimize	
2	Weight	QOI: Weight	✗	Minimize	

#	Name	Source	Active	Direction	Reference Value	Actions
1	Noise	QOI: Noise	✓	Lower than or equal to	105	

#	Name	Source	Actions
No data available			

Design Variable Settings

func_fulfill_2

Type: Discrete

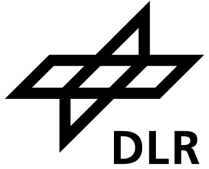
Fixed value

Notes

Optionally fix design variable values

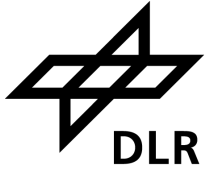
QOI type can be changed to objective, constraint or they might be deactivated

Exercise: Define a Design Problem



- Define a design problem
- Fix “Fulfill Generate Power” to always be the “Turboprop”
- Disable the “Flight Time” objective

Exercise: Define a Design Problem



- Define a design problem
- Fix “Fulfill Generate Power” to always be the “Turboprop”
- Disable the “Flight Time” objective

Design Problem: Design Problem #1

Search



BACK TO OVERVIEW ×

Design Variables

# ↑	Name	Type	Source	Options	Fixed Value	Actions
1	Fulfill Generate Power	Discrete	Choice #6	2	Turboprop	
2	Fulfill Reverse Air	Discrete	Choice #0	2		
3	Instantiate System Turboprop Subsystem	Discrete	Choice #7	2		
4	Instantiate Component Turbofan	Discrete	Choice #10	2		
5	Fulfill Provide Shaft Power 1	Discrete	Choice #2	3		
6	Fulfill Provide Shaft Power 2	Discrete	Choice #3	3		
7	Fulfill Provide Shaft Power 3	Discrete	Choice #4	3		
8	Fulfill Provide Shaft Power 4	Discrete	Choice #5	3		
9	Fulfill Generate DC Power	Discrete	Choice #1	3		
10	Design Variable (QOI) Batteries -> Weight	Continuous	QOI: Weight	Between 500 and 1000 [kg]		

Objectives

# ↑	Name	Source	Active	Direction	Actions
1	Weight [kg]	QOI: Weight	✓	Minimize	
2	Flight time [min]	QOI: Flight time	✗	Minimize	

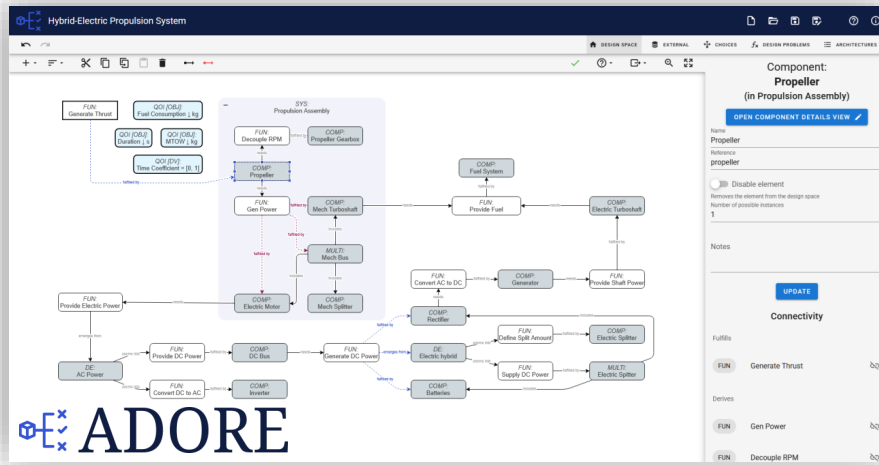
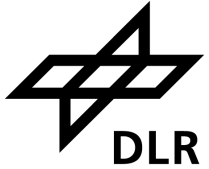
Constraints

# ↑	Name	Source	Active	Direction	Reference Value	Actions
1	Final load	QOI: Final load	✓	Greater than or equal to	0	

Generic Output Metrics

No generic output metrics

Thank you for your attention!



Contact:

jasper.bussemaker@dlr.de and/or luca.boggero@dlr.de

German Aerospace Center (DLR),
Institute of System Architecture in Aeronautics (SL),
Hamburg, Germany

- Bussemaker, J.H., 2025, July. [System Architecture Optimization: Function-Based Modeling, Optimization Algorithms, and Multidisciplinary Evaluation](#). Dissertation, Delft University of Technology.
- Bussemaker, J.H., García Sánchez, R., Fouda, M., Boggero, L. and Nagel, B., 2023, July. Function-Based Architecture Optimization: An Application to Hybrid-Electric Propulsion Systems. In INCOSE International Symposium. doi: [10.1002/iis2.13020](https://doi.org/10.1002/iis2.13020)
- Bussemaker, J.H. et al. System Architecture Optimization Strategies: Dealing with Expensive Hierarchical Problems. In: Journal of Global Optimization (2025). doi: [10.1007/s10898-024-01443-8](https://doi.org/10.1007/s10898-024-01443-8)
- Bussemaker, J.H., 2023. SBArchOpt: Surrogate-based architecture optimization. Journal of Open Source Software, 8(89), p.5564. doi: [10.21105/joss.05564](https://doi.org/10.21105/joss.05564)
- Garg, S. et al. Dynamic Formulation and Execution of MDAO Workflows for Architecture Optimization. In: AIAA AVIATION 2024 FORUM. Las Vegas, NV, USA, July 2024. doi: [10.2514/6.2024-4402](https://doi.org/10.2514/6.2024-4402)

ADORE: <https://adore.mbse-env.com/docs/>

SBArchOpt: <https://sbarchopt.readthedocs.io/>

ADSG-Core: <https://adsg-core.readthedocs.io/>