

System Architecture Optimization (SAO) with ADORE

ADORE Introduction | 2026-02-23

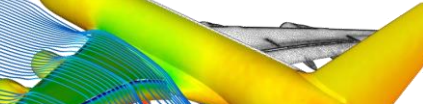
Jasper Bussemaker (jasper.bussemaker@dlr.de)

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



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



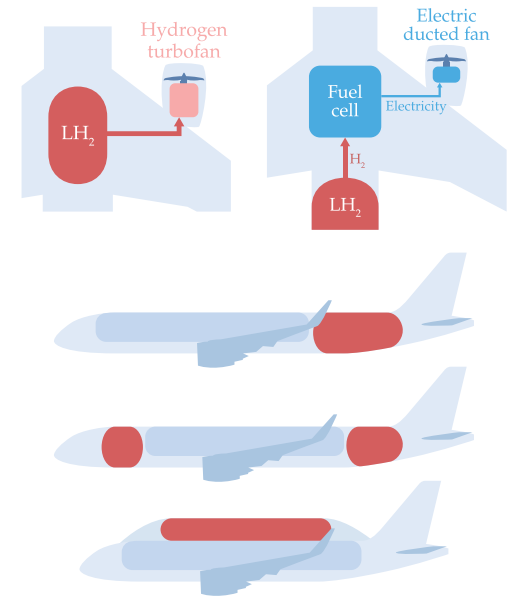
- 

-

-

-

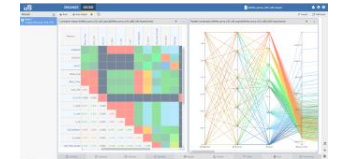
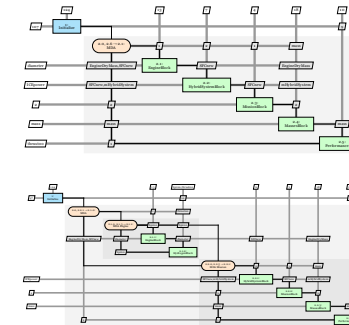
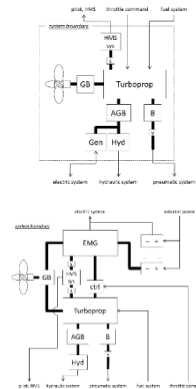
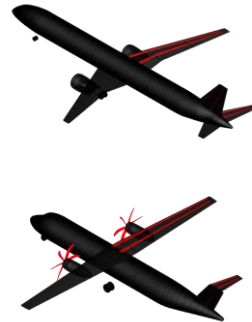
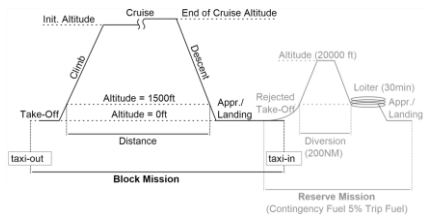
-



From Manual to Automatic Architecting



Manual Architecting Process



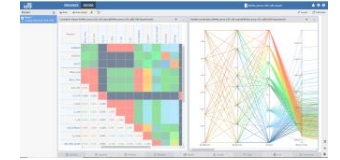
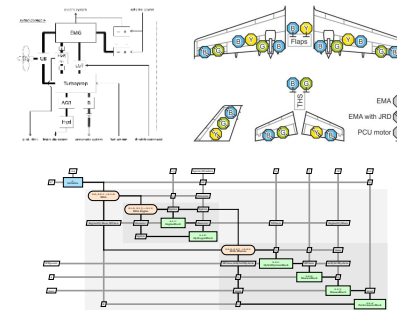
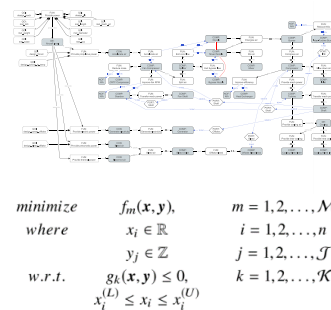
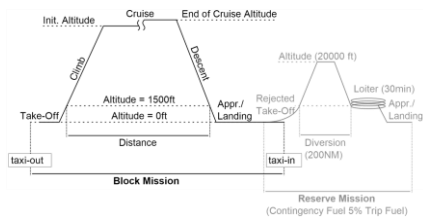
System requirements

Define *several* architectures

Design & analyze

Compare and decide

System Architecture Optimization (SAO) Process



System requirements

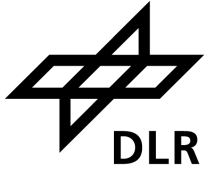
Define architecture optimization problem

Generate, design & analyze *many* architectures

Compare and decide

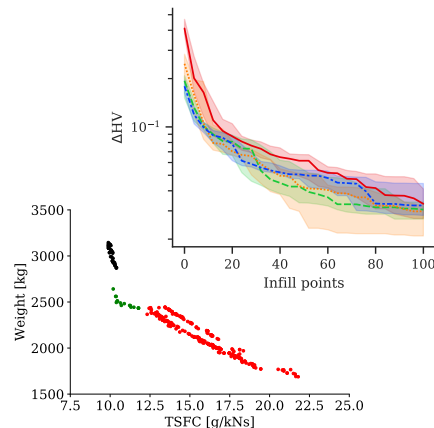


System Architecture Optimization with ADORE



Explore and optimize

- Automatically generate architectures
- Driven by machine learning algorithms

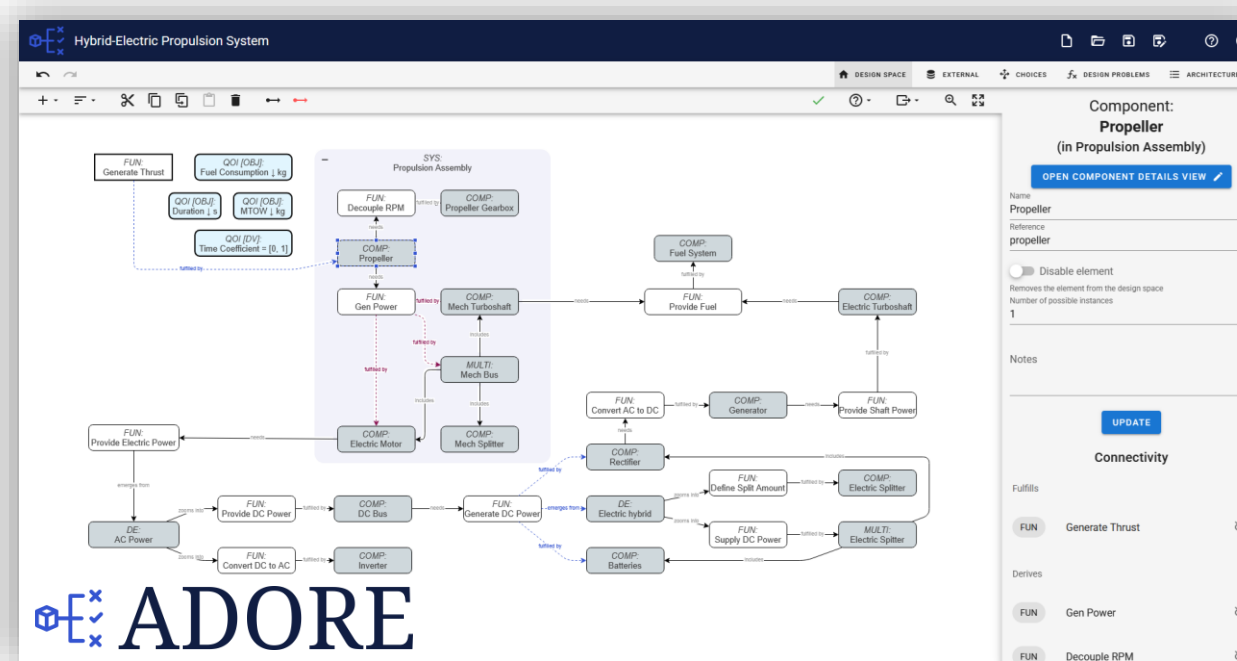


Model the architecting problem

- Define functions, components, connections
- Model architectural choices
- Set input parameters and metrics

Connect to your evaluation code

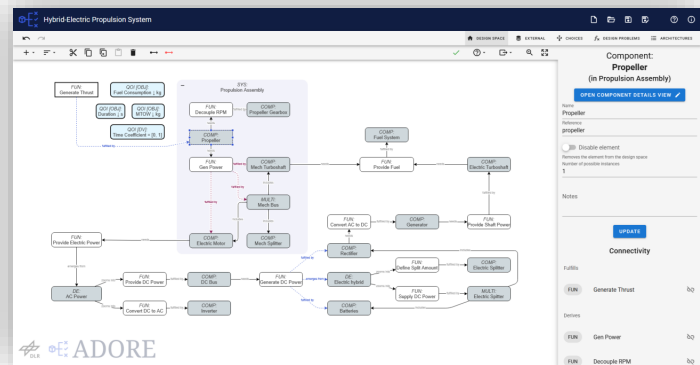
- Python- or file-based
- Local or remote execution



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

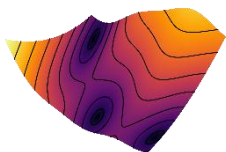


The System Architecture Optimization Loop



The ADORE Model
Defines architectural choices
and metrics.

formalizes the
optimization
problem



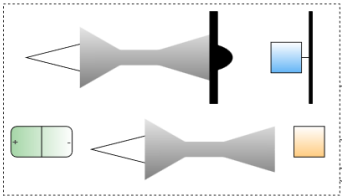
The Optimizer
Uses machine
learning algorithms.

generates design vector x

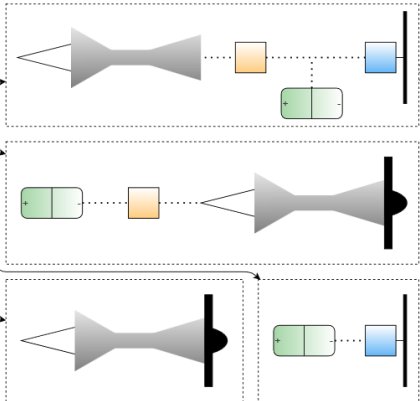
provides objective f and
constraint g values to optimizer

The Generator
Generates an
architecture instance
from x .

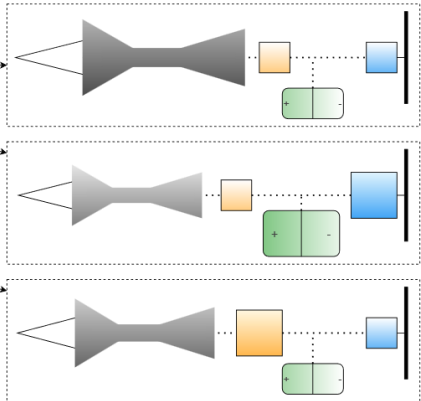
**Architecture
Design Space**



taking
Architectural choices
yields ...
Architectures



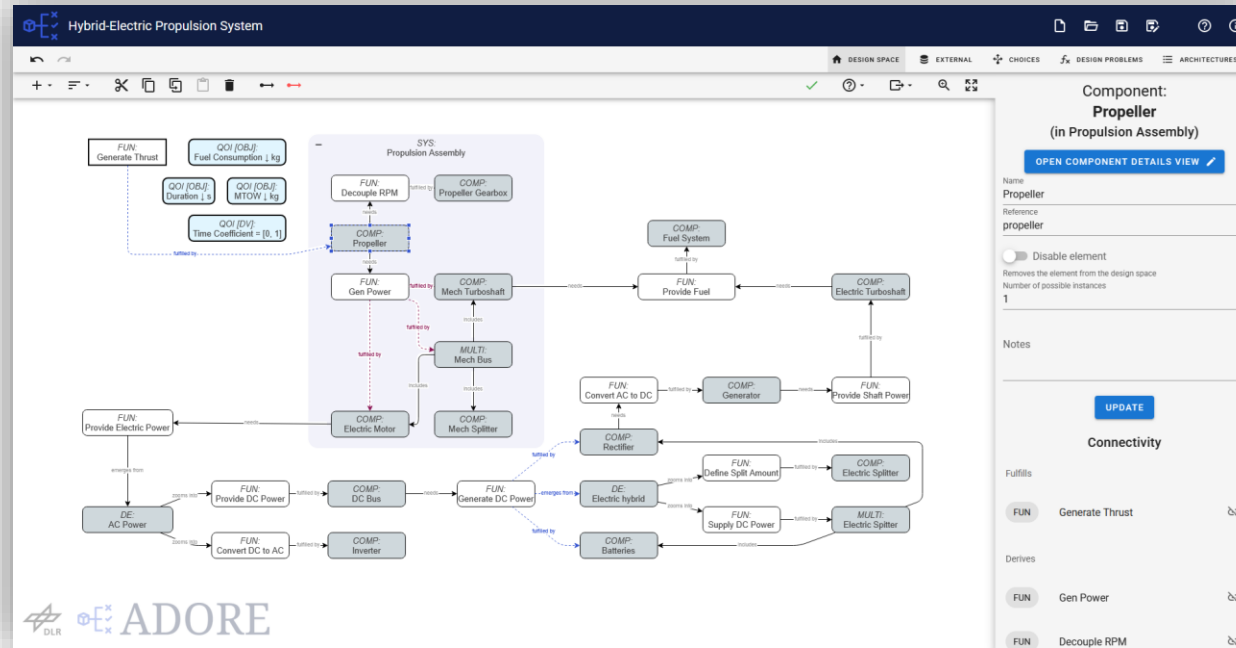
assigning values to
*Architecture-specific design
parameters* yields ...
Architecture Instances



each
architecture
instance is
evaluated by



The Evaluator
Executes problem-
specific multidisciplinary
analysis code.



 ADOPRE

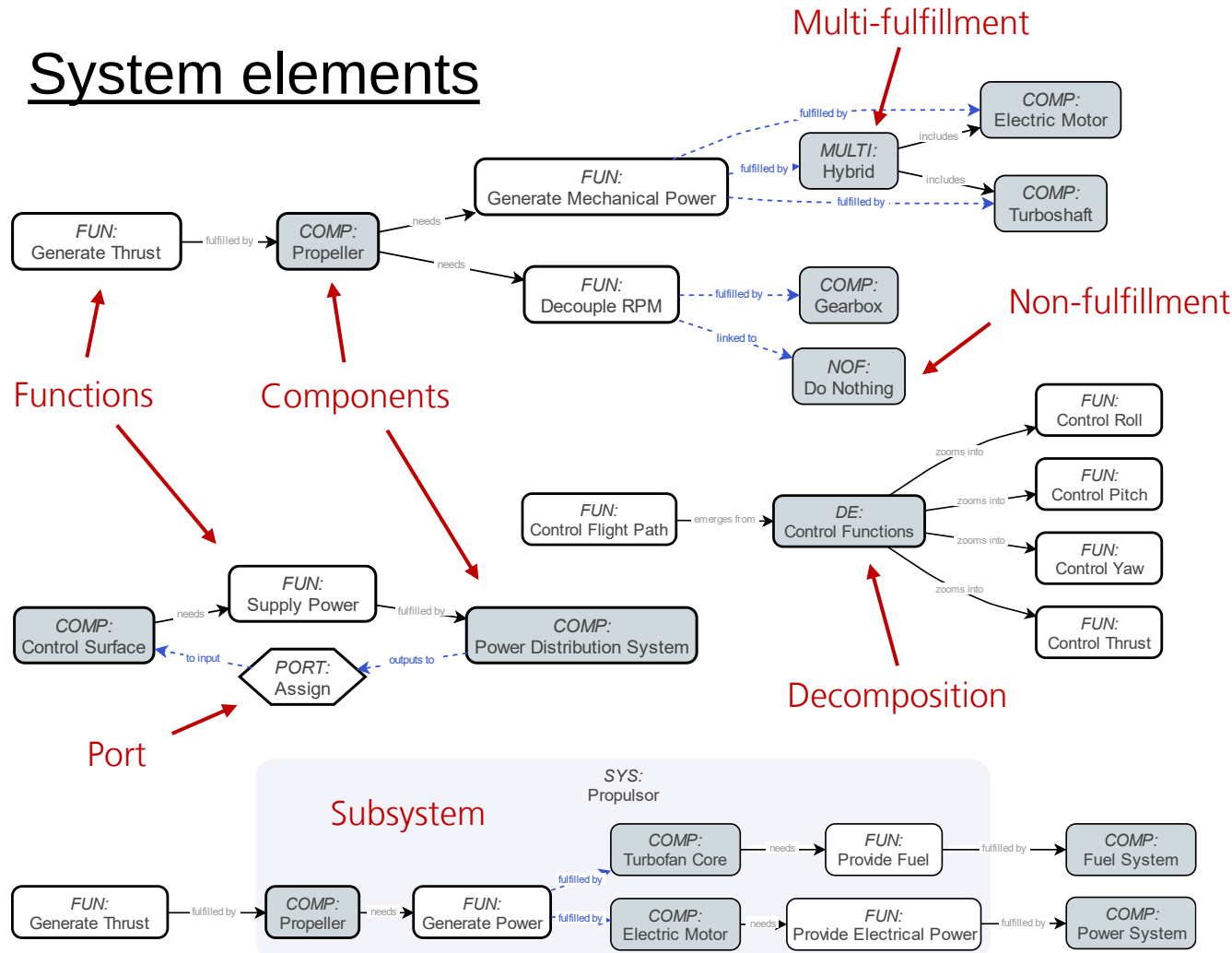
Modeling Architecture Design Spaces

Overview of Model Elements

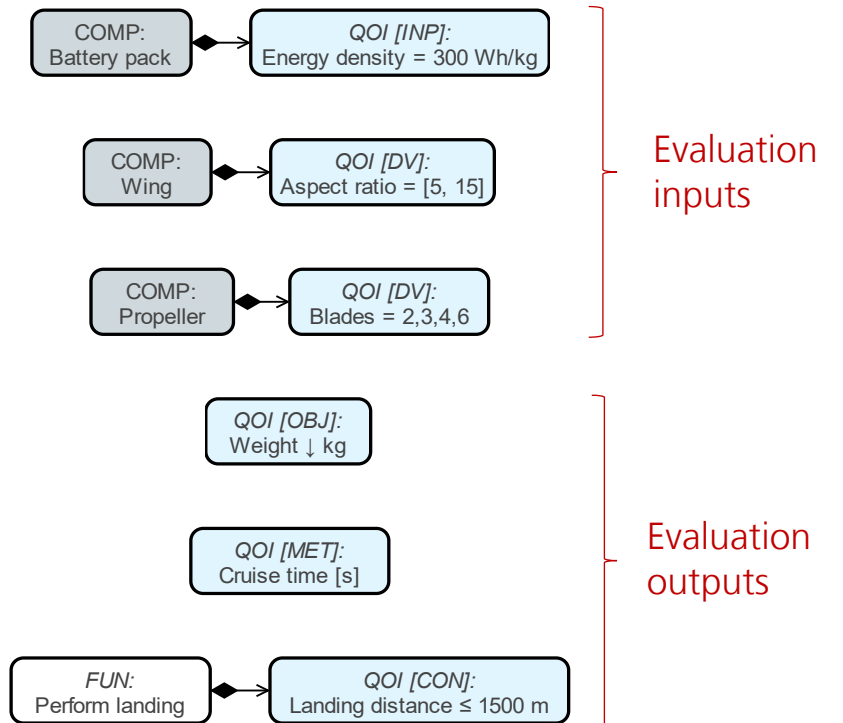
Non-exhaustive



System elements



Quantities of Interest (QOIs)



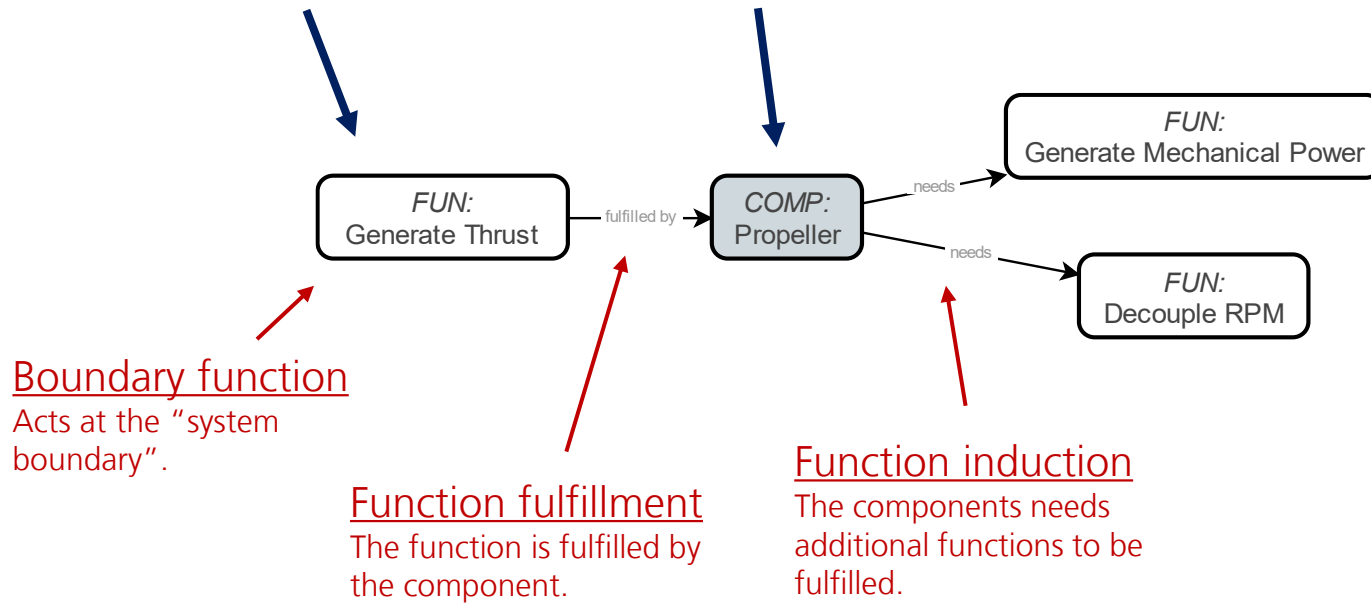
Function Allocation

Functions

Represent *what* the system does: "verb + operand".

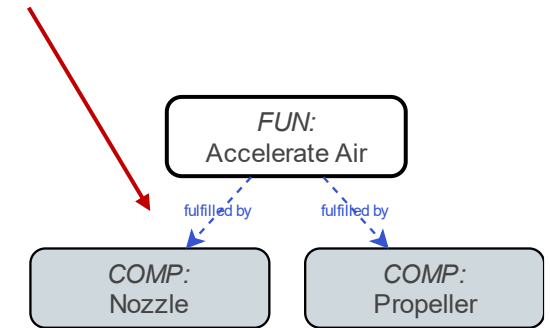
Components

Represent *how* the system fulfills its functions.



Architectural choice

Only *one* component can fulfill a function in an architecture instance.



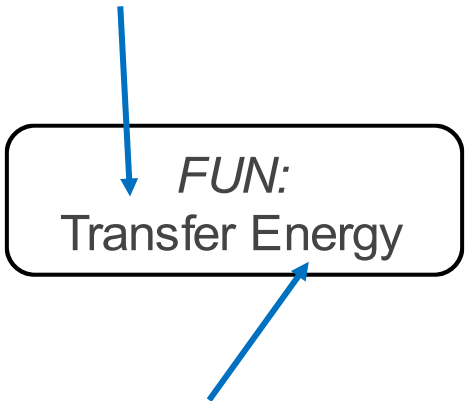
Formulating Functions



“What should the system do to bring benefits to its users?”

Process

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



FUN:
Control Aircraft on Ground

Condition (optional)
Operating condition where the function is active

Operand

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

Table 5. Functional basis reconciled function set

Class (Primary)	Secondary	Tertiary
Branch	Separate	Divide Extract Remove
Channel	Distribute Import Export Transfer	Transport Transmit
	Guide	Translate Rotate Allow DOF
Connect	Couple	Join Link
Control	Mix Actuate Regulate	
Magnitude	Change	Increase Decrease
		Increment Decrement Shape Condition
	Stop	Prevent Inhibit
Convert	Convert	
Provision	Store	Contain Collect
Signal	Supply Sense	Detect Measure
	Indicate	Track Display
Support	Process Stabilize Secure Position	

J. Hirtz, R.B. Stone, D.A. McAdams, S. Szykman, K.L. Wood, "A functional basis for engineering design", 2002, DOI: 10.1007/s00163-001-0008-3

← Process/verb Operand/object/
flow/noun
↓

Table 3. Functional basis reconciled flow set

Class (Primary)	Secondary	Tertiary
Material	Human Gas Liquid Solid	Object Particulate Composite
	Plasma Mixture	Gas-gas Liquid-liquid Solid-solid Solid-liquid Liquid-gas Solid-gas Solid-liquid-gas Colloidal
Signal	Status	Auditory Olfactory Tactile Taste Visual Analog Discrete
Energy	Control	
	Human Acoustic Biological Chemical Electrical Electromagnetic	Optical Solar
	Hydraulic Magnetic Mechanical	Rotational Translational
	Pneumatic Radioactive/Nuclear Thermal	

Function Allocation: Advanced

Functions

Represent *what* the system does: "verb + operand".

Components

Represent *how* the system fulfills its functions.

Architectural choice

Only *one* component can fulfill a function in an architecture instance.

Multi-fulfillments

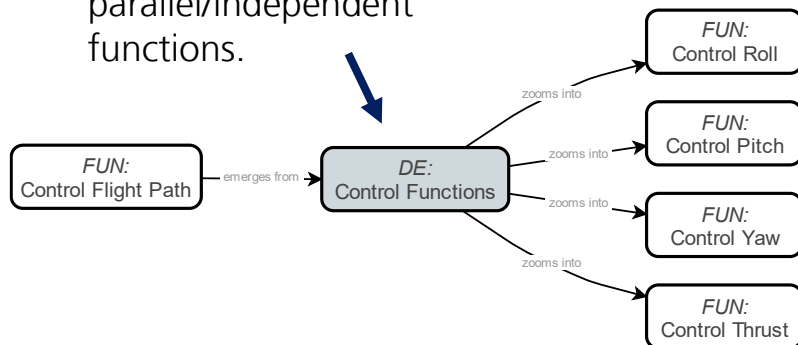
Represent the option for *multiple* components to fulfill a given function.

Non-fulfillments

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

Decompositions

Map one to multiple parallel/independent functions.

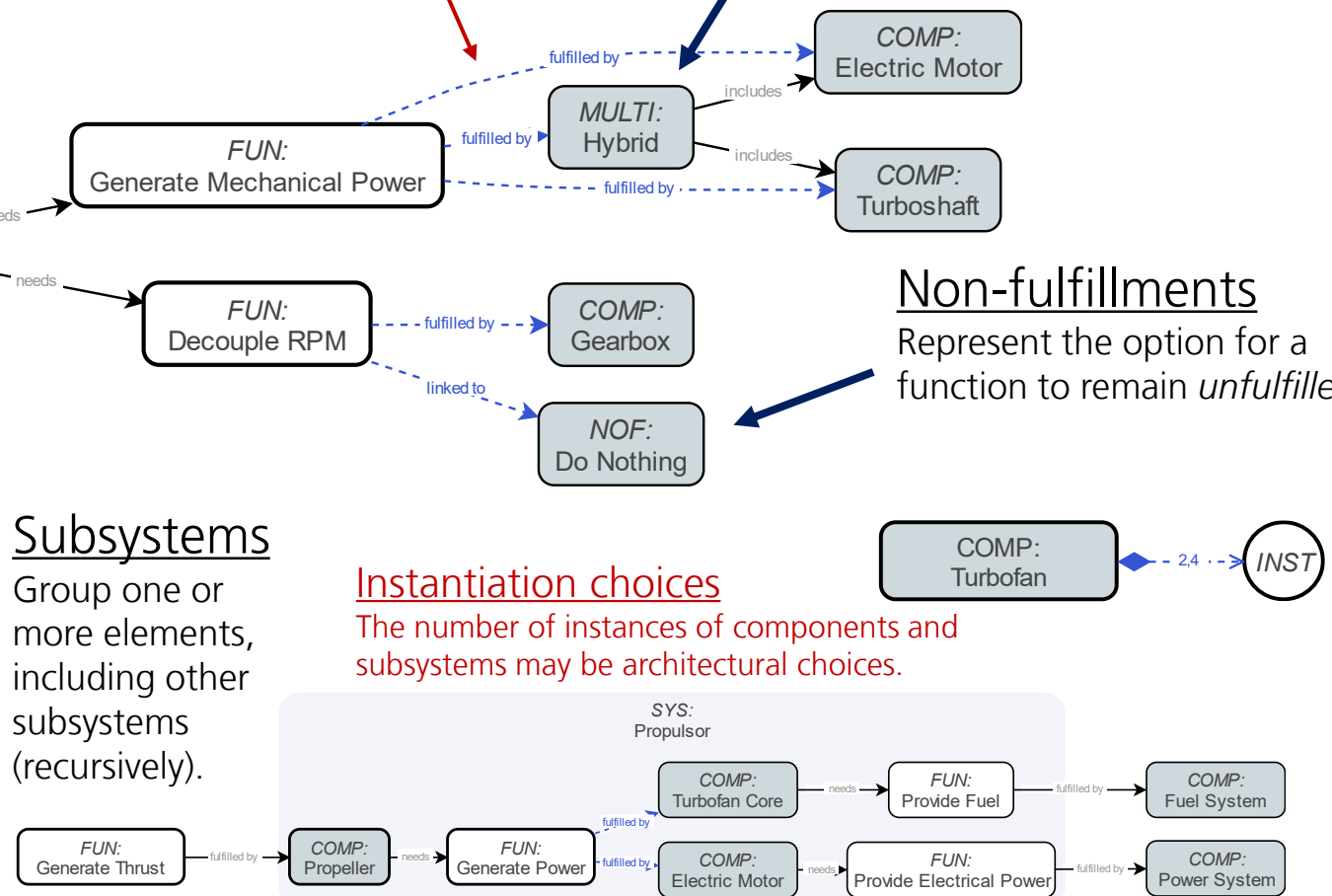


Subsystems

Group one or more elements, including other subsystems (recursively).

Instantiation choices

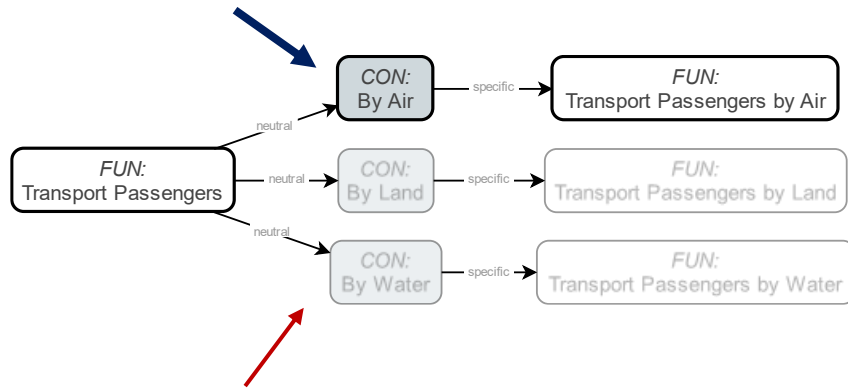
The number of instances of components and subsystems may be architectural choices.



Function Allocation: Advanced (2)

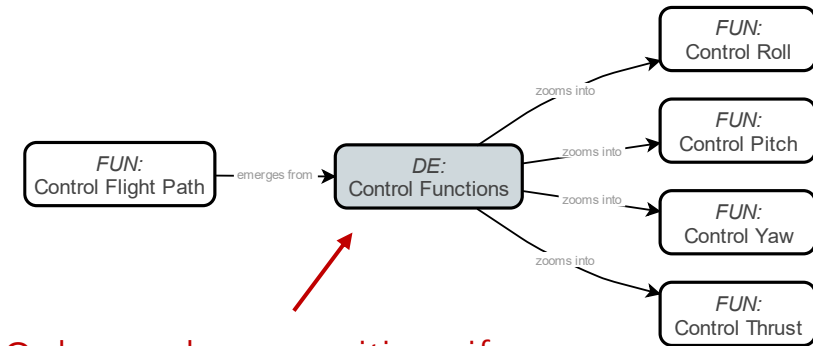
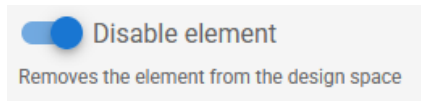
Concepts

Map a solution-neutral to a solution-specific function.



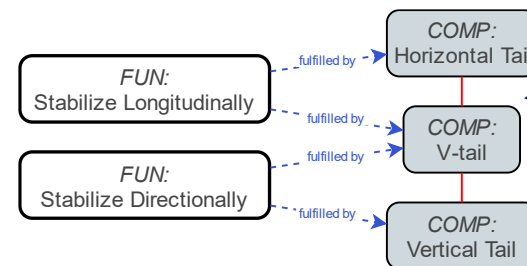
Disabled elements

Acknowledge technology alternatives without including them in the actual design problem.



Only use decompositions if:

1. Decomposed functions **act parallelly** or in a tightly-coupled loop (without a clear starting point).
2. The decomposed functions are **as solution-neutral** as the source function.
3. Downstream **cross-tree incompatibility constraints** are **not** needed.



Incompatibility constraints

Declare that two elements may never exist together in an architecture instance.

Quantities of Interest (QOIs)

QOIs

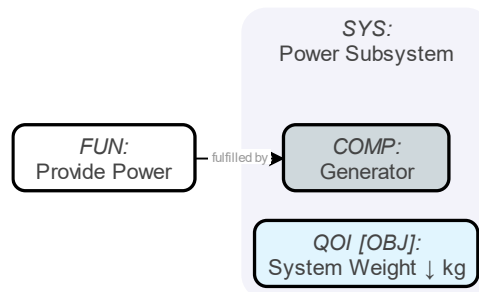
Represent quantifiable inputs or outputs to/from the architecture evaluation.

The type can be changed easily.

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

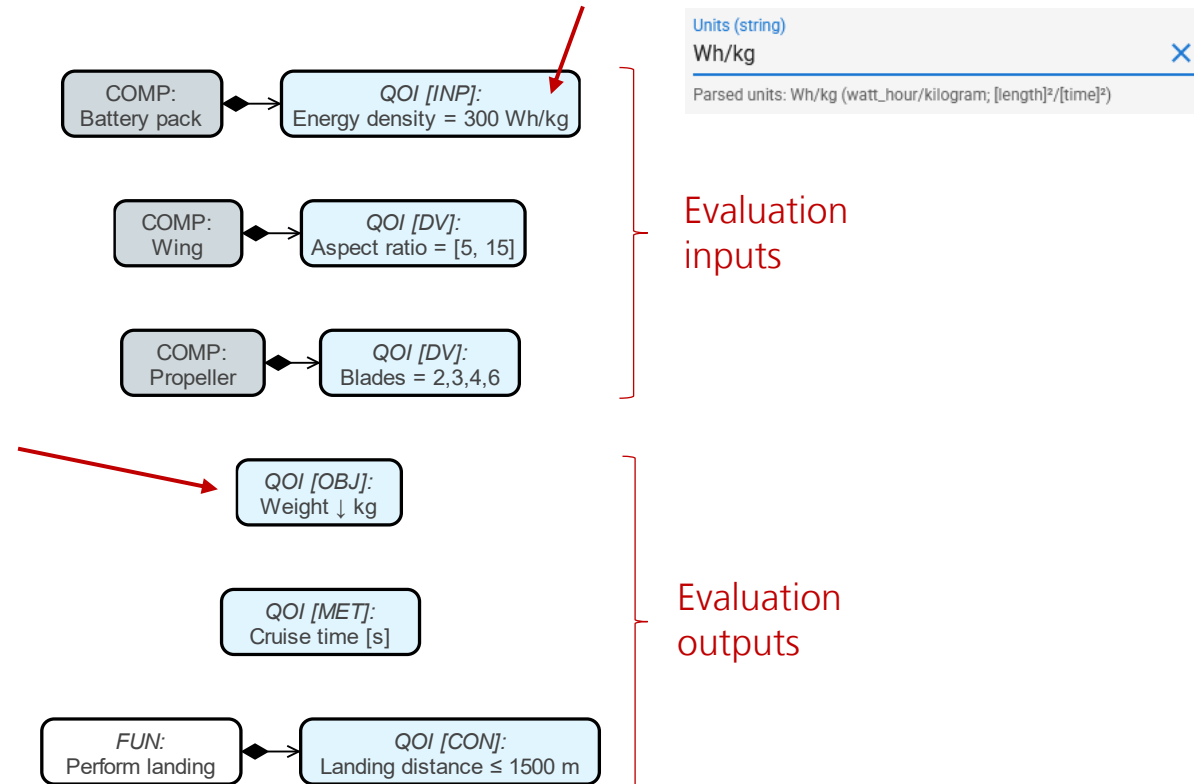
Objective QOIs

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



Units

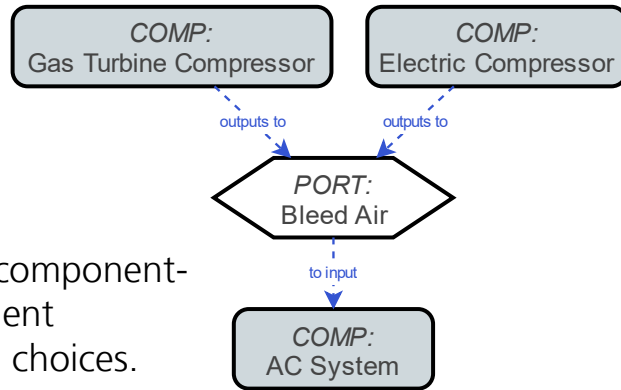
QOIs can have units defined.
Unit conversions are supported.



Component Connections through Ports

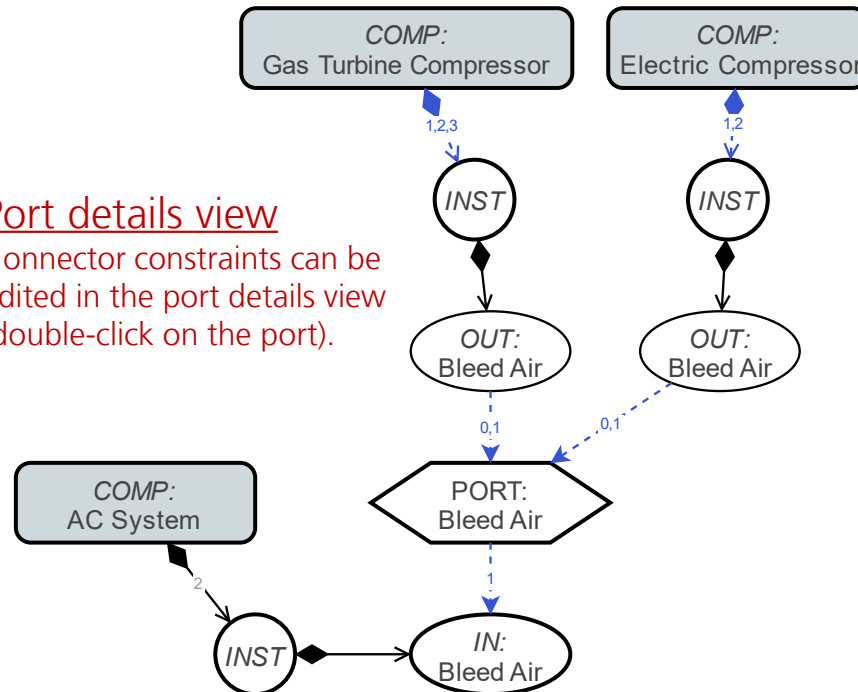
Ports

Represent component-to-component connection choices.



Port details view

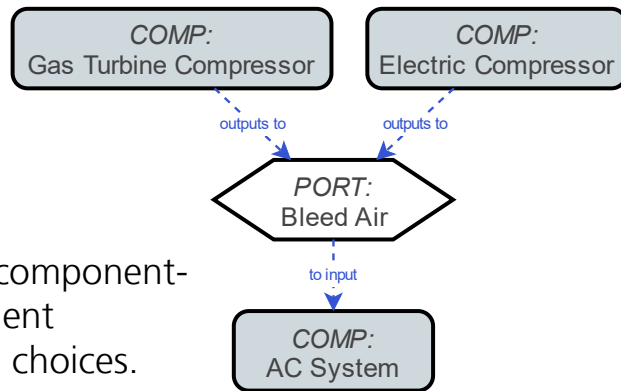
Connector constraints can be edited in the port details view (double-click on the port).



Component Connections through Ports

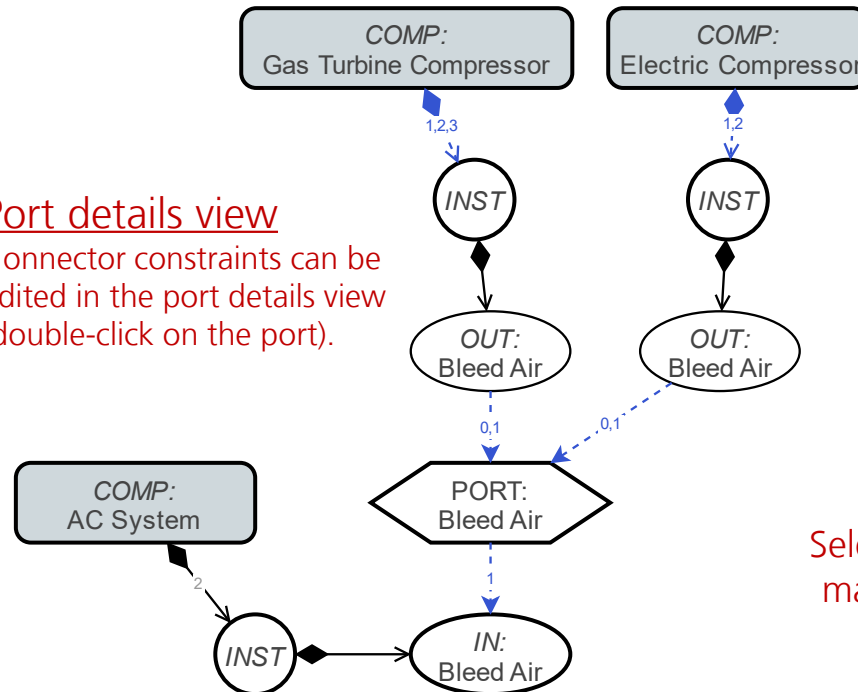
Ports

Represent component-to-component connection choices.



Port details view

Connector constraints can be edited in the port details view (double-click on the port).



Select connection order aggregation level (see next slide)

Set number of allowed connections for each connector

Select if multiple connections may be established between the same connector pairs

Port connector:
Bleed Air

☐ Disable element
Removes the element from the design space

Connection aggregation determines whether different connection orders (e.g. first connect to port instance 1, then to port instance 2) are treated as different architectures or not. Refer to the QRM for more details.

Connections are aggregated at the level of...
Port connectors

Allowed connections per connector: **0 or 1**

Specific number of connections allowed
0,1

Connections ranging from **0** to **no upper limit**

☐ Repeated connections allowed
Repeated connections between the same connectors allowed

☒ Self-connect allowed
Connections between ports owned by the same component allowed

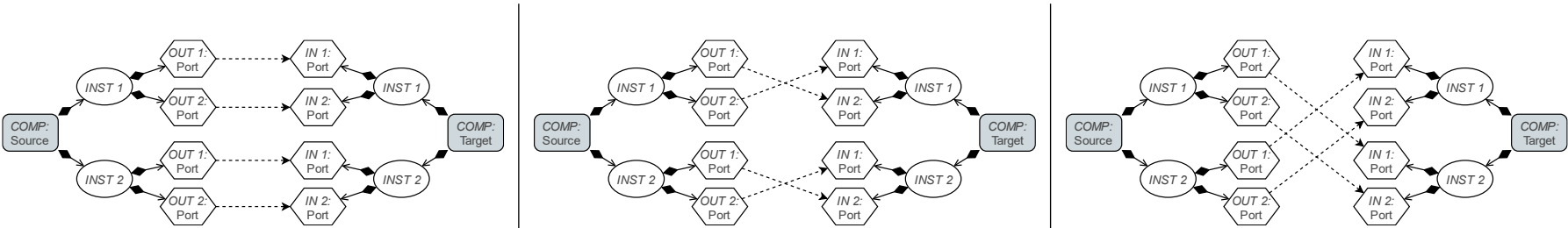
Port Connection Order Aggregation



Determines at which level different connection orders are regarded as different architecture instances.

2 components, with 2 instances each, and 2 port instances* per component instance

Connections are aggregated at the level of...



Port connectors

Architecture instance 1

Architecture instance 2

Architecture instance 3

Component instances*

Architecture instance 1

Architecture instance 2

Components

Architecture instance 1

* Enable advanced mode!



Advanced Mode

Advanced mode allows you to edit component "attributes", and change the number of port connector instances.

Advanced mode



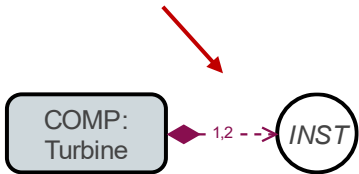
Choice Constraints



Choice constraints

Restrict the available options for a choice based on one or more other choices.

Constrained choices are shown in dark red



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)

Architecture Choices

	# ↑	Operation	Subject	Component Instance	System Instance	Options	Constraints
☐	1	Fulfill function	Generate DC Power			Electric hybrid, Batteries, Rectifier	↔
☐	2	Fulfill function	Gen Power		1	Electric Motor, Mech Bus, Mech Turboshaft	^ (5, Unordered Combinations ①)
☐	3	Fulfill function	Gen Power		2	Electric Motor, Mech Bus, Mech Turboshaft	↔
☐	4	Fulfill function	Gen Power		3	Electric Motor, Mech Bus, Mech Turboshaft	↔
☐	5	Fulfill function	Gen Power		4	Electric Motor, Mech Bus, Mech Turboshaft	↔
☐	6	Fulfill function	Gen Power		5	Electric Motor, Mech Bus, Mech Turboshaft	↔
☐	7	Instantiate system	Propulsion Assembly				
☐	11	Continuous design variable	COMP: a -> elec_DoH_climb				
☐	12	Continuous design variable	COMP: a -> elec_DoH_cruise				
☐	13	Continuous design variable	COMP: a -> elec_DoH_descend				
☐	14	Continuous design variable	COMP: a -> elec_DoH_loiter				
☐	15	Continuous design variable	COMP: a -> elec_DoH_reserve_climb				

Constraint Type

Linked Values

Permutations

Unordered Combinations

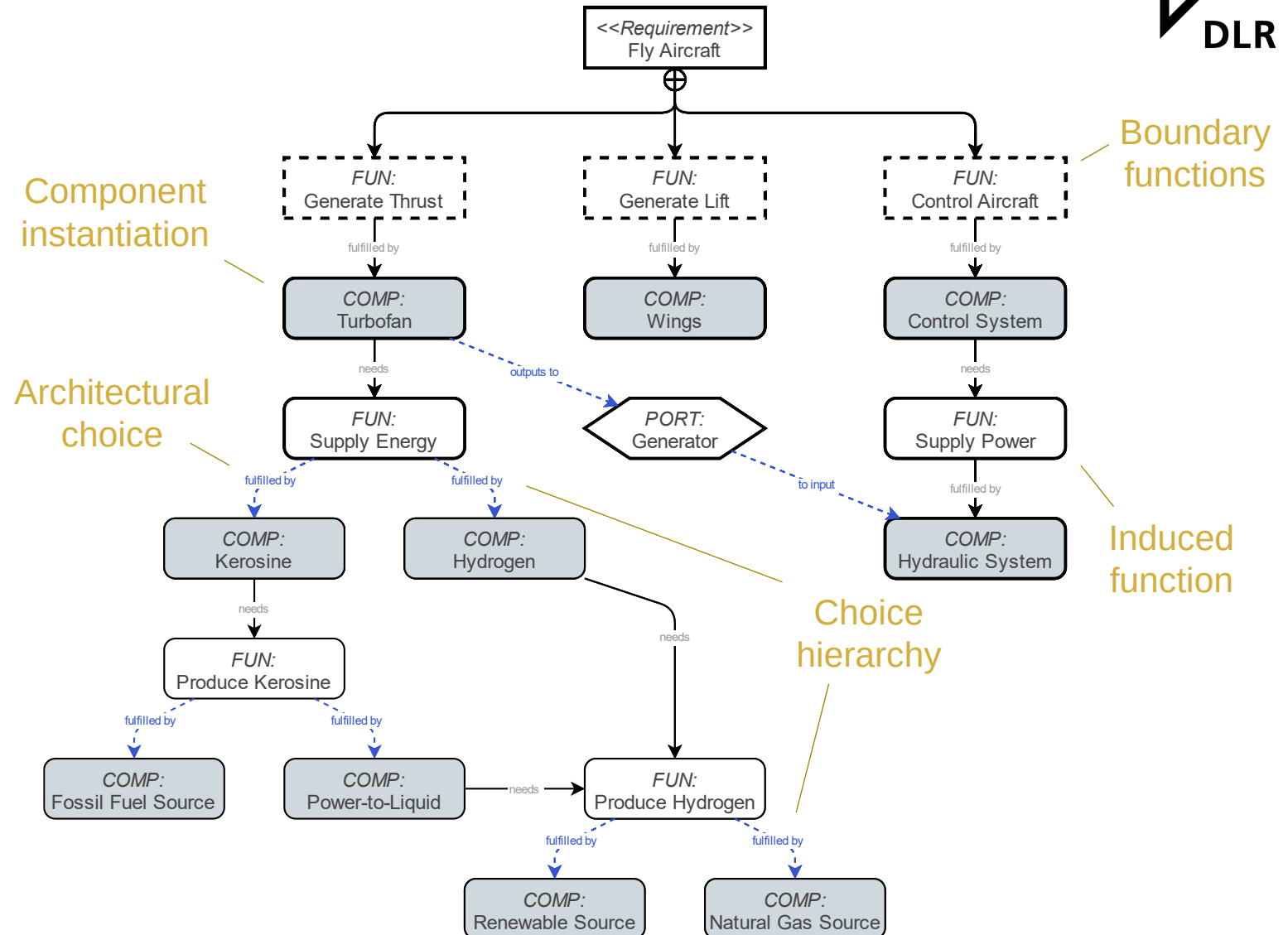
Unordered Non-replacing Combinations

CANCEL

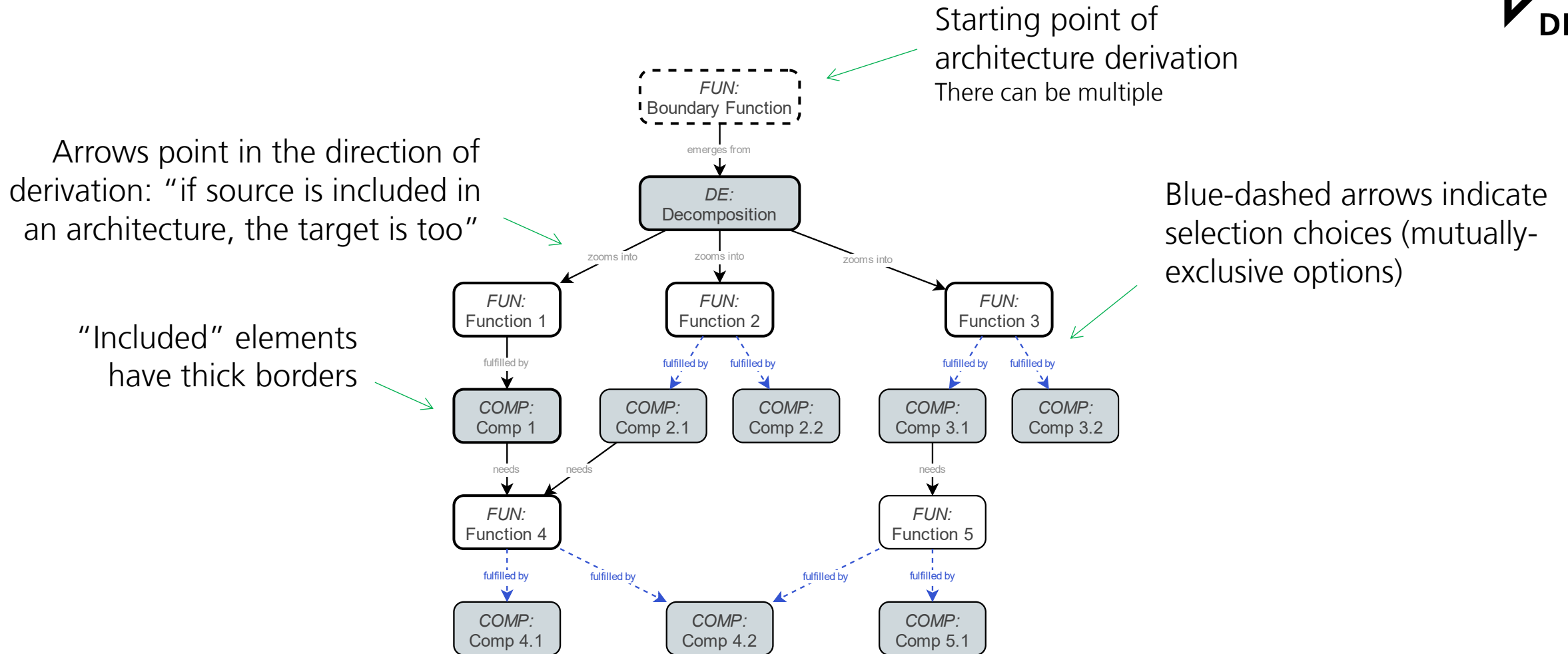
APPLY CHOICE CONSTRAINT

Modeling Process Overview

- Start from boundary functions
 - Traced from requirements
- Assign** functions to components
- Induce** additional functions from components
 - Repeat...
- Model **architectural choices**
- Possible to represent complex choice scenarios

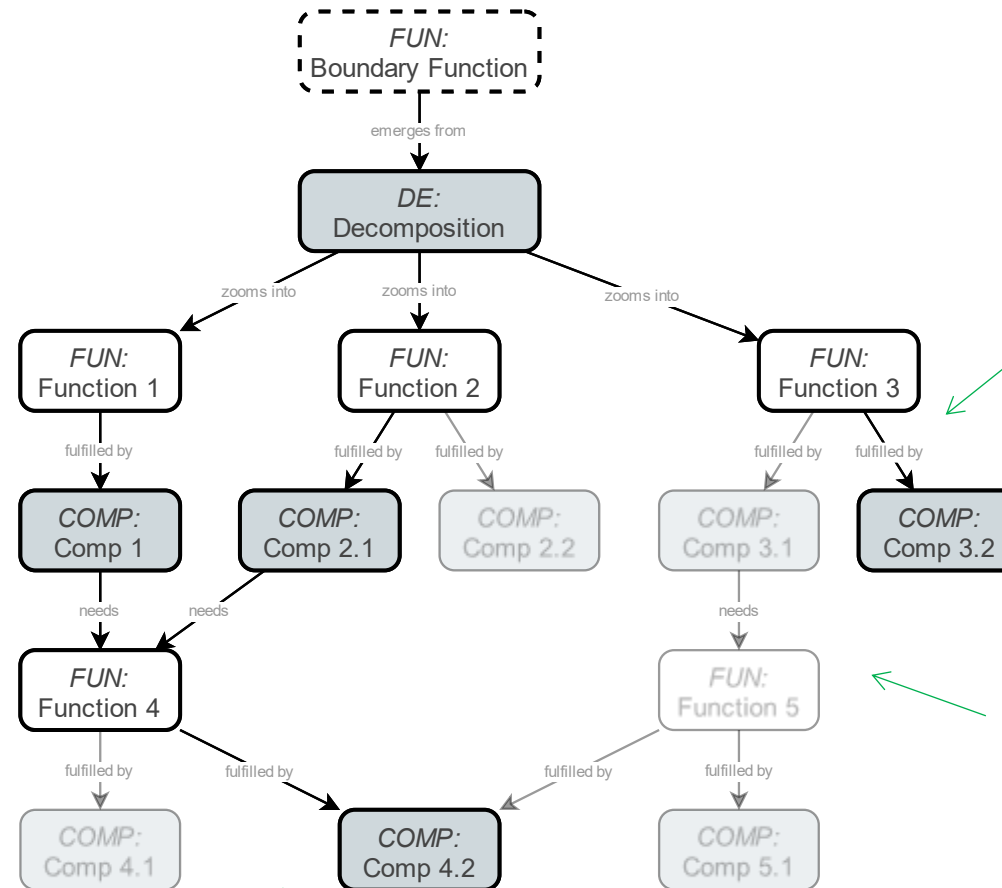


Generating an Architecture Instance



Generating an Architecture Instance

This represents an architecture *instance*, because there are no more choices to take

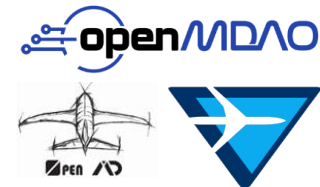


Select one of the options for each selection choice
Manually (in GUI) or from a design vector

Not-selected options and their derived nodes (and choices) are marked for removal

One incoming derivation edge is enough to select an element

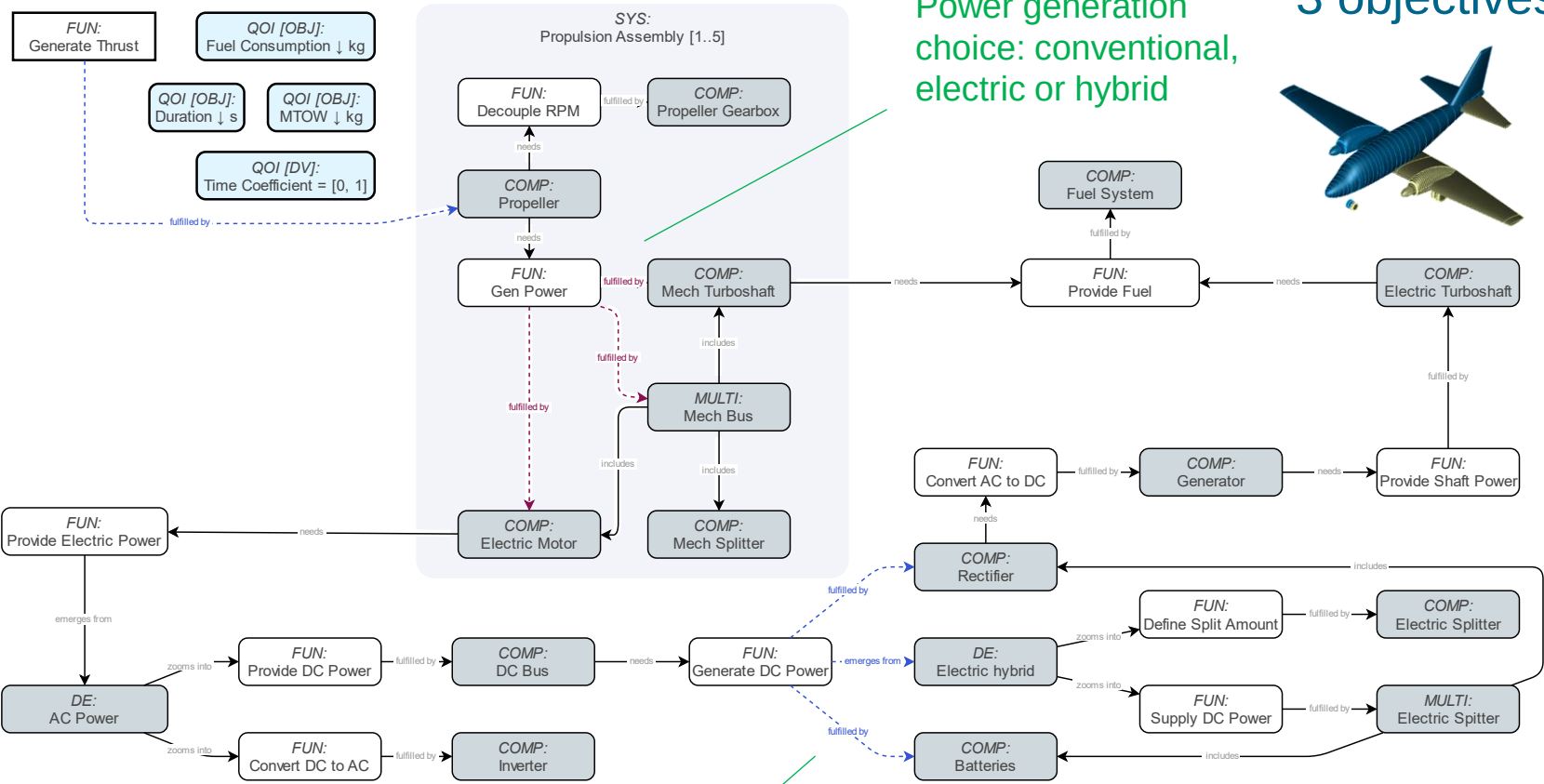
Showcase: Hybrid-Electric Propulsion System



Boundary function and system-level performance metrics

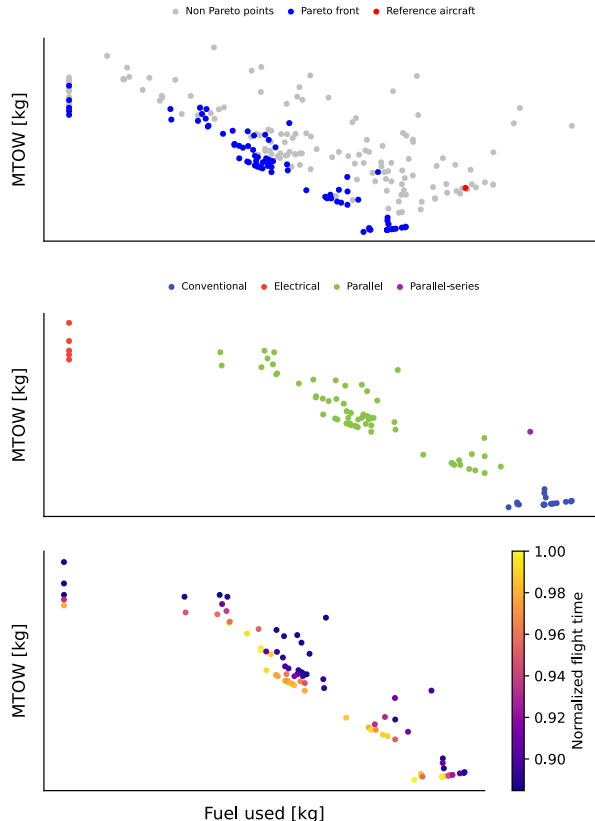
Propulsion assembly:
between 1 and 5 per wing

310 architectures
26 design variables
3 objectives



Power generation
choice: conventional,
electric or hybrid

Electricity generation choice:
generator, batteries or hybrid

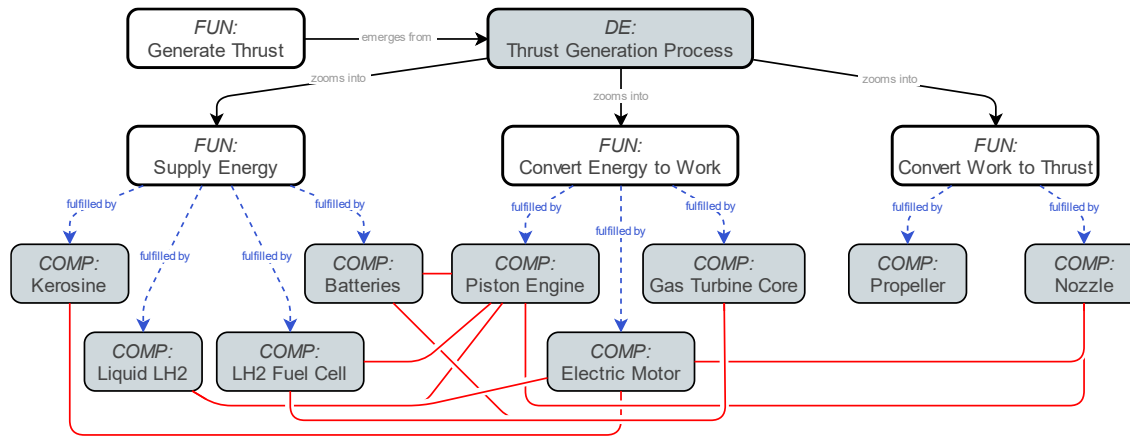


Comparison of Modeling Approaches

Tree-structured

Morphological matrix / Feature model

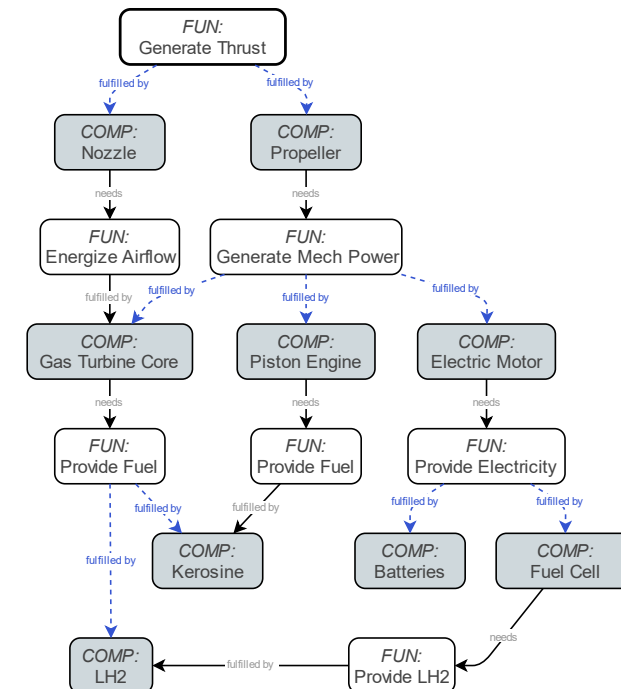
- Top-down using function decomposition
- One layer of function-means modeling
- Many incompatibilities / cross-tree constraints



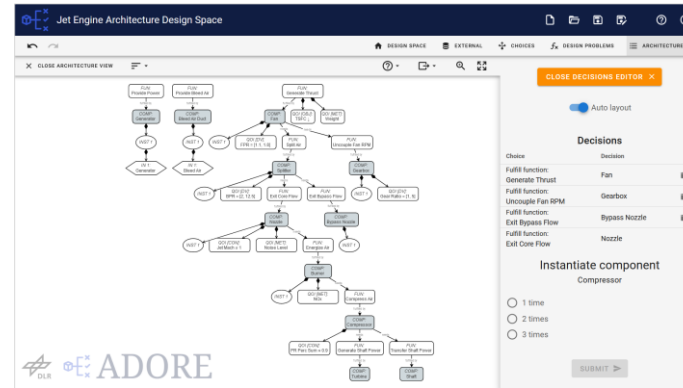
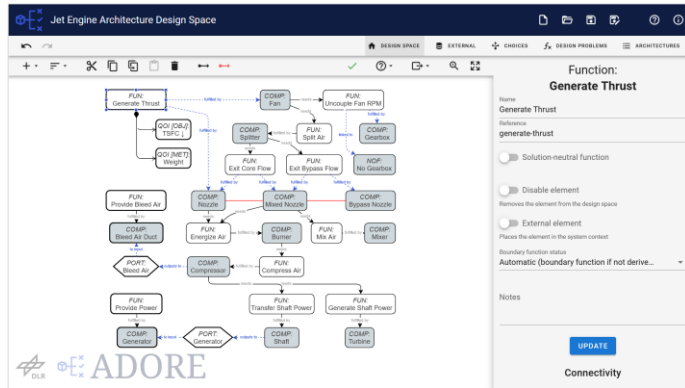
Graph-structured

ADORE / ADSG

- Start from main value-providing function(s)
- Derive by zigzagging (function $\leftrightarrow$ means)



Architecture Design Space Graph (ADSG) Layers



ADORE

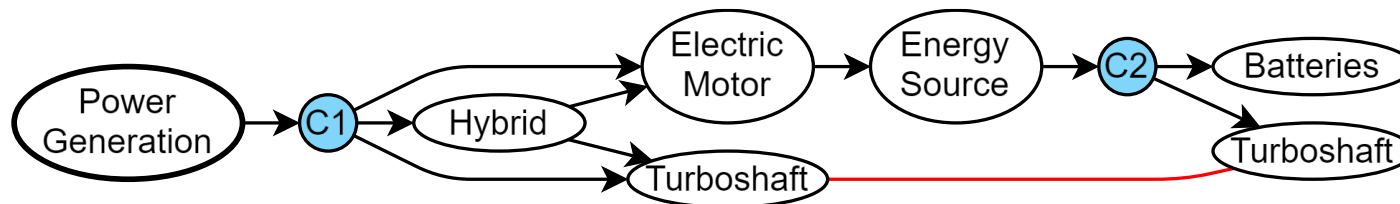
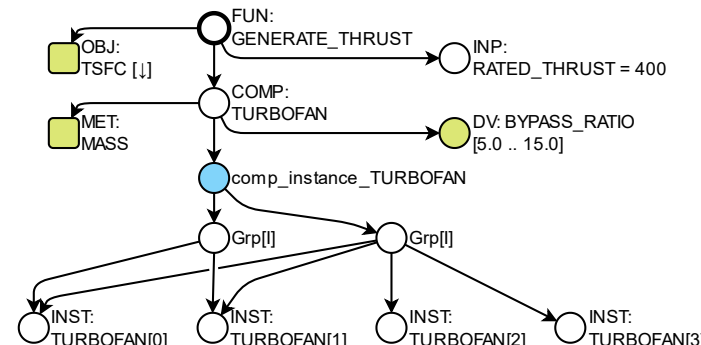
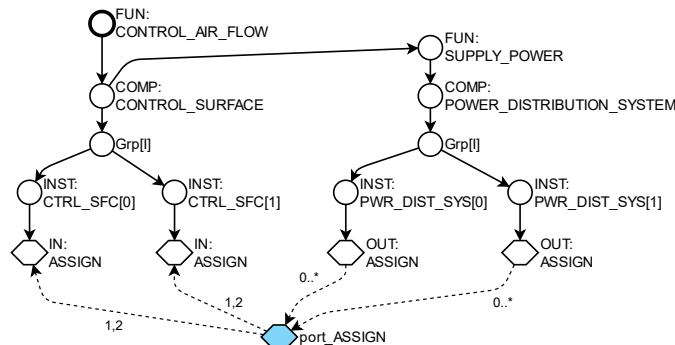
- Graphical user interface
- Evaluation and optimization interfaces

Architecture DSG (ADSG)

- System architecture terminology layer
- Functions, components, ports, etc.

Design Space Graph (DSG)*

- Hierarchical selection and connection choice modeling
- Encoding and decoding algorithms



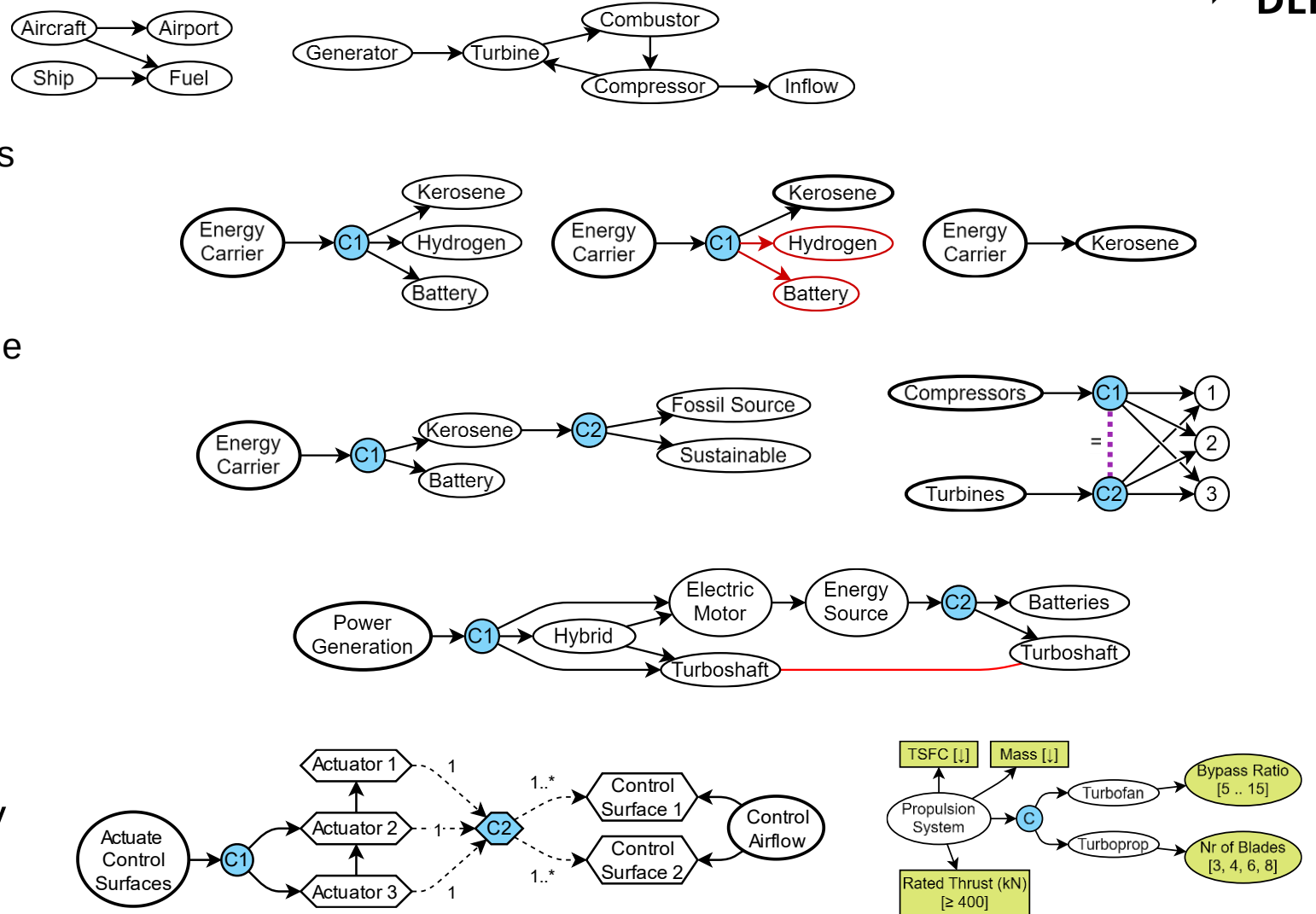
* Open-source at:

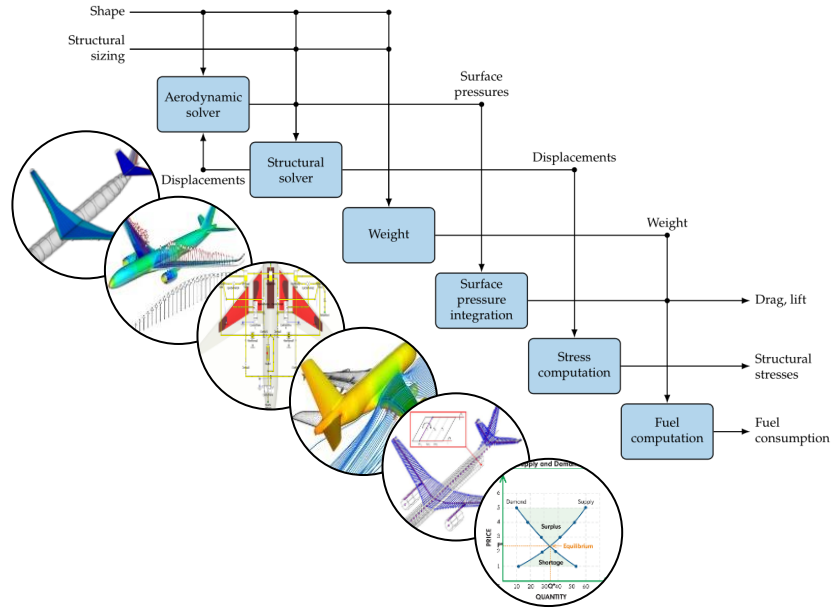
<https://adsg-core.readthedocs.io/>



Design Space Graph (DSG)

- Directed graph
 - Nodes represent system elements (to be selected)
 - Edges: derivation, connection
- Hierarchical choice modeling
 - Selection choices: select one node from a set of options to be included in an architecture instance
 - Choice constraints, incompatibilities
 - Connection choices: select connections between sets of source and target nodes
- Custom design variables and output metrics
- Choices nodes are automatically encoded as design variables

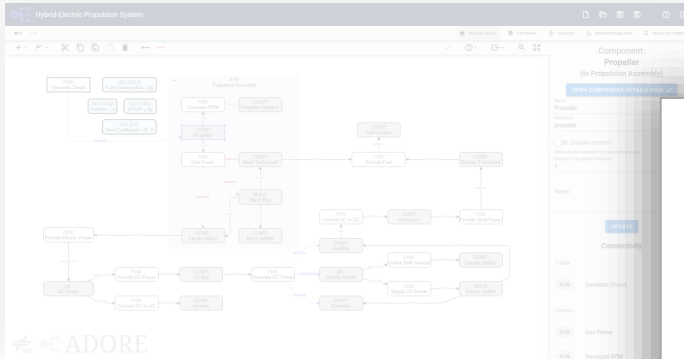
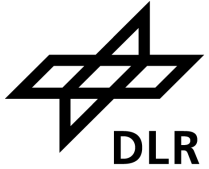




 **ADORE**

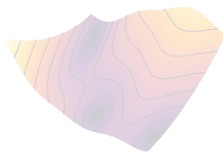
Architecture Evaluation

The Architecture Evaluator



The ADORE Model
Defines architectural choices
and metrics.

formalizes the
optimization
problem



The Optimizer
Uses machine
learning algorithms.

generates design vector x

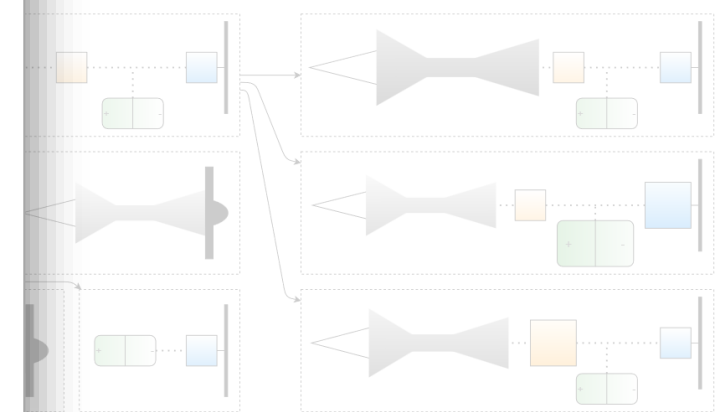
provides objective f and
constraint g values to optimizer

The Evaluator

- Is problem-specific.
- Should calculate **all** relevant metrics for **all** architectures with **similar** fidelity.
- Should be **sensitive** to architectural choices.
- Should be **automatically executable** (i.e. without user interaction).

taking
Architectural choices
yields ...
Architectures

assigning values to
Architecture-specific design
parameters yields ...
Architecture Instances



each
architecture
instance is
evaluated by



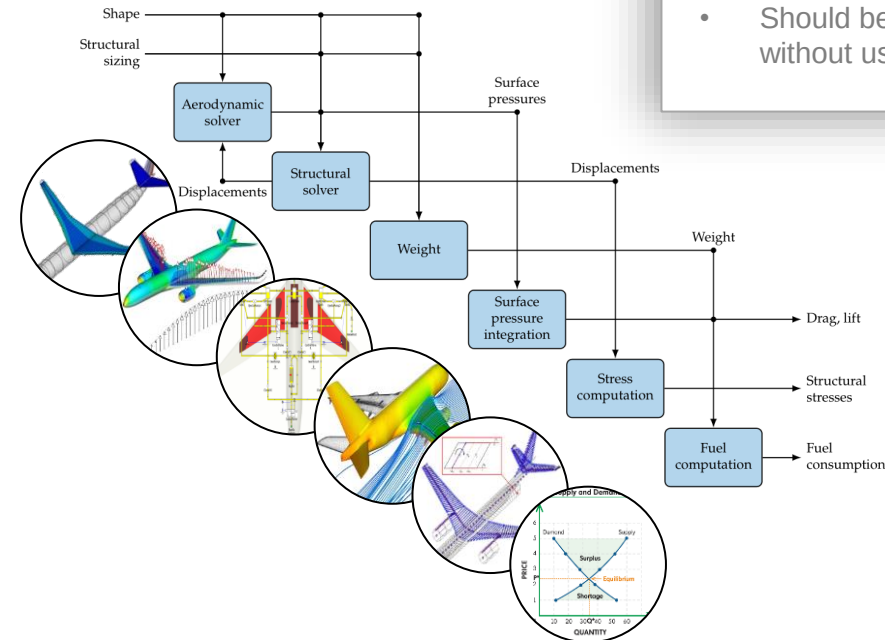
The Evaluator
Executes problem-
specific multidisciplinary
analysis code.

Developing the Evaluator

- Multiple engineering disciplines will be involved with different design goals
- Multidisciplinary Design Analysis and Optimization (MDAO)
 - Integrates multiple engineering tools in coupled workflows
 - Ensures the design is feasible and consistent across engineering disciplines
- Collaborative MDAO
 - MDAO with a managed central data format so tools “speak the same language”
 - Distributed execution of tools
 - For large-scale projects involving many partners

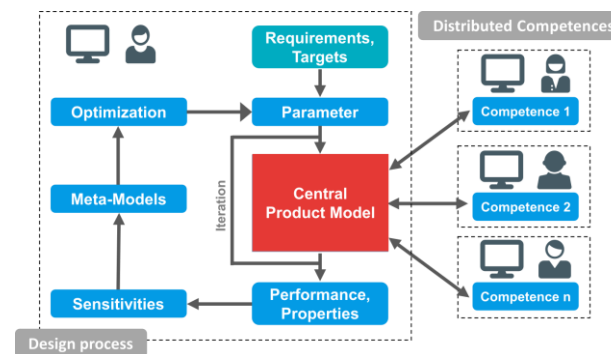
The Evaluator

- Is problem-specific.
- Should calculate **all** relevant metrics for **all** architectures with **similar** fidelity.
- Should be **sensitive** to architectural choices.
- Should be **automatically executable** (i.e. without user interaction).



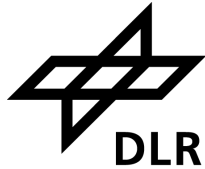
Introduction to MDAO:

Chapter 13 of
<https://mdobook.github.io/>



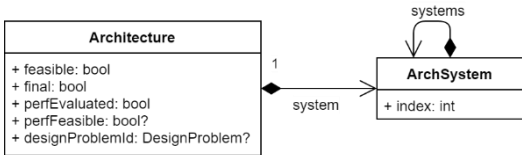
 **ADORE**
<https://adore.mbse-env.com/>

ADORE Evaluation Interfaces



Supports no-code, external optimization
(e.g. in MDAO integration platforms).



	Python-based	File-based
Direct access to ADORE data model	Evaluation API • Architecture instance as input • Need to parse data model yourself 	Architecture Serialization • Serialization of architecture to file • Placeholders for outputs 
Rule-based translation	Class Factory Evaluator • Define rules to instantiate classes from architecture elements <pre>class WingClassFactoryEvaluator(ClassFactoryApiEvaluator): @staticmethod def get_class_factories() -> List[ClassFactory]: return [ClassFactory(el=ExternalComponentDef(name='Wing', n_inst=1)), cls=Wing, props={ 'area': ExternalQIDef(name='Wing Area', qoi_type=QOIType.DESIGN_VAR, bounds=(40., 60.)), 'cl': ExternalQIDef(name='Lift Coefficient', qoi_type=QOIType.DESIGN_VAR, bounds=(0., .5)), 'ias': ExternalQIDef(name='Indicated Airspeed', qoi_type=QOIType.INPUT_PARAM, value=100.), },),]</pre>	Node Factory Evaluator • Define rules to create XML nodes from architecture elements • Dedicated CPACS interface available 

Python-based Evaluation



For more details and the API reference, refer to the documentation.



Problem-specific evaluator

Implement the `_evaluate` function.

Instantiate

Instantiate the evaluator with your ADORE model.

Generate

Generate and evaluate architectures. Connect to optimization algorithms.

```
import adore as ad

class MyPythonEvaluator(ad.GraphApiEvaluator):
    """Problem-specific Python-based evaluator."""

    def _evaluate(self, architecture: ad.Architecture,
                  arch_qois: List[ad.ArchQOI], **kwargs) \
        -> Dict[ad.ArchQOI, Union[float, ad.Quantity]]:
        """
        The function that evaluates an architecture instance.
        Returns the value of each requested architecture QOI,
        either as a float (or NaN) or a Quantity (value + units).
        Use `self.quantity` to create a Quantity object.
        """

    if __name__ == '__main__':
        # Instantiate your evaluator from your ADORE model
        evaluator = MyPythonEvaluator.from_file('model.adore')

        # Create random architectures and evaluate them
        for _ in range(10):
            architecture, design_vector, activeness_vector = \
                evaluator.get_architecture(
                    evaluator.get_random_design_vector())

            objectives, constraints = evaluator.evaluate(architecture)
            print(f'Objective values: {objectives}')
            print(f'Constraint values: {constraints}')

        # Save the ADORE file and a CSV export
        evaluator.to_file('updated_model.adore')
        evaluator.save_results_csv('results.csv')
```

Class Factory Evaluator (CFE)

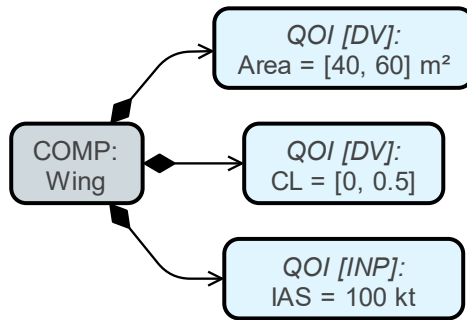
Use the CFE to help extract data from the **Architecture**, by instantiating custom classes from architecture elements.

See next slide.

Class Factory Evaluator (Python)



For more details and the API reference, refer to the documentation.



```
from typing import *
from dataclasses import dataclass
import adore as ad
```

```
@dataclass
class Wing:
    area: float
    cl: float
    ias: float
```

```
class MyClassFactoryEvaluator(ad.ClassFactoryApiEvaluator):
    """Problem-specific Python-based Class Factory Evaluator."""
```

```
def get_class_factories(self) -> List[ad.ClassFactory]:
    return [ad.ClassFactory(
        el=ad.ExternalComponentDef(name='Wing', auto=True),
        cls=Wing,
        props={
            'area': ad.ExternalQOIDef(
                name='Area', auto=True, units='m^2'),
            'cl': ad.ExternalQOIDef(
                name='Lift coefficient', auto_match_pattern='CL'),
            'ias': ad.ExternalQOIDef(
                name='IAS', auto=True, units='kts'),
        }
    )]
```

Class factories

Rules for linking to ADORE elements and instantiating objects for each occurrence in an architecture instance.

```
classes = {list: 1} [Wing(area=50.0, cl=0.25, ias=100.0)]
0 = {Wing} Wing(area=50.0, cl=0.25, ias=100.0)
  area = {float} 50.0
  cl = {float} 0.25
  ias = {float} 100.0
  __len__ = {int} 1
```

```
def _evaluate(self, architecture, arch_qois, **_):
    # Get Wing objects from the architecture
    wings = self.instantiate(
        architecture, factories=self.get_factories_by_el_name('Wing'))
    ...
```

Instantiation

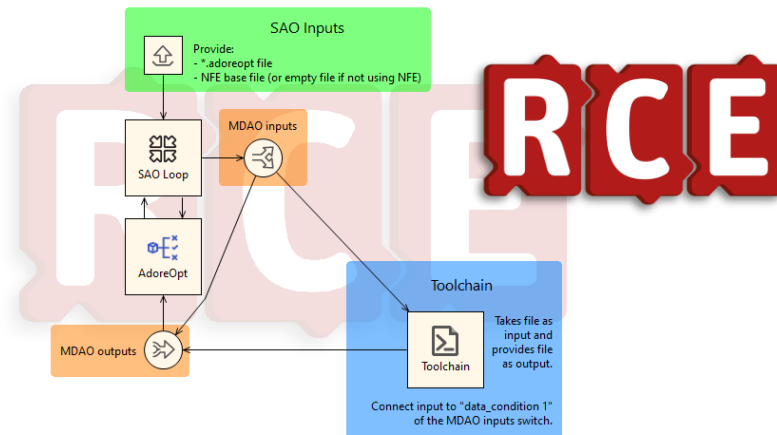
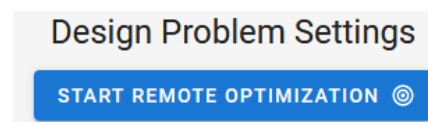
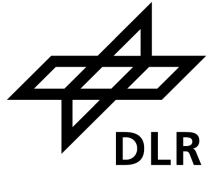
Instantiate objects in the `_evaluate` function.

File-based Evaluation

- Direct serialization of the architecture instance
- Rule-based construction of the evaluation input file
 - XML Node Factory Evaluator
 - CPACS
- Supports remote optimization



```
{
  "projectName": "Guidance, Navigation & Control",
  "architecture": {"id": 103...},
  "outputs": [
    {
      "archQoiId": 109,
      "name": "Mass",
      "ref": "mass",
      "type": "OBJECTIVE",
      "units": "kg",
      "value": null
    },
    {
      "archQoiId": 110,
      "name": "Failure Rate",
      "ref": "failure-rate",
      "type": "OBJECTIVE",
      "units": null,
      "value": null
    }
  ],
  "designSpace": [ 6 elements... ],
  "designProblem": {"id": 67...}
}
```



31



ADORE to CPACS Interface

CPACS

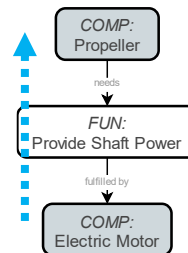
Common Parametric Aircraft Configuration Schema; <https://cpacs.de/>

XML format for modeling aircraft. Includes onboard system architecture definitions.

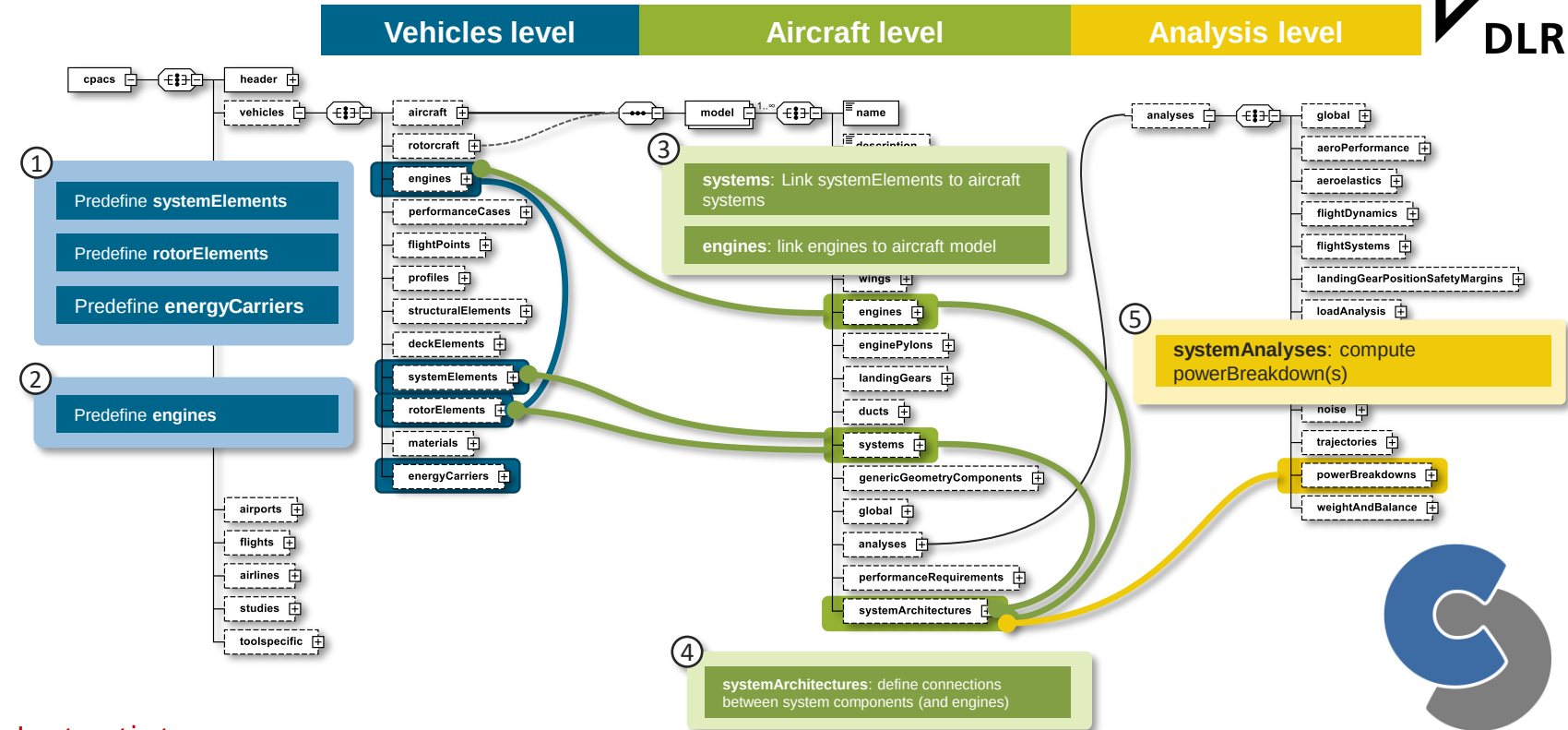
Link elements
Link ADORE elements to CPACS elements.

Name	Turbofan
Reference	turbofan
Linked CPACS elements	turbofan
☐ engine: Turbofan Engine	

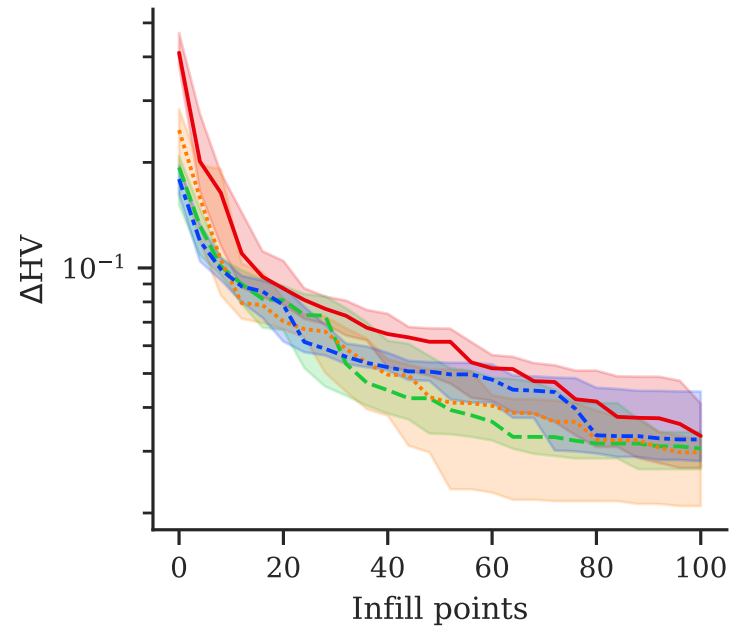
Instantiate
Automatically create CPACS architecture elements.



```
<systemArchitectures>
  <systemArchitecture>
    <connections>
      <connection uID="Electric_Motor_0_d6f0c9_to_Propeller_0_50d6d9">
        <source>
          <componentUID>Electric_Motor_0_d6f0c9</componentUID>
        </source>
        <target>
          <componentUID>Propeller_0_50d6d9</componentUID>
        </target>
      </connection>
    </connections>
  </systemArchitecture>
</systemArchitectures>
```



Burschky, T. et al. (2025) Introduction of a System Definition in the Common Parametric Aircraft Configuration Schema (CPACS). MDPI Aerospace, 12 (5). doi: [10.3390/aerospace12050373](https://doi.org/10.3390/aerospace12050373)

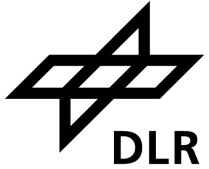


 **ADORE**

The ADORE logo features a blue icon on the left, which is a cube with a bracket-like shape extending from its side. Inside the bracket are three symbols: a cross (x) at the top, a checkmark (✓) in the middle, and another cross (x) at the bottom. To the right of this icon, the word "ADORE" is written in a large, dark blue, serif typeface.

Running the Optimization Problem

SAO Problem Challenges

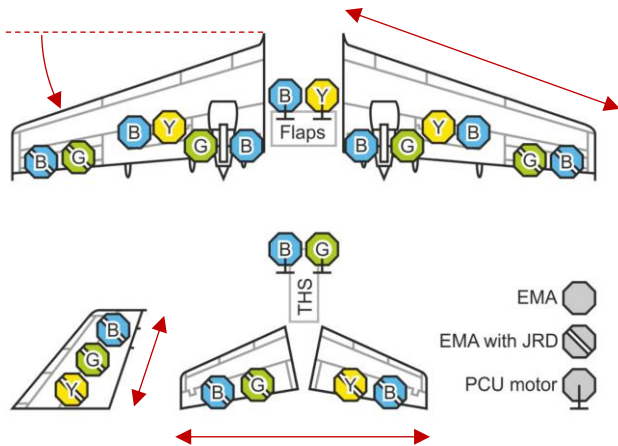


Mixed-Discrete

Discrete architectural choices

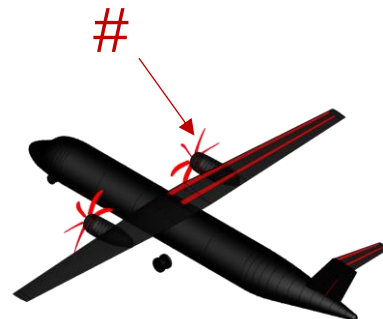
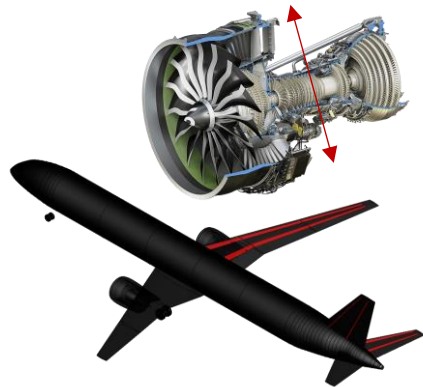
Large combinatorial design space

Continuous design parameters



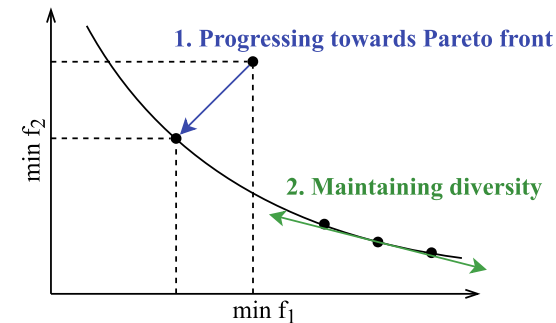
Hierarchical

Conditionally active design choices



Multi-Objective

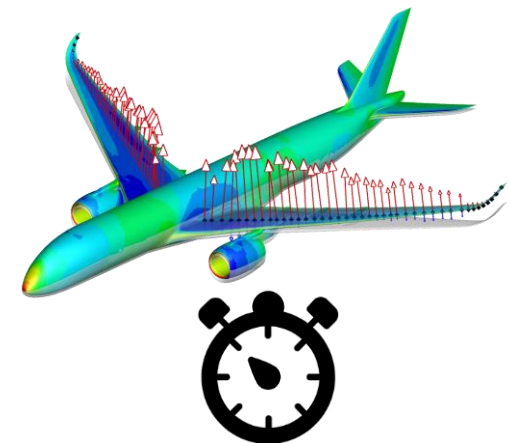
Conflicting stakeholder needs



Black-box

Non-linear, expensive evaluation

Failing simulations

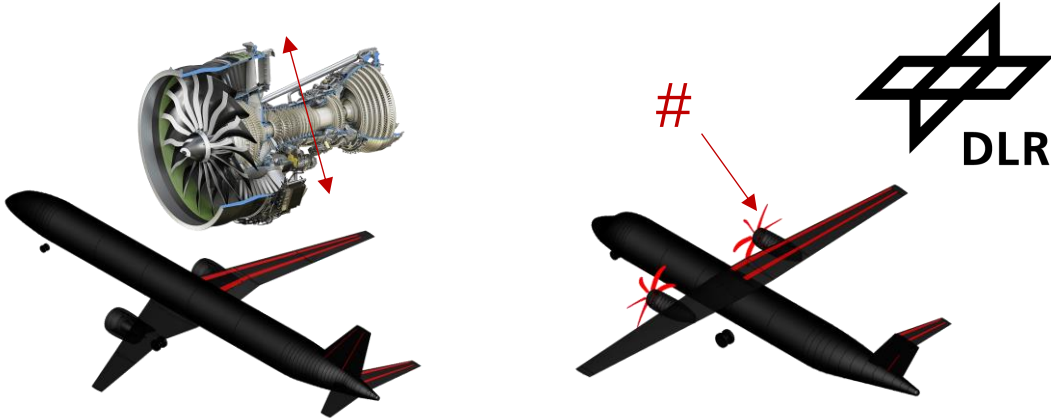


Design Space Challenge: Hierarchy

- Design variables can be **(in)active** due to other design variables
- Multiple declared design vectors can point to the same canonical design vector

Imputation

- Ensures a design vector is canonical
- Replace inactive variables by 0 (discrete) or mid-bounds (continuous)
- Imputation ratio $IR = n_{declared}/n_{valid}$
- Activeness vector (δ) specifies which design variables are active in a design vector



Declared design vectors

engine	BPR	n_blades
tprop	6	3
tprop	6	4
tprop	12	3
tprop	12	4
tfan	6	3
tfan	6	4
tfan	12	3
tfan	12	4

Valid design vectors

engine	BPR	n_blades
tprop	inactive	3
tprop	inactive	4
tfan	6	inactive
tfan	12	inactive

Activeness δ

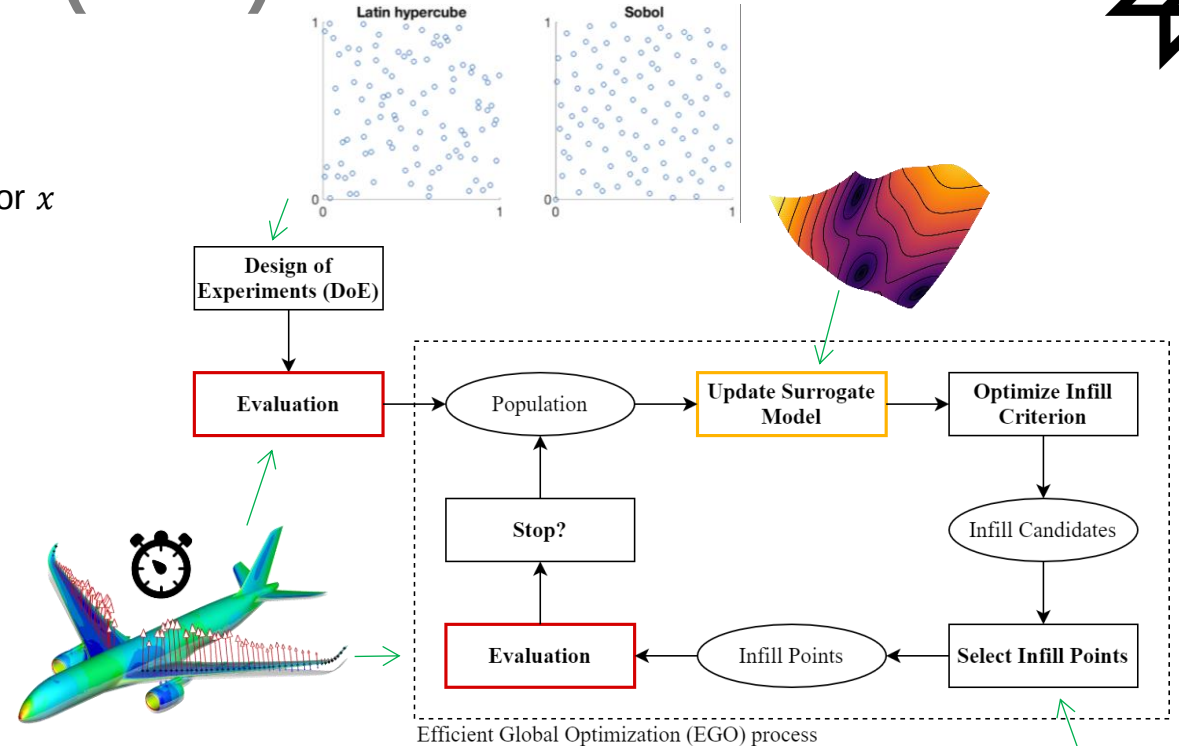
engine	BPR	n_blades
active	inactive	active
active	inactive	active
active	active	inactive
active	active	inactive

Surrogate-Based Optimization (SBO)

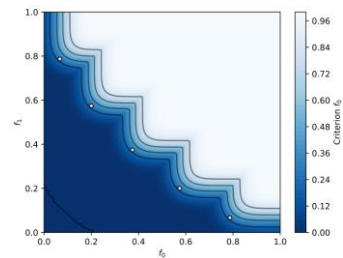
- Train a regression model of the design space
 - Predict objectives f and constraints g given a design vector x
- Use this model to propose interesting new x
- Balance exploration vs exploitation
- Supports multi-objective optimization

Bayesian Optimization (BO)

- SBO using Gaussian Processes (Kriging)
- Mean μ and uncertainty σ estimates
- Supports mixed-discrete, hierarchical variables
- Significant training and infill-search time
- Implemented open-source in SBArchOpt
- Usable out-of-the-box in ADORE



About **10x less evaluations** are needed compared to a genetic algorithm (NSGA-II).



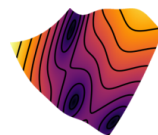
Garnett, R., 2023. *Bayesian optimization*. Cambridge University Press.
Freely available at: <https://bayesoptbook.com/>

SBArchOpt: Surrogate-Based Architecture Optimization

Tests passing | pip install sbarchopt | 2025-10-21 10:05:05 | docs | passing

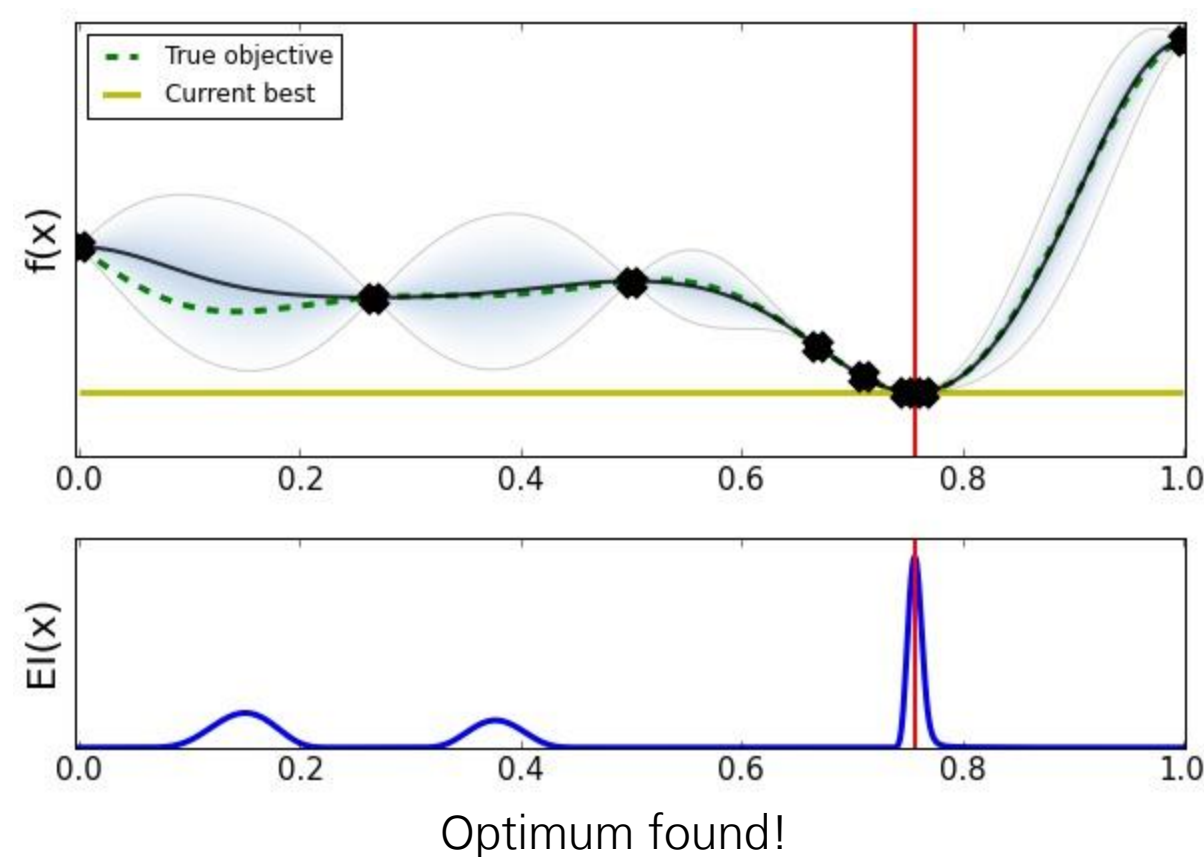
GitHub Repository | Documentation

SBArchOpt (es-bee-ARK-opt) provides a set of classes and interfaces for applying Surrogate-Based Optimization (SBO) for system architecture optimization problems.



SBArchOpt

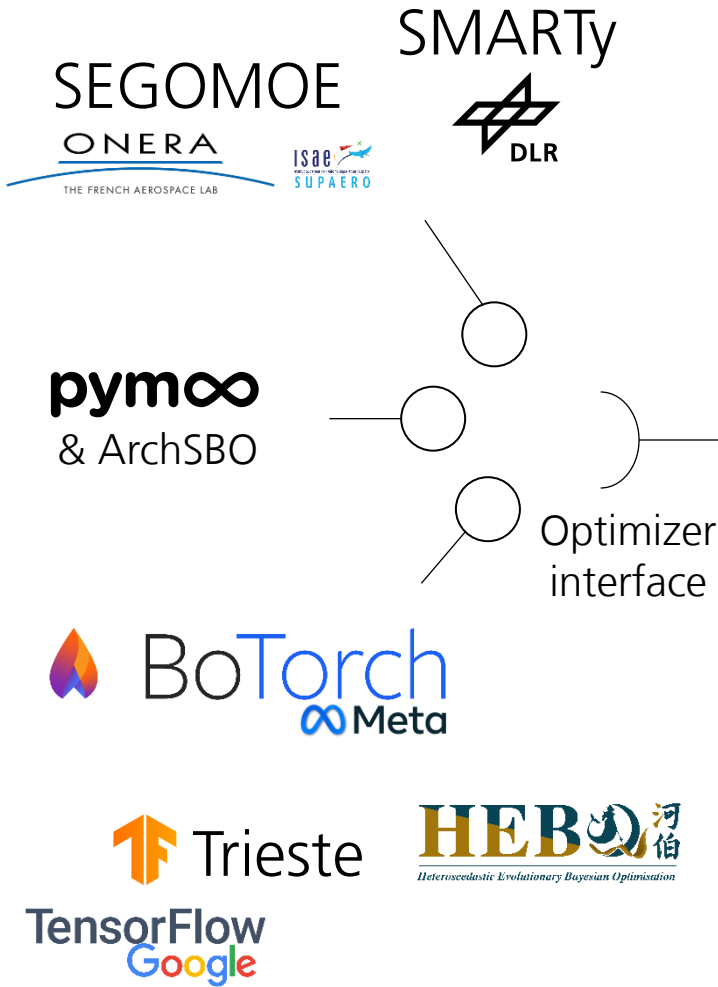
Simple Illustration of Bayesian Optimization



González, J., 2017. *Introduction to Bayesian Optimization*. Gaussian process summer school, Sheffield University.



SBArchOpt Library

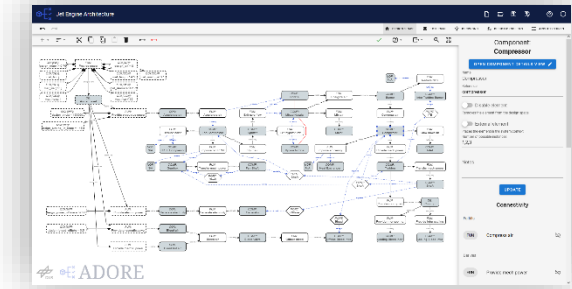



SBArchOpt

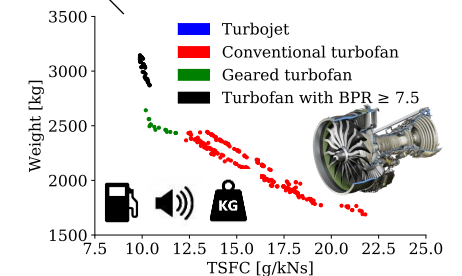
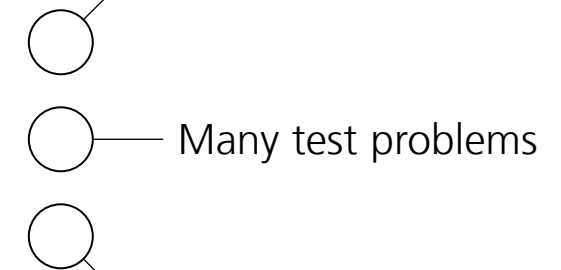
<https://sbarchopt.readthedocs.io/>
doi: [10.21105/joss.05564](https://doi.org/10.21105/joss.05564)
pip install sb-arch-opt

- **Problem definition** based on pymoo*, modified for arch opt (mixed-discrete, hierarchy, hidden constraints, restart)
- Library of **test problems** (many analytical problems, GN&C, jet engine problem)
- **Connectors** to SBO frameworks
- Good documentation and tutorials

Problem interface



`evaluator.get_arch_opt_problem()`



Executing the Optimization Problem

From Python

Execute the optimization from a Python script.

Setup and run
Create problem, select and configure the optimization algorithm, and execute the optimization.

```
# Instantiate your evaluator from your ADORE model
evaluator = MyPythonEvaluator.from_file('model.adore')

# Connect to SBArchOpt and pymoo for optimization
from sb_arch_opt.algo.arch_sbo import get_arch_sbo_gp
from pymoo.optimize import minimize

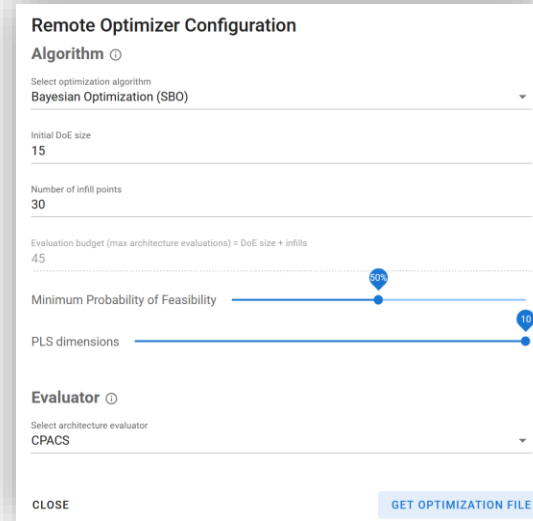
problem = evaluator.get_arch_opt_problem()
algorithm = get_arch_sbo_gp(problem, init_size=50, results_folder='results')
result = minimize(problem, algorithm, termination=('n_eval', 100), verbose=True)

# Save the resulting ADORE file and a CSV export
evaluator.to_file(results.adore')
evaluator.save_results_csv('results.csv')
```

Remote Execution

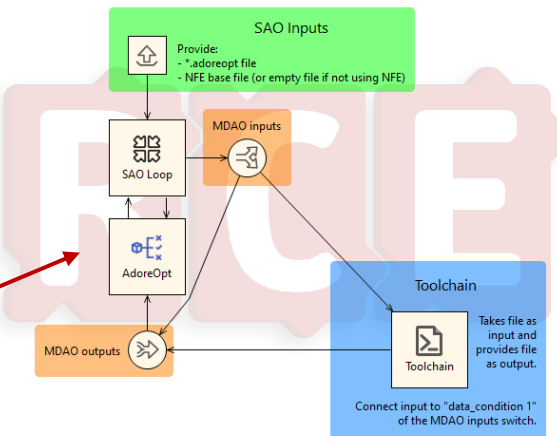
Configure the algorithm and evaluation interface in the GUI. Execute the optimization in an external environment.

START REMOTE OPTIMIZATION

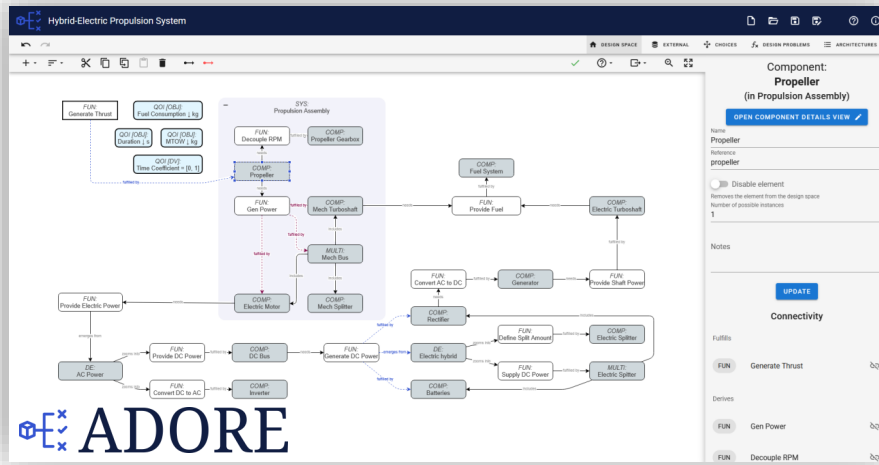


The GUI shows configuration options for a remote optimizer. It includes a dropdown for the algorithm (Bayesian Optimization (SBO)), input fields for Initial DoE size (15) and Number of infill points (30), a slider for Evaluation budget (45), a slider for Minimum Probability of Feasibility (50%), a slider for PLS dimensions (10), a dropdown for the Evaluator (CPACS), and buttons for CLOSE and GET OPTIMIZATION FILE.

AdoreOpt file
Configuration and state is stored in an .adoreopt file.



References



Contact: adore@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](#)
- 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](#)
- Bussemaker, J.H., 2023. SBArchOpt: Surrogate-based architecture optimization. Journal of Open Source Software, 8(89), p.5564. doi: [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](#)

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

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

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