Automatic Architecture Generation and Optimization in SysML v2

ANSYS Aerospace & Defense Summit | 2026-10-15 | Hamburg

Jasper Bussemaker (jasper.bussemaker@dlr.de)

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

Research at the DLR

AERONAUTICS	SPACE	ENERGY	TRANSPORT
			



SECURITY
civil security & defence research





DIGITALISATION, QUANTUM TECHNOLOGY AND SYSTEM MODELLING



- Europe's largest research centre for aeronautics and space
- Close cooperation with academia, research, business and industry

DLR Institute of System Architectures in Aeronautics

Key data about the institute:

- Founded in November 2017
- Located at ZAL TechCenter in Hamburg
- Director of Institute: Dr.-Ing. Björn Nagel

Departments:

- Aviation System Concepts and Evaluation
- Integrated Cabin and Industrialization
- Digital Methods for System Architecting

Digital Methods for System Architecting

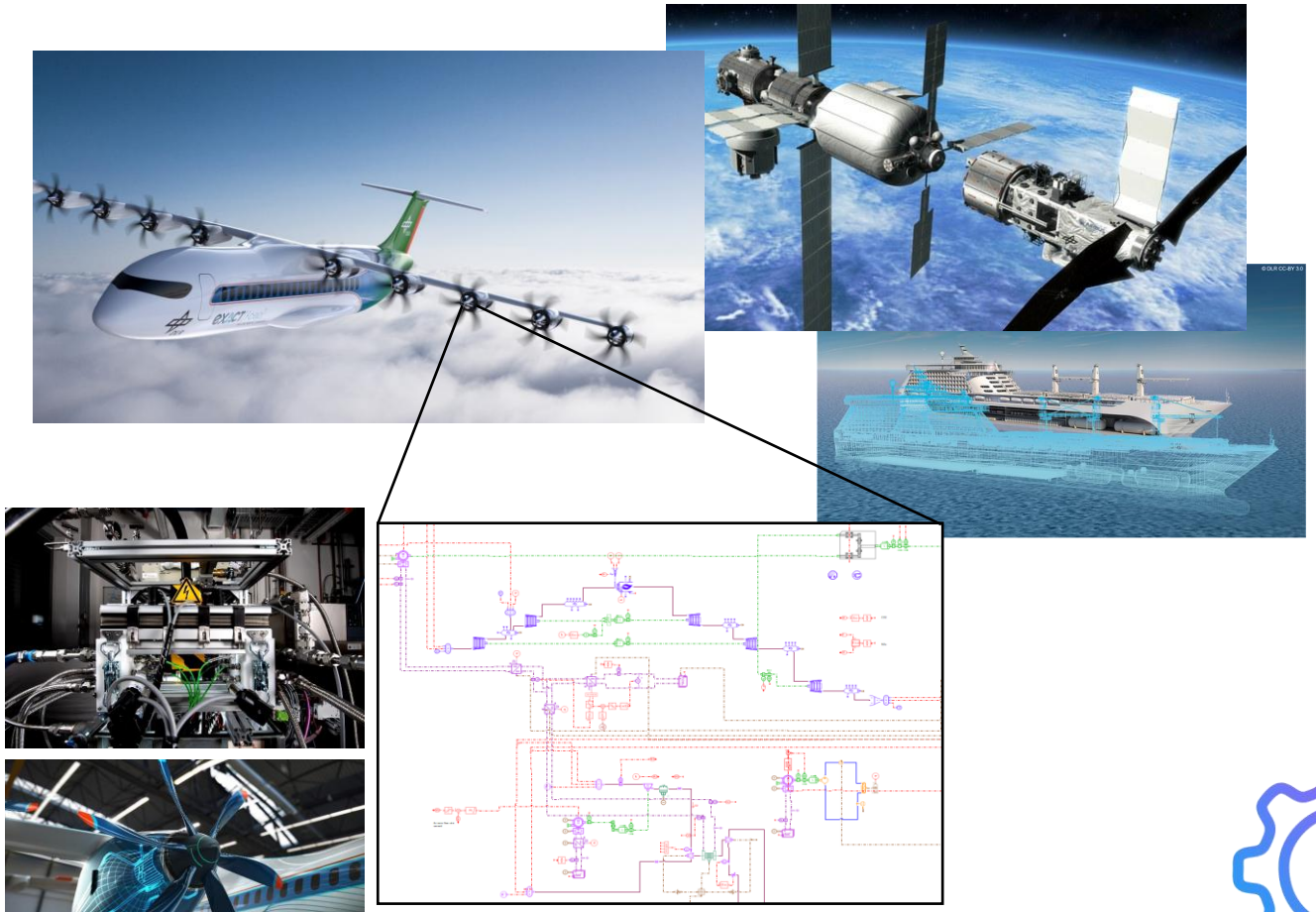
- Development of novel digital design methods
- Digitalization of the system design process
- Optimization of entire system architectures
- Cross-organizational collaborative design
- AI to accelerate and gain insights







DLR's IDEA Project
Integrated Digital Engineering Workbench for System Architecting



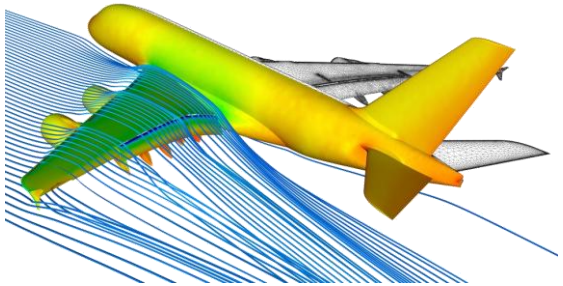
- Increasing system complexity, very large (architectural) design spaces
- Towards *Digital Engineering* in the DLR
- Connect data models and tools using SysML v2
- Increase collaboration and design automation
- 12 institutes: aero, space & maritime
- 2025 to end of 2027

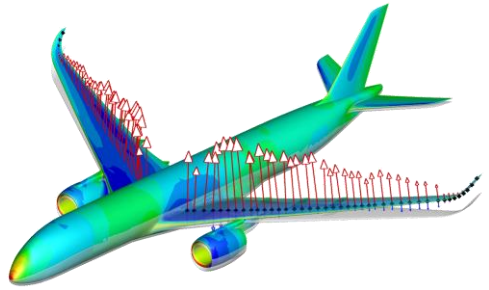
IDEA

Systems Engineering Challenges

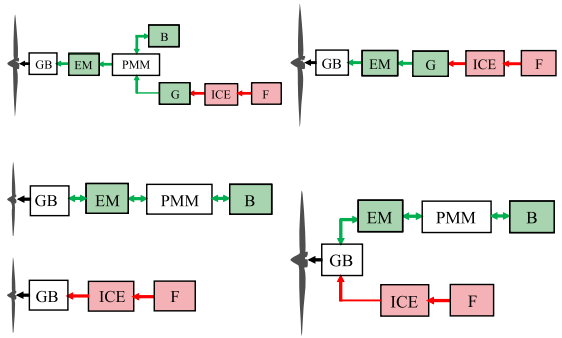
- Non-intuitive design interactions
- New technologies to be integrated

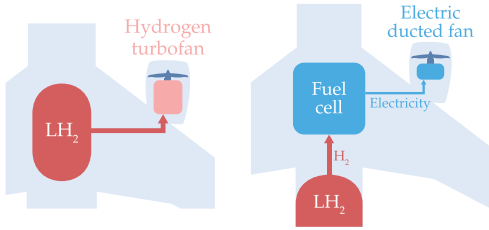
➢ Cannot evaluate system performance based on experience



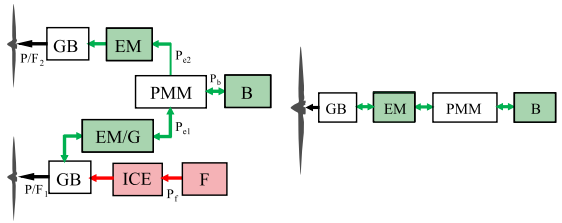


1. Need to run physics-based simulations
   - ➢ Simulations can be computationally expensive
   - ➢ Runtime per simulation in the order of minutes to hours





2. Huge design space
   - ➢ Many possible *system architectures*
   - ➢ Valid combinations in the order of thousands to millions



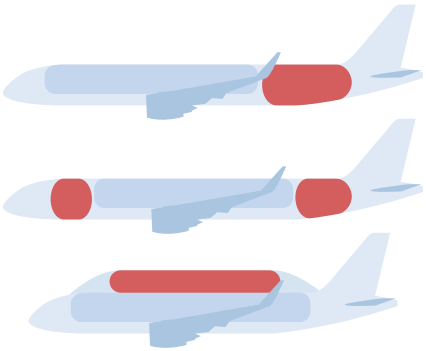
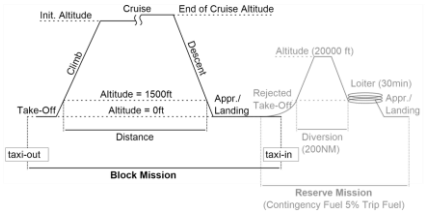


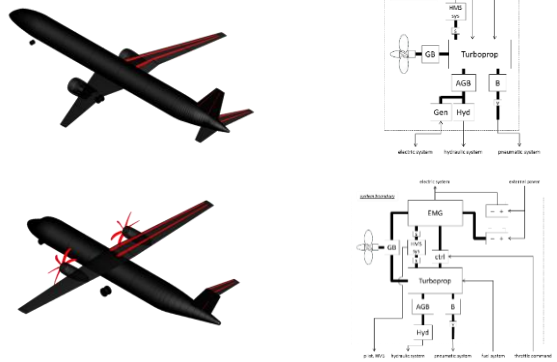
Image copyrights:
CFD and aeroelastic simulation: DLR
Hydrogen propulsion architectures: Adler and Martins
Hybrid-electric architectures: Salem, Palaia and Quarta

From Manual to Automatic Architecting

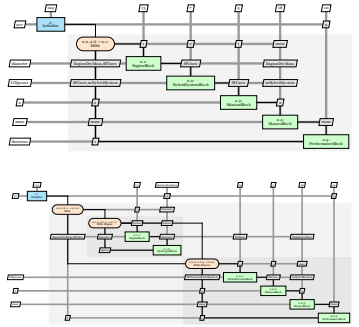
Manual Architecting Process



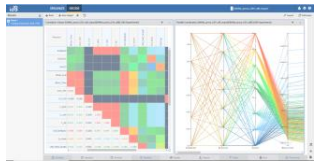
System requirements



Define *several* architectures

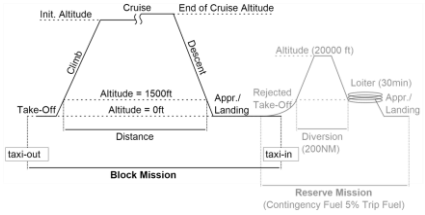


Design & analyze

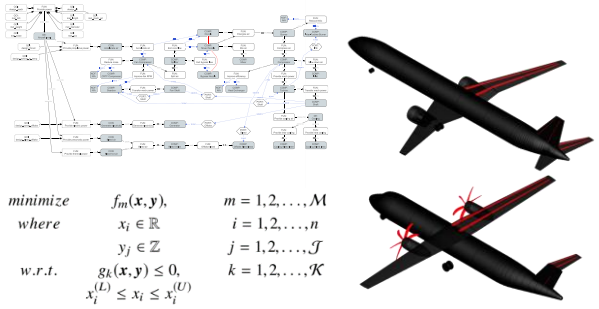


Compare and decide

System Architecture Optimization (SAO) Process

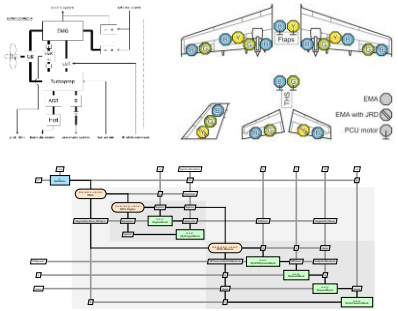


System requirements

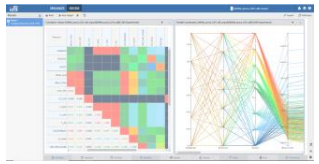


$$\begin{aligned} &\textit{minimize} && f_m(\boldsymbol{x},\boldsymbol{y}), && m = 1,2,\ldots,\mathcal{M} \\ &\textit{where} && x_i \in \mathbb{R} && i = 1,2,\ldots,n \\ & && y_j \in \mathbb{Z} && j = 1,2,\ldots,\mathcal{J} \\ &\textit{w.r.t.} && g_k(\boldsymbol{x},\boldsymbol{y}) \leq 0, && k = 1,2,\ldots,\mathcal{K}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

Define architecture optimization problem



Generate, design & analyze *many* architectures



Compare and decide



```
private import ArchitectureDesign::*;

abstract #architectureDesignSpace part def DesignSpace {

    #evalOut attribute specificFuelConsumption {
        :>> role = OutputRole::MinimizationObjective; }

    action generateThrust : GenerateThrust;

    part turbofan : Turbofan {
        perform generateThrust;
        ref #needsAction action references provideFuel;

        #contX attribute bypassRatio {
            :>> lowerBound = 5.0; :>> upperBound = 12.0; }
    }
    part turboprop : Turboprop {
        perform generateThrust;
        ref #needsAction action ::> provideFuel;
    }

    action provideFuel;
    part fuelSystem { perform provideFuel; }
}

#architecture part def turbofanPropSys specializes DesignSpace {

    action :>> generateThrust;

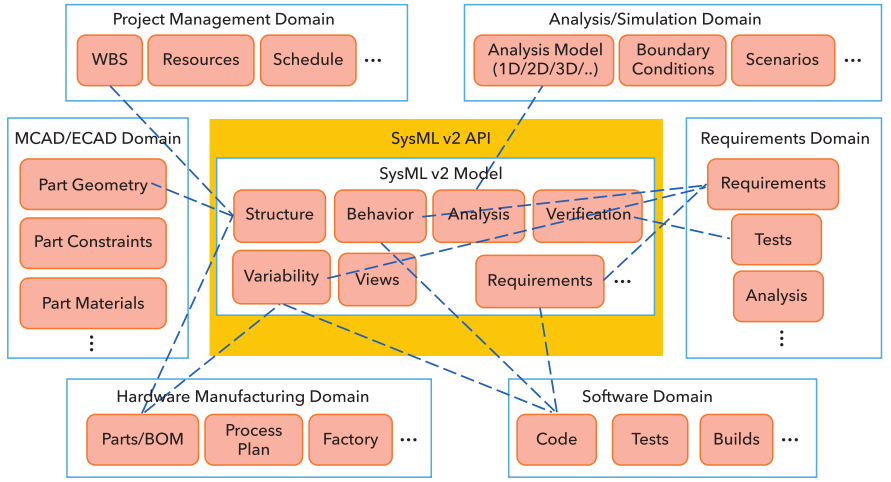
    part :>> turbofan {
        :>> bypassRatio = 10.0;
        :>> weight = 5000[SI::kg];
    }
    part :>> turboprop[0] = null;
}
```

Automatic Architecting with SysML v2

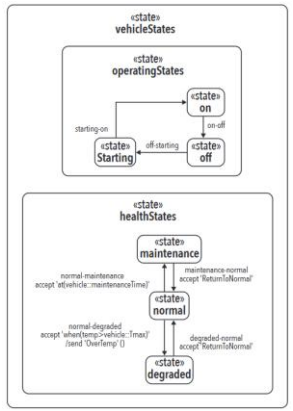
Why SysML v2?

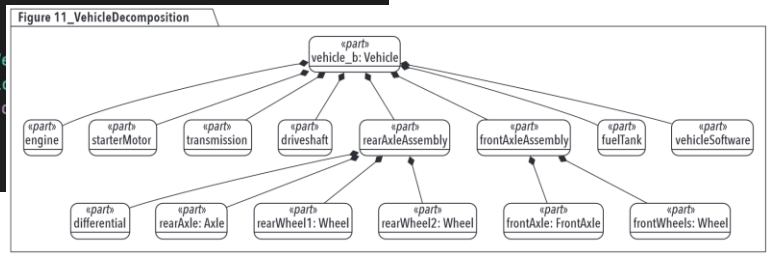
New version of the Systems Modeling Language:

- A **semantic and standardized** MBSE language covering many aspects of systems engineering, such as requirements, architecture, behavior, and analysis.
- A continuation of the **well-established SysML** language, thus benefiting from two decades of experience in MBSE and widespread industry adoption.
- A **standardized API definition**, enabling flexible interchanging of SysML v2 providers and modular integration of external engineering tools.
- Standardized graphical and text-based **representations**, for viewing and editing the central repository models.
- A **packaging system** enabling modular and shareable (meta)model definitions.



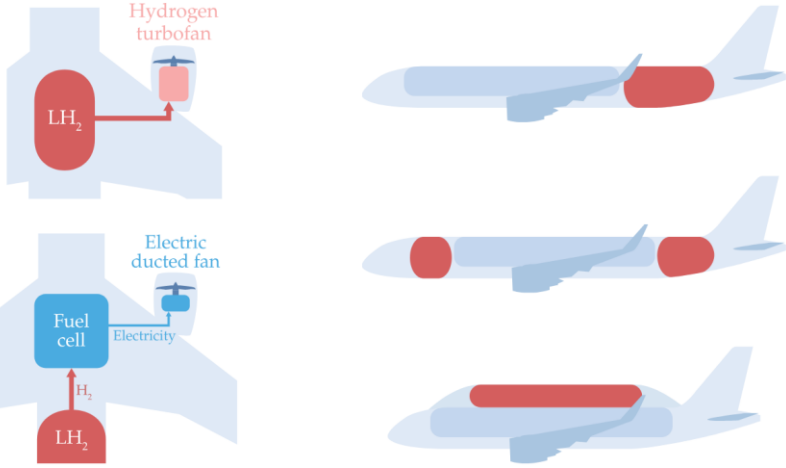

```
1  package Sample {
2      import SI::*;
3      import NumericalFunctions::sum;
4
5      enum def Colors {black;grey;red;}
6
7      enum def DiameterChoices :> ISQ::LengthValue{
8          enum = 60 [mm];
9          enum = 80 [mm];
10         enum = 100 [mm];
11     }
12
13     part def Vehicle {
14         doc /* This is a doc about vehicle */
15         /* This is a comment */
16         attribute partMasses : ISQ::MassValue[*];
17         attribute totalMass = sum(partMasses);
18     }
19
20     part vehicles : Ve
21     metadata def vehi
22         :>> baseType
23     }
24 }
```

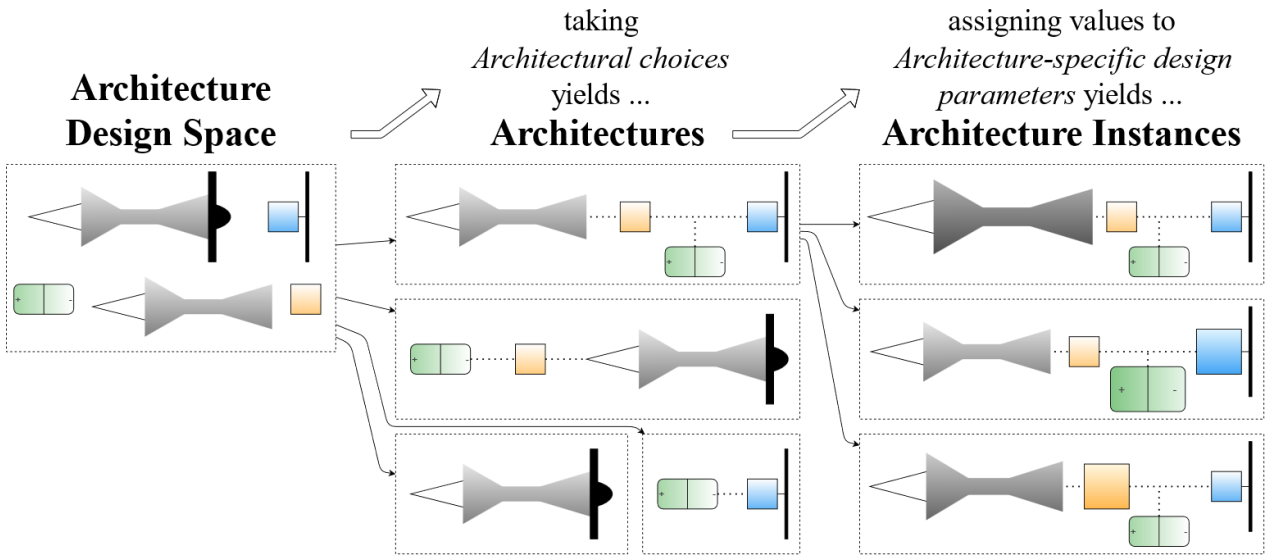




System Architecting with SysML v2

- SysML v2 is appropriate for system architecting, of course…
- However, for *automatic* architecting we need more:
  - Degrees of design freedom: architecture choices (variability), design parameters
    - ➢ The *Architecture Design Space*
  - Metrics for comparing architecture alternatives
  - Automatic metric calculations





Architecture Design Library

SysML v2 library defining concepts for modeling architecture design spaces and architecture instances.



Sysand

```
sysand add ode4hera/architecture-design
```

Design parameters and metrics

```
private import ArchitectureDesign::*;

action def GenerateThrust {
    attribute seaLevelThrust default 200e3[SI::N];
}

part def Turbofan {
    #contX attribute bypassRatio {
        :>> lowerBound = 5.0; :>> upperBound = 12.0; }

    #evaluationOutput attribute weight :> ISQ::mass {
        :>> role = OutputRole::Metric; }

    #minObj attribute specificFuelConsumption;
}

part def Turboprop {
    #disX attribute nrBlades { :>> values = (3, 4, 6, 8); }

    #con attribute bladeTipMach :> ISQ::machNumber {
        assert constraint { that <= 0.9 }
    }
}
```

Architecture design space

```
#architectureDesignSpace part def DesignSpace {

    action generateThrust : GenerateThrust;

    part turbofan : Turbofan {
        perform generateThrust;
        #needsAction references provideFuel;
    }
    part turboprop : Turboprop {
        perform generateThrust;
        #needsAction ::> provideFuel;
    }

    action provideFuel;
    part fuelSystem { perform provideFuel; }
}
```

Architecture instance derived from design space

```
#architecture part def TurbofanPropSys
        specializes DesignSpace {

    action :>> generateThrust {
        :>> seaLevelThrust = 200e3[SI::N];
    }

    part :>> turbofan {
        :>> bypassRatio = 10.0;
        :>> weight = 5000[SI::kg];
        :>> specificFuelConsumption;
    }

    part :>> turboprop[0] = null;
}
```

Defining the #architectureDesignSpace

Function allocation and induction

Top-level boundary functions are always part of an architecture.

System functions (actions) can be performed by a part.

Parts can also *need* additional functions.

```
abstract #architectureDesignSpace part def DesignSpace {
    action generateThrust :> boundaryFunctions;
    part turbofan {
        perform generateThrust;
        #needsAction references provideFuel;
    }
    action provideFuel;
}
```

Function allocation choice

Multiple parts performing an action represents an architectural choice.

```
abstract #architectureDesignSpace part def DesignSpace {
    action generateThrust :> boundaryFunctions;
    part turbofan [0..1] { perform generateThrust; }
    part turboprop [0..1] { perform generateThrust; }

    :>> architecturalChoices {
        :>> choices[1] = generateThrustFulfillmentChoice;

        occurrence generateThrustFulfillmentChoice
                :> choices : FunctionFulfillmentChoice {
            :>> actionToPerform = generateThrust;
            :>> performingAlternatives[2] = (turbofan, turboprop);
        }
    }
}
```

Part Instantiation, Design Parameters and Metrics

```
abstract #architectureDesignSpace part def PropulsionSystem {

    // System-level objective
    #minObj attribute mass :> ISQ::mass;

    action generateThrust :> boundaryFunctions {
        // Static input
        attribute seaLevelthrust = 150e3[SI::N]; }

    // Part instantiation choice: between 2 and 4 times
    part turboprop[2..4] {
        perform generateThrust;
        #needsAction ::> provideFuel;

        // Design variables
        #intX attribute nrStages { :>> lowerBound = 1; :>> upperBound = 3; }
        #catX attribute bladeManufacturer { :>> values = ("MT", "Sensenich"); }
        #contX attribute pressureRatio { :>> lowerBound = 10; :>> upperBound = 30; }

        // Objective, constraint and output metric
        #minObj attribute tsfc;
        #con attribute turbineTemp {
            assert constraint { that <= 1500[SI::'°C_abs‘] } }
        attribute mass :> ISQ::mass;
    }

    action provideFuel;
    // Subsystem instantiation choice: 1 or 2 times
    #subsystem part fuelSystem[1..2] {
        // Independent part instantiation choice
        part fuelTank[1..2] {
            perform provideFuel;
            attribute mass :> ISQ::mass; }
    }
}
```

Part and subsystem instantiation choices.

Design variables and performance metrics (objectives, constraints).

Deriving #architecture Instances

```
abstract #architectureDesignSpace part def TorqueGenDesSpace {
    action generateTorque[1] :> boundaryFunctions;

    part electricMotor[0..1] {
        perform generateTorque;
        #needsAction references provideElectricity; }

    part turboshaft[0..1] {
        // Optional function fulfillments
        perform action p1 ::> generateTorque[0..1];
        perform action p2 ::> providePower[0..1];
        #needsAction ::> provideFuel; }

    action provideElectricity[0..1];
    part battery[0..1] { perform provideElectricity; }
    part generator[0..1] {
        perform provideElectricity;
        #needsAction ::> providePower; }

    action providePower[0..1];
    part combustionEngine[0..1] {
        perform providePower;
        #needsAction ::> provideFuel; }

    action provideFuel[0..1];
    part fuelSystem[0..1] { perform provideFuel; }
}
```



Architecture instances are specializations of the architecture design space with *resolved* architectural choices.

Parts are only included if actually allocated to an action. Induced actions are only included if needed by an included part.

```
#architecture part def SeriesHybrid specializes TorqueGenDesSpace {
    action :>> generateTorque[1];

    part redefines electricMotor[1];
    part redefines turboshaft[1] {
        // Turboshaft only fulfills "providePower"
        perform generateTorque[0] redefines p1;
        perform providePower[1] :>> p2;
    }

    action :>> provideElectricity[1];
    part :>> battery[0] = null; // Removed element
    part :>> generator[1];

    action :>> providePower[1];
    part :>> combustionEngine[0] = null;

    action :>> provideFuel[1];
    part :>> fuelSystem[1];
}
```

```
private import ArchitectureDesign::*;

abstract #architectureDesignSpace part def DesignSpace {

    #evalOut attribute specificFuelConsumption {
        :>> role = OutputRole::MinimizationObjective; }

    action generateThrust : GenerateThrust;

    part turbofan : Turbofan {
        perform generateThrust;
        ref #needsAction action references provideFuel;

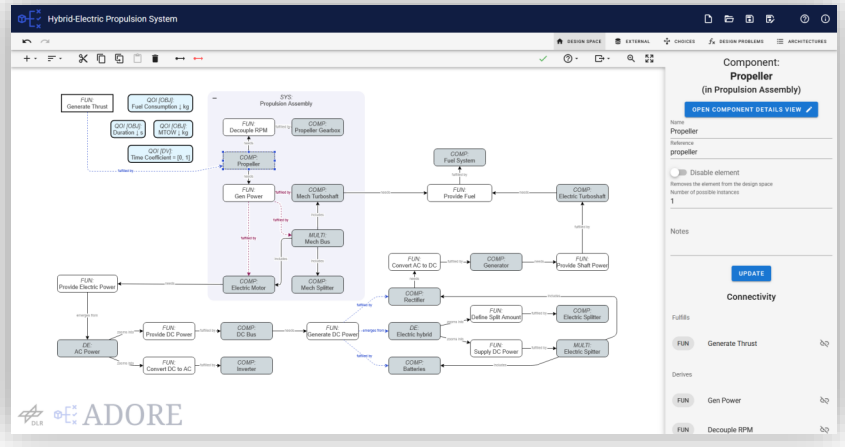
        #contX attribute bypassRatio {
            :>> lowerBound = 5.0; :>> upperBound = 12.0; }
    }
    part turboprop : Turboprop {
        perform generateThrust;
        ref #needsAction action ::> provideFuel;
    }

    action provideFuel;
    part fuelSystem { perform provideFuel; }
}

#architecture part def turbofanPropSys specializes DesignSpace {

    action :>> generateThrust;

    part :>> turbofan {
        :>> bypassRatio = 10.0;
        :>> weight = 5000[SI::kg];
    }
    part :>> turboprop[0] = null;
}
```



ADORE

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

ADORE: The Architecture Generator

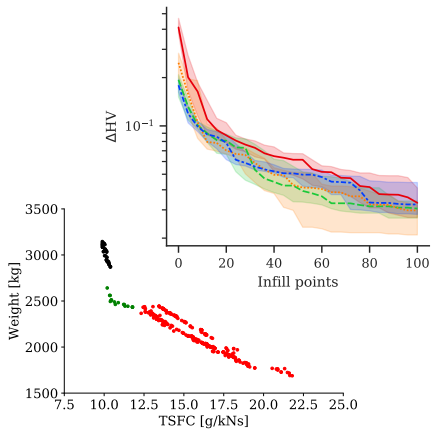
System Architecture Optimization with ADORE and SysML v2

Explore and optimize

- Automatically generate architectures
- Driven by machine learning algorithms

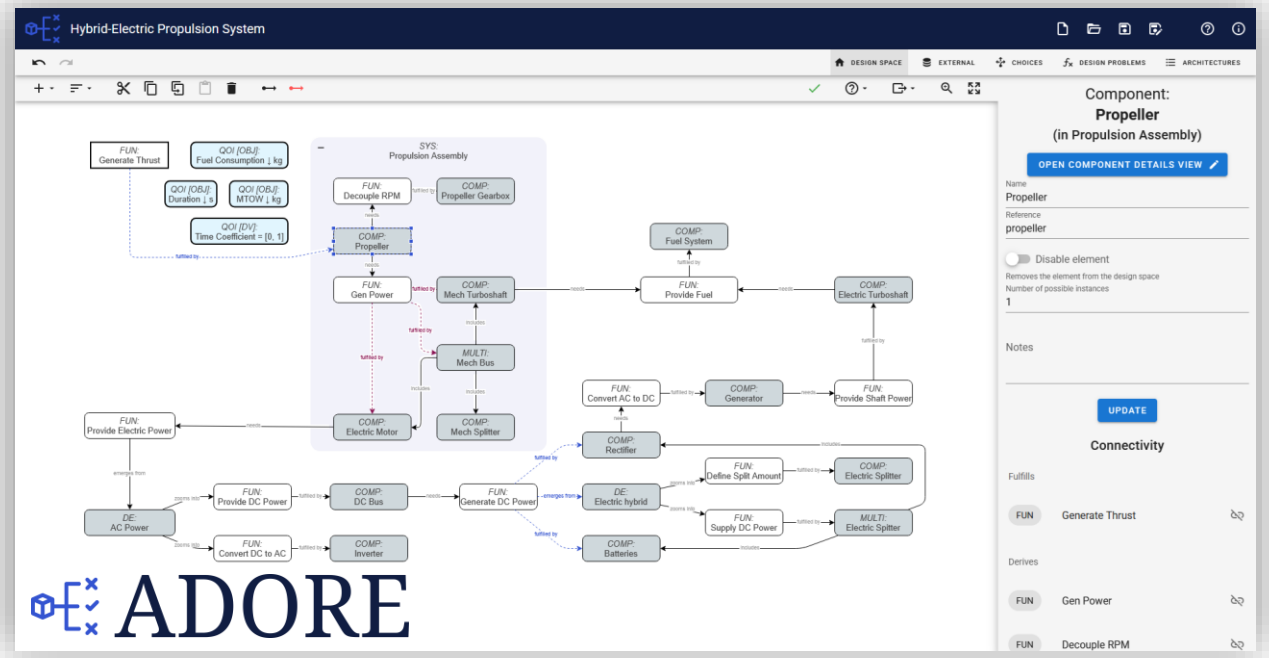
SBArchOpt

pymoo



Model the architecting problem

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



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

Connect to your MBSE platform

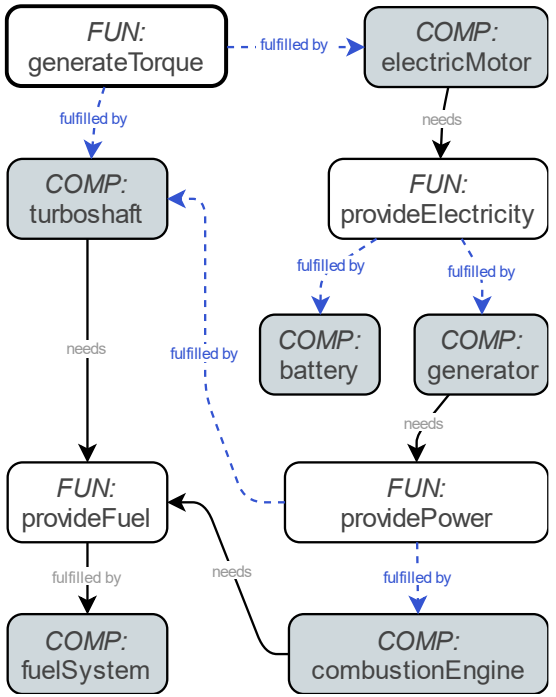
- Based on SysML v2
- Local or remote execution

SysML v2

Model Synchronization

Full bi-directional synchronization: edit the design space model in SysML v2, in the ADORE GUI, or both.

ADORE additionally:
- Shows architectural choices
- Calculate statistics
- Allows you to manually generate architecture instances
- Allows you to define and execute optimization problems




```
abstract #architectureDesignSpace part def TorqueGenDesSpace {
    action generateTorque[1] :> boundaryFunctions;

    part electricMotor[0..1] {
        perform generateTorque;
        #needsAction references provideElectricity; }

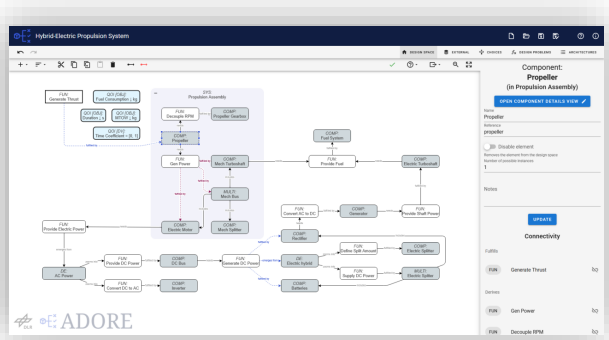
    part turboshaft[0..1] {
        // Optional function fulfillments
        perform action p1 ::> generateTorque[0..1];
        perform action p2 ::> providePower[0..1];
        #needsAction ::> provideFuel; }

    action provideElectricity[0..1];
    part battery[0..1] { perform provideElectricity; }
    part generator[0..1] {
        perform provideElectricity;
        #needsAction ::> providePower; }

    action providePower[0..1];
    part combustionEngine[0..1] {
        perform providePower;
        #needsAction ::> provideFuel; }

    action provideFuel[0..1];
    part fuelSystem[0..1] { perform provideFuel; }
}
```

The SysML v2 System Architecture Optimization Loop



The ADORE Model

Defines architectural choices and metrics.

synchronized

The SysML v2 Model

`model.sysml`

```
#architectureDesignSpace part torqueGenDesSpace {
    action generateTorque[1] :> boundaryFunctions;

    part electricMotor[0..1] {
        // Function fulfillment and induction
        perform generateTorque;
        #needsAction references provideElectricity; }

    part turboshaft[0..1] {
        // Optional function fulfillments
        perform action p1 ::> generateTorque[0..1];
        perform action p2 ::> providePower[0..1];
        #needsAction ::> provideFuel; }
```

specializes

`architecture_<#>.sysml`

```
#architecture part seriesHybrid :> torqueGenDesSpace {

    part redefines electricMotor[1];
    part redefines turboshaft[1] {
        // Turboshaft only fulfills "providePower"
        perform generateTorque[0] redefines p1;
        perform providePower[1] :>> p2;
    }

    action :>> provideElectricity[1];
    part :>> battery[0]; // Removed element
    part :>> generator[1];

    action :>> providePower[1];
    part :>> combustionEngine[0];

    action :>> provideFuel[1];
    part :>> fuelSystem[1];
}
```

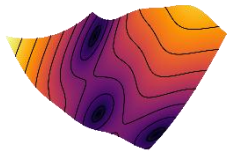
provides

formalizes the optimization problem

The Generator

Generates an architecture instance from x.

each architecture instance is exported to



generates design vector x

The Optimizer

Uses machine learning algorithms.

provides objective f and constraint g values to optimizer

An Evaluator

Evaluates the architecture.

Only uses the SysML v2 model.

and is evaluated by

Demo: GNC Problem in SysML v2

https://github.com/jbussemaker/GNC-SysML-v2-Demo

```
abstract #architectureDesignSpace part def GNCDesignSpace : Object {
    #minObj attribute :>> mass { :>> units = kg; }
    #evalOut attribute :>> failureRate;
    #minObj attribute logFailureRate;



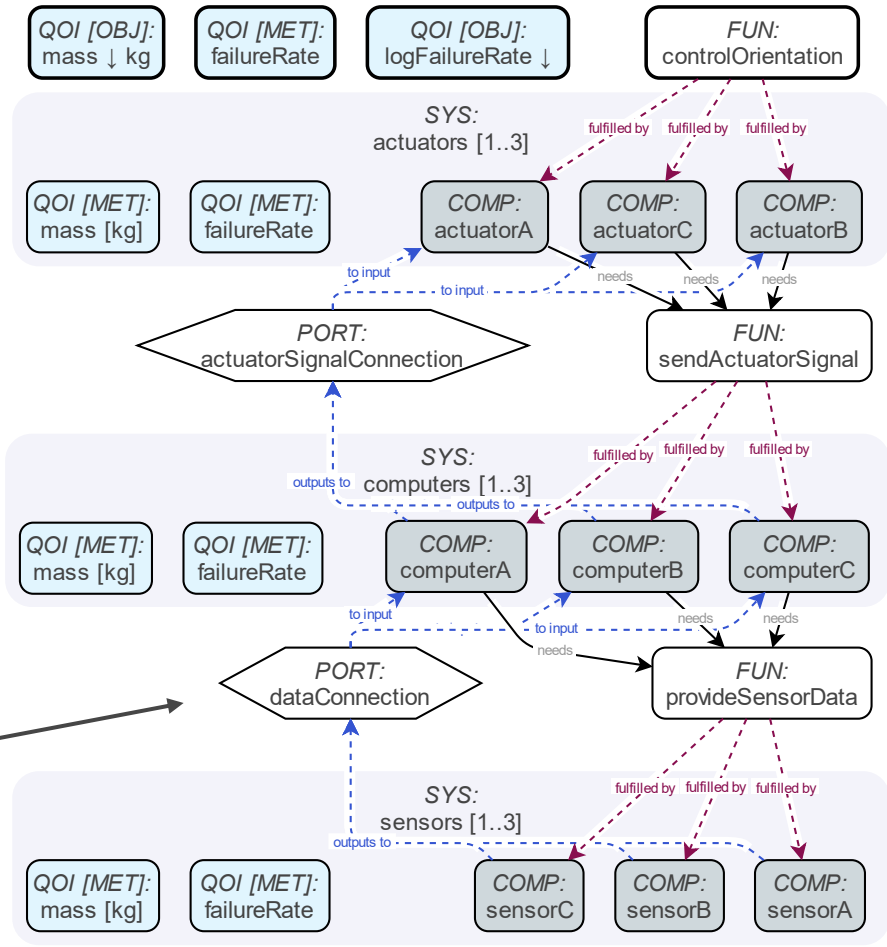
    action sendActuatorSignal [0..1];
    action provideSensorData [0..1];

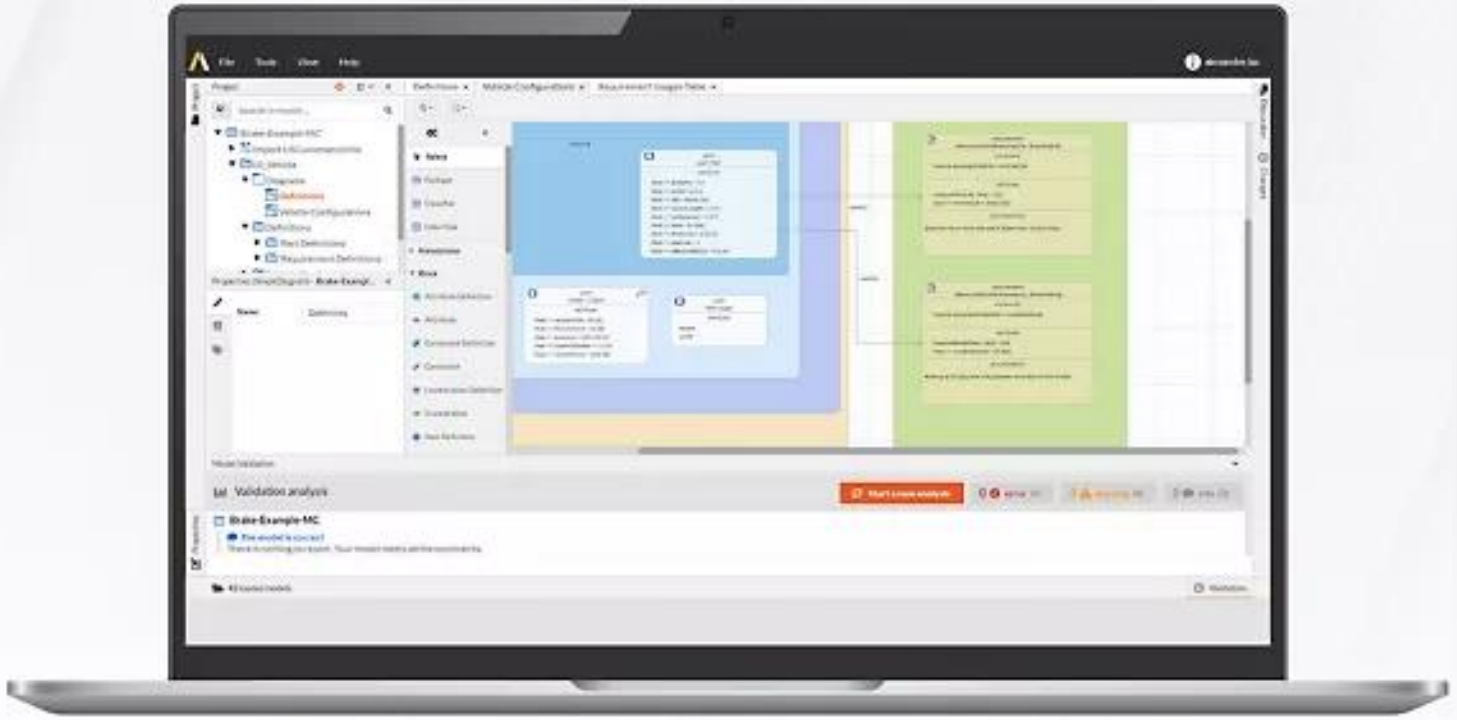
    #subsystem part computers : Object [1..3] {
        #evalOut attribute :>> mass { :>> units = kg; }
        #evalOut attribute :>> failureRate;

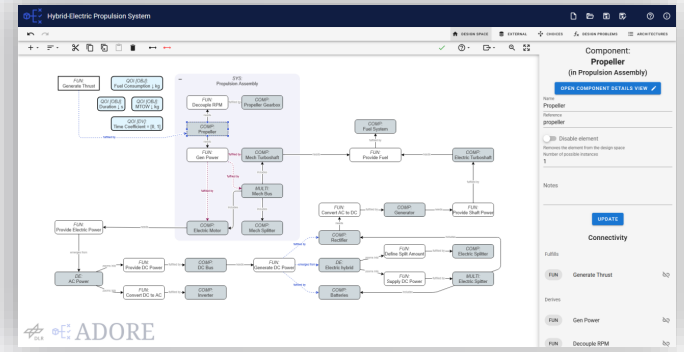
        part computerA : ComputerA [0..1] {
            perform sendActuatorSignal;
            #needsAction ::> provideSensorData;
            attribute :>> mass = 3.0 [kg];
            attribute :>> failureRate = .00010; }
        part computerB : ComputerB [0..1] { ... }
        part computerC : ComputerC [0..1] { ... } }



    connection dataConnection [0..1] connect (
        sensors.sensorA.dataOutPort,
        sensors.sensorB...,
        sensors.sensorC...,
        computers.computerA.dataInPort, ... );
```





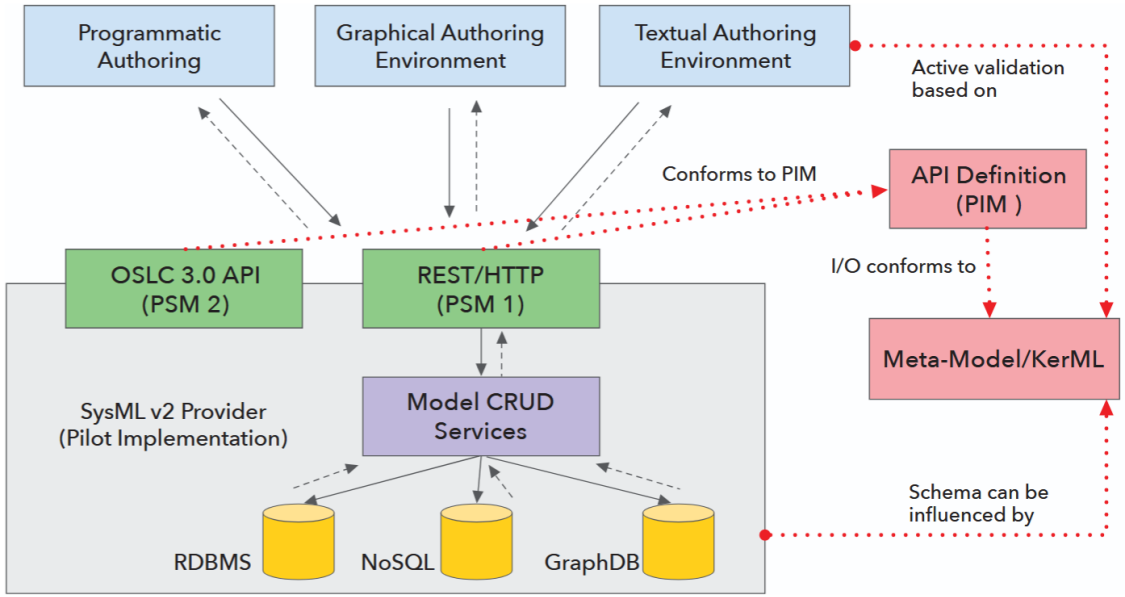


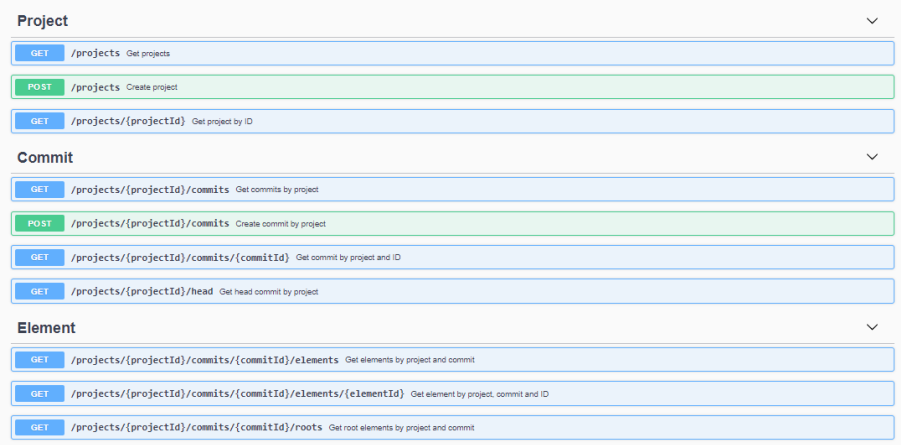
Ansys / SYSTEM ARCHITECTURE MODELER

Vision: Integration with Enterprise MBSE Platforms

SysML v2 API

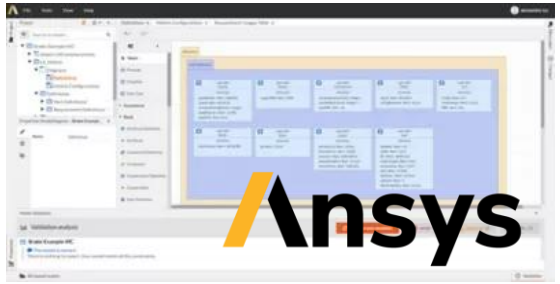
- Standard on top of SysML v2 for tool-independent data access
  - Platform independent model
  - Platform specific models: OSLC and REST
- Enables specialized viewing or authoring tools to connect to any compliant host platform
  - ➢For example….





Top figure from:
M. Bajaj, S. Friedenthal, and E. Seidewitz. "Systems Modeling Language (SysML v2) Support for Digital Engineering". In: INSIGHT 25.1 (Mar. 2022)

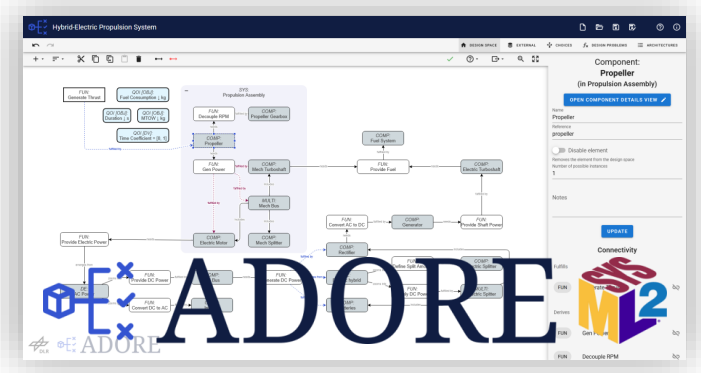
What could it look like in practice?



1. Create a new project and modes (or import) a library of actions and parts.

{REST}

2. Load the model from SAM's REST API.



3. Model the architecture design space.

4. When done, the `#architectureDesignSpace` is committed to the REST API and appears in SAM.

What could it look like in practice?

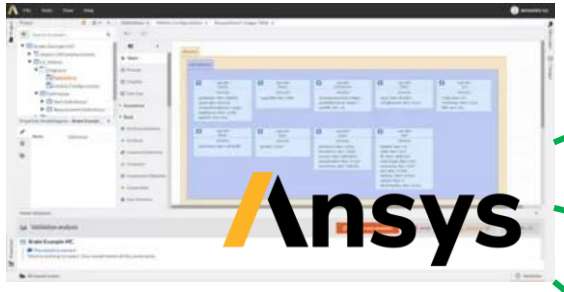
For example SysML v2 expressions…

```
package KBEAircraftExample {
  part def AircraftWing {
    doc /* Definition of a half-wing (i.e. only one side) */

    attribute aspectRatio : ScalarValues::Real;
    attribute area : ISQ::AreaValue;
    attribute taperRatio : ScalarValues::Real;
    attribute sweepAngle : ISQ::AngularMeasureValue;
    attribute isLeftWing : ScalarValues::Boolean default false;

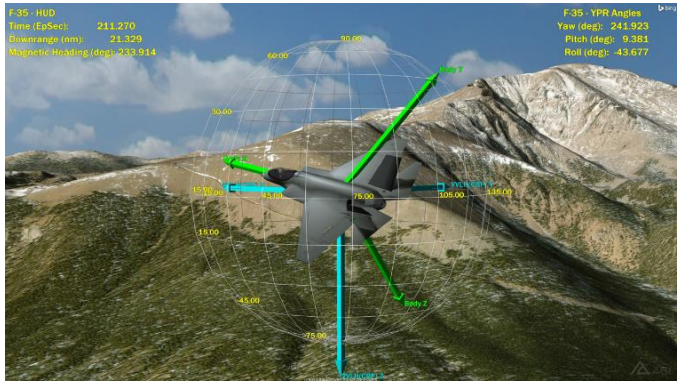
    assert constraint { aspectRatio > 0 }
    assert constraint { area > 0 }
    assert constraint { 0 < taperRatio <= 1 }

    attribute span = RealFunctions::sqrt(aspectRatio / area);
    attribute halfSpan = span / 2;
    attribute rootChordLength = area / (.5 * span * (1 + taperRatio));
    attribute tipChordLength = rootChord * taperRatio;
    attribute tipOffset = .25 * rootChordLength + sin(sweepAngle) * halfSpan - .25 * tipChordLength;
  }
}
```



5. Implement architecture evaluation.

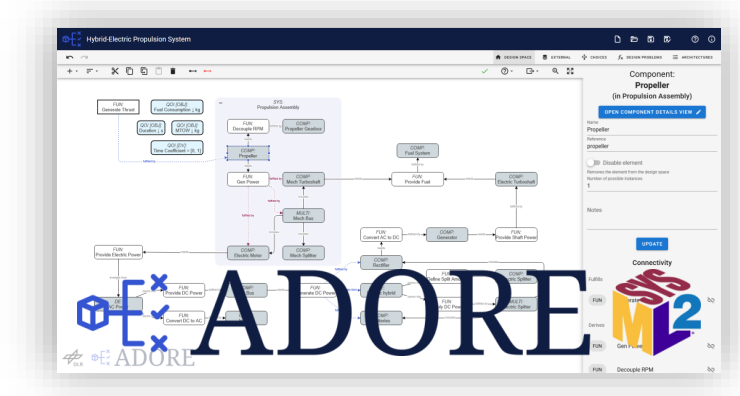
… or STK …



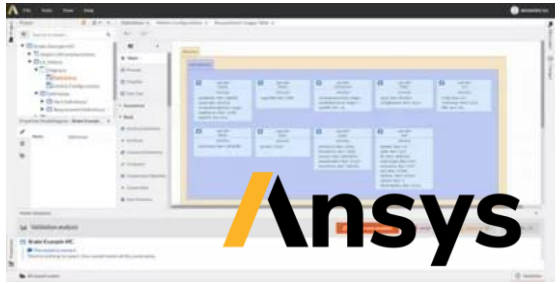
… or ModelCenter.



What could it look like in practice?



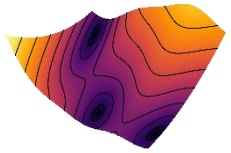
6. Select an optimization algorithm and start the optimization loop.



8. The architecture's performance metrics are calculated.

{REST}

7. The optimizer generates an **#architecture** and commits it to the REST API.



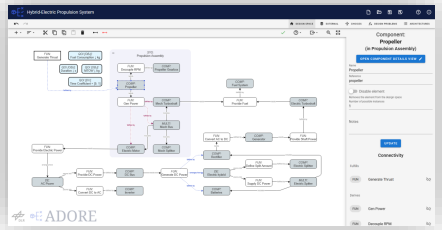
9. Performance metric values are loaded from the REST API.

```
attribute mass :> ISQ::mass = 5000 [SI::kg];
attribute cost = 1000000;
```

10. The optimizer interprets the metrics and starts the next iteration.

Conclusions

- Architecture Optimization in SysML v2
  - ➢ Language extension library
  - ➢ ADORE integration
- ADORE SysML v2 integration
- Future: enterprise MBSE connection




```
#architectureDesignSpace part torqueGenDesSpace {
    action generateTorque[1] :> boundaryFunctions;

    part electricMotor[0..1] {
        // Function fulfillment and induction
        perform generateTorque;
        #needsAction references provideElectricity; }

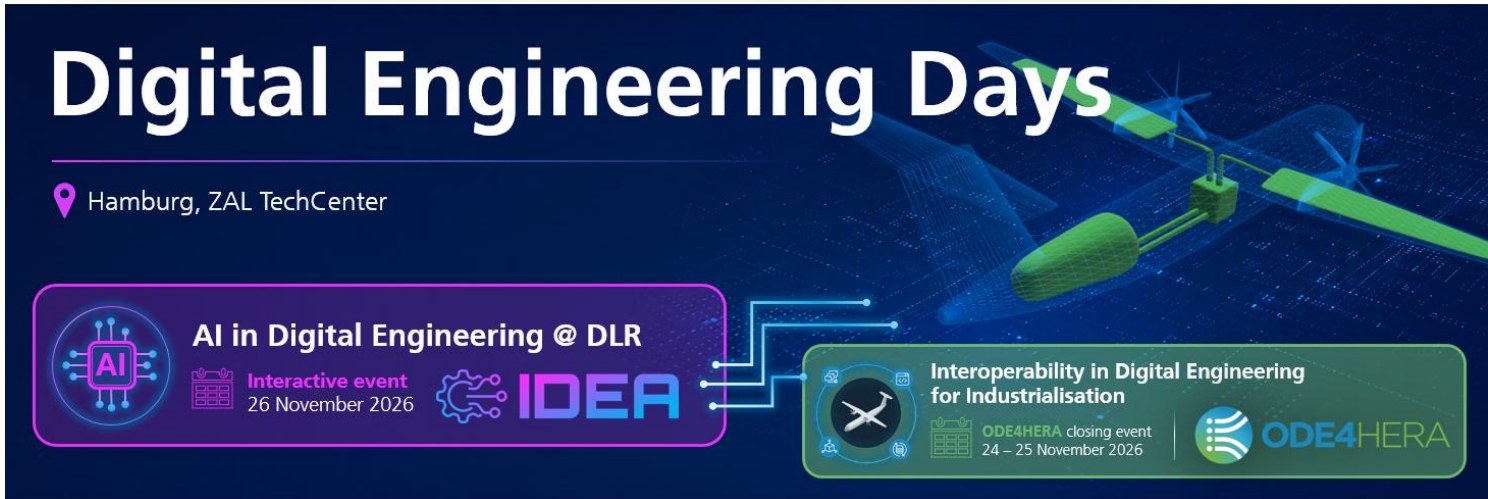
    part turboshaft[0..1] {
        // Optional function fulfillments
        perform action p1 ::> generateTorque[0..1];
        perform action p2 ::> providePower[0..1];
        #needsAction ::> provideFuel; }
```

More from us:

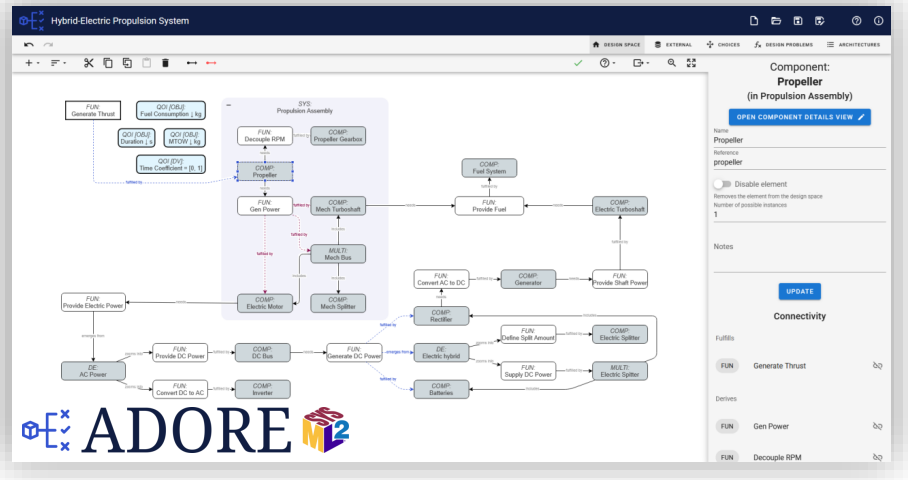
EASN Conference
27-30 Oct 2026 in Toulouse, FR
https://easnconference.eu/

SysML v2 Extravaganza (S2X)
3 Nov 2026 in Hamburg (@ ZAL)
https://www.gfse.org/de/all-events/219-sysml-v2-extravaganza-2026.html

ODE4HERA Closing Event and AI @ DLR
24-26 Nov 2026 in Hamburg (@ ZAL)
https://dlr.expert/digital-engineering-days



References



- Bussemaker, J.H. et al. System Architecture Optimization Using SysML v2: Language Extension and Implementation. In: IEEE SysCon 2026. Halifax, Canada, April 2026. doi: 10.1109/SysCon66367.2026.11503593
- Bussemaker, J.H., 2025, July. *System Architecture Optimization: Function-Based Modeling, Optimization Algorithms, and Multidisciplinary Evaluation*. Dissertation, Delft University of Technology.
- 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

Contact: adore@dlr.de

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

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

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

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