Overview of VAST

Architecture for a new rotor dynamics software

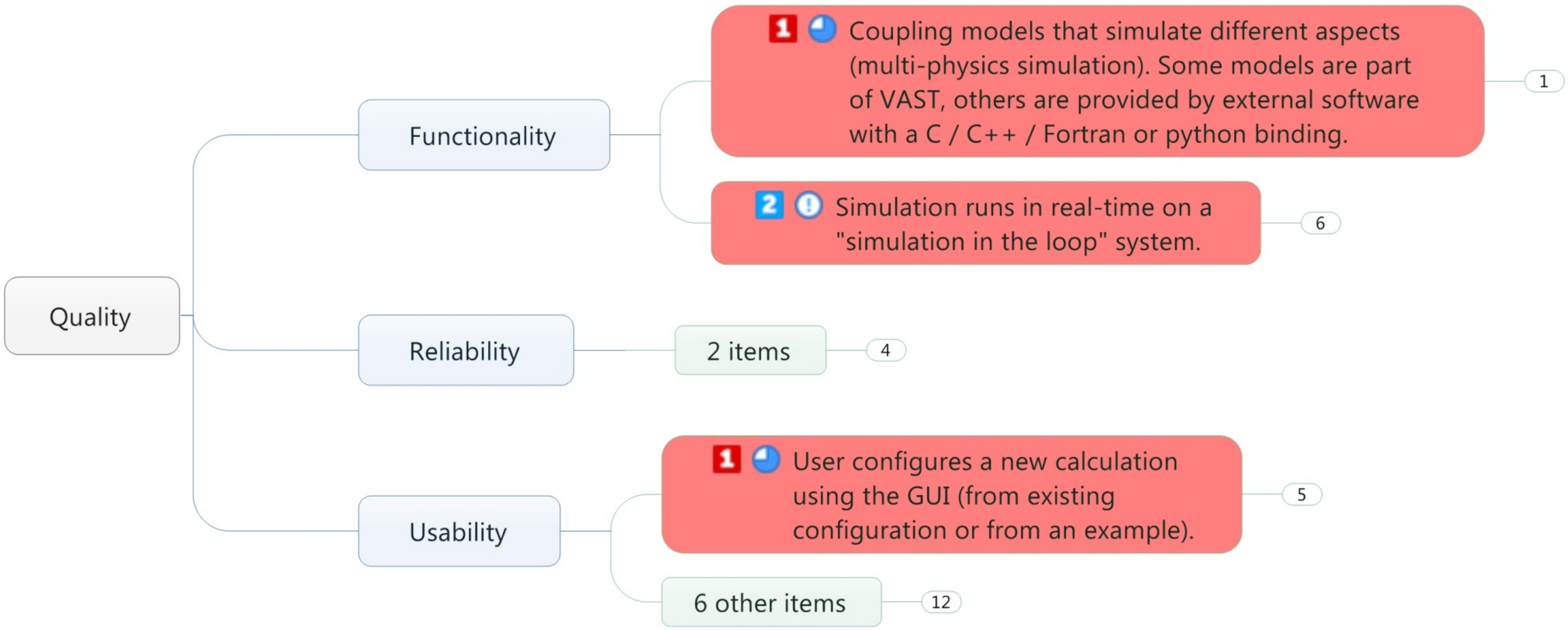
Melven Röhrig-Zöllner (SC-HPC, cooperation with FT-HUB)

DLR

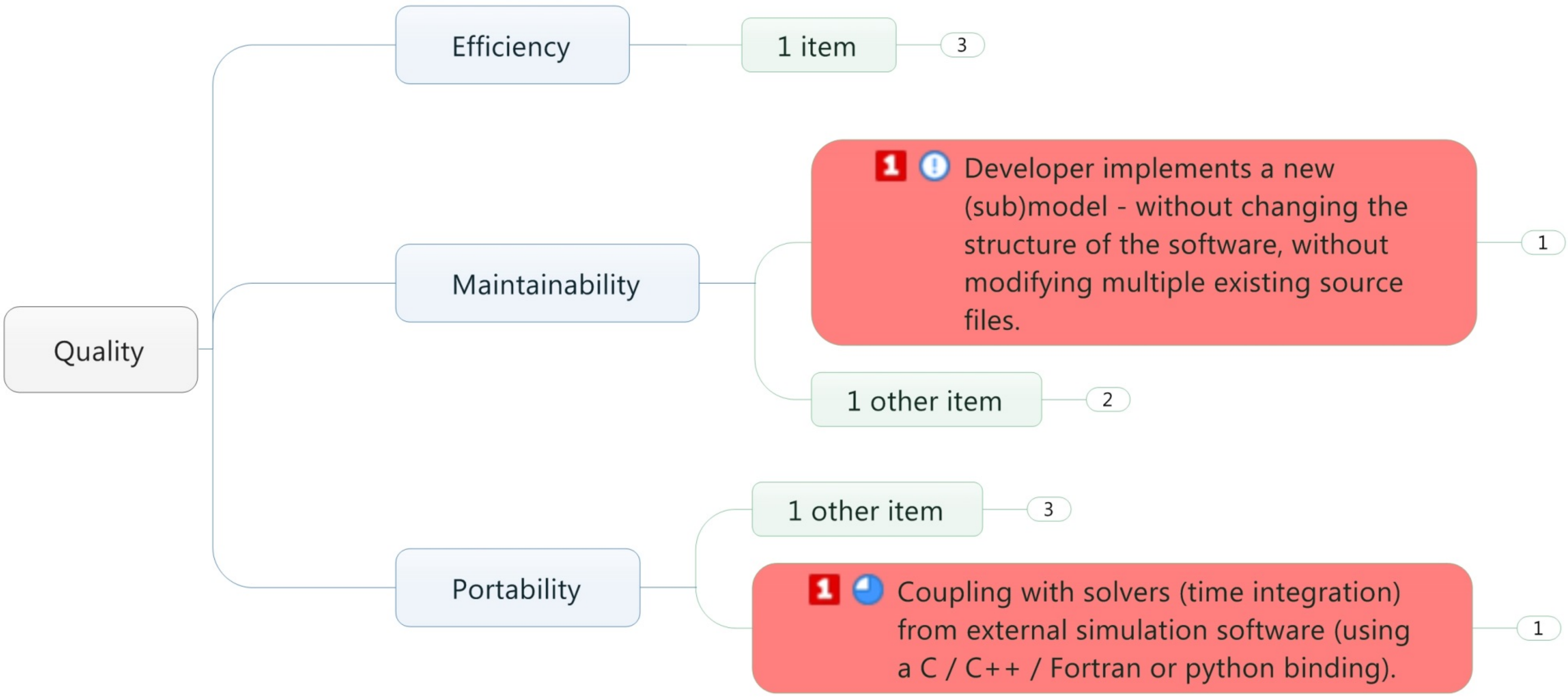
Goals

- Overcome shortcomings in current (DLR) software:
  - Simulate free flying helicopters
  - Allow arbitrary helicopter / wind turbine configurations
  - Focus on dynamic behavior
- Complexity:
  Consider **interdependencies of components** as elaborately as possible
  (→ simulate dynamic behavior correctly)
- Modularity:
  Allow **interchanging components** of the entire model
  (→ allow simulations of different fidelity)

Utility tree 1 (Qualitätsbaum)



Utility tree 2 (Qualitätsbaum)



Generic multi-model approach

- "Sub"-model i in state-space form:

$$\dot{x}_i = f(x_i, u_i, t)$$
$$y_i = g_i(x_i, u_i, t)$$

  - x_i state vector
  - y_i output vector, u_i input vector

- Goal:

  - Strong coupling in time (mathematic view)

  - Weak coupling of components (software view)

Abstract model interface

AbstractModel		
attributes:		
inputDescription	: string[]	
outputDescription	: string[]	
methods:		
calculate_time_derivative	(state, input, time)	// $f_i(x_i, u_i, t)$
calculate_output	(state, input, time)	// $g_i(x_i, u_i, t)$
calculate_output_grad	(state, input, time)	// $\frac{\partial g_i}{\partial u_i}(x_i, u_i, t)$

Abstract model interface (2)

- General definition of a **model in state-space form**
    - Inputs / outputs described by strings
      → inputs defined by matching outputs of other models

- No internal state ("state" is a parameter!)

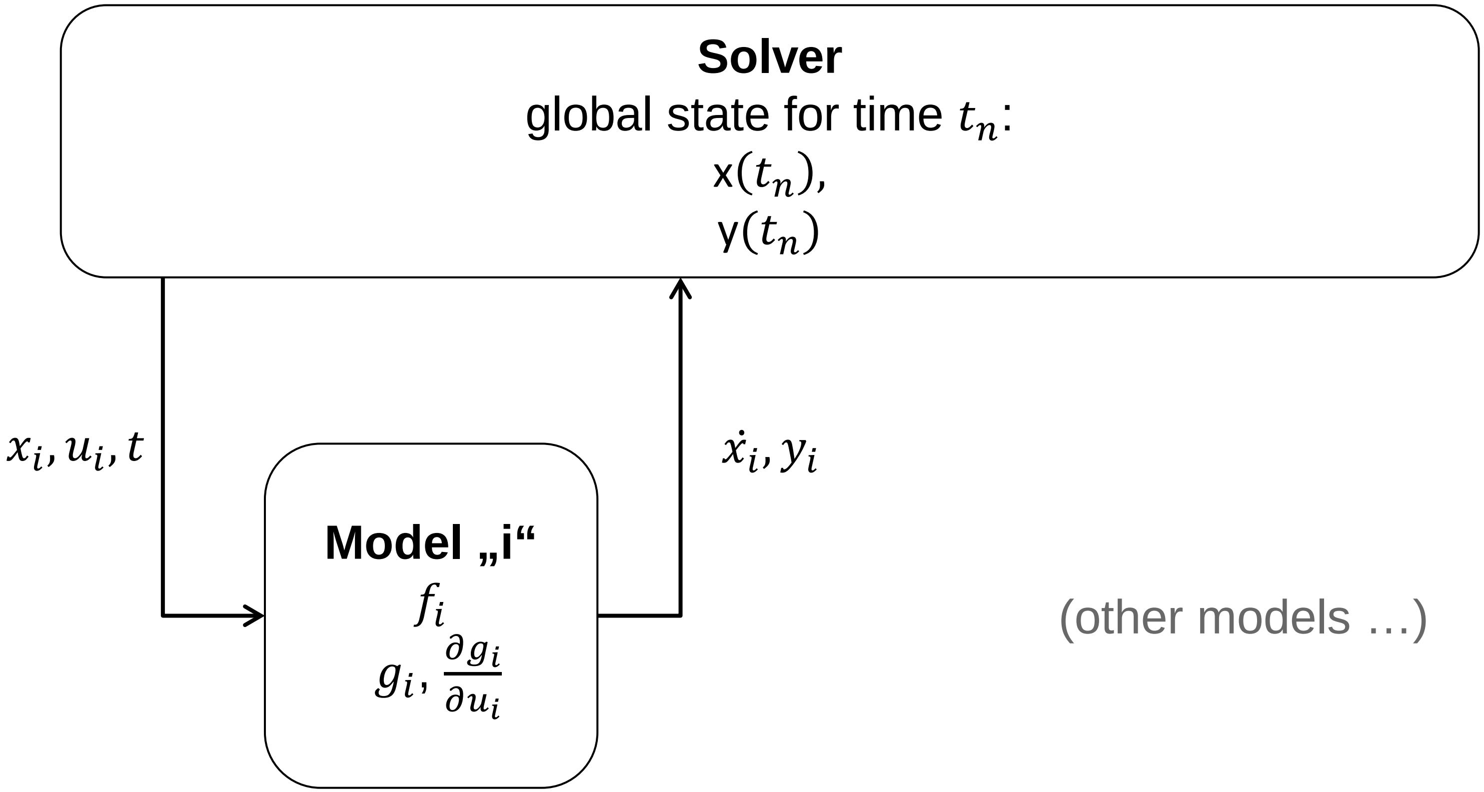
- Simple interface
    - usable from different languages
      Fortran, C, C++, Python, Matlab?, …
    - easy to wrap existing code

DLR

Abstract solver interface

- "Solver" in VAST:
  - Numerical algorithm that works on the system of all models
    (time integration, stability analysis, model testing)
  - Controls the actual "state"
- **Puppeteer design pattern**
  - "Solver" manages interactions between models
    (reduces code complexity to $O(n_{models})$ from $O(n_{models}^2)$)
  - Allows arbitrary model interdependencies

Solver - model interaction

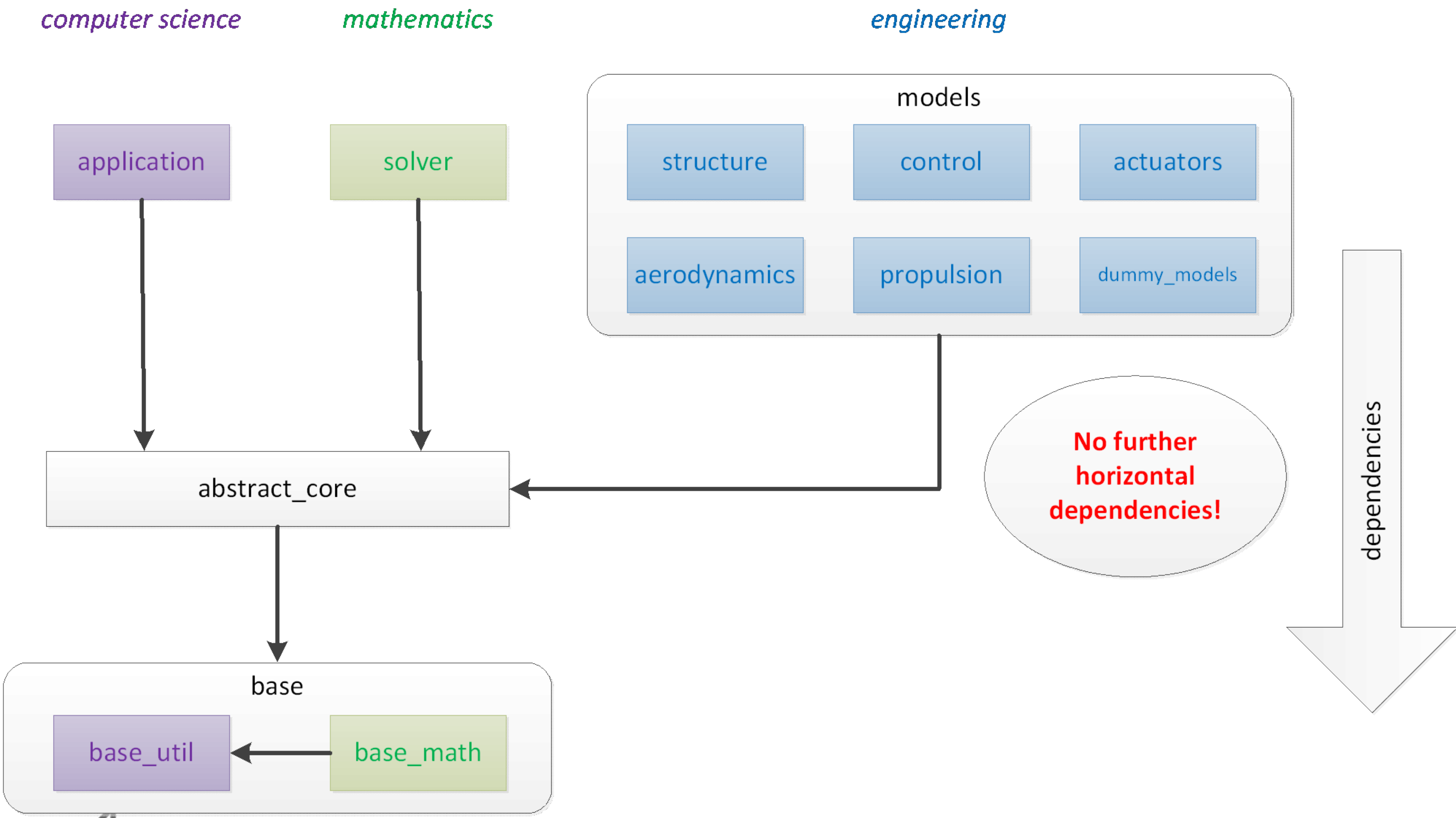


Multi-model approach: advantages

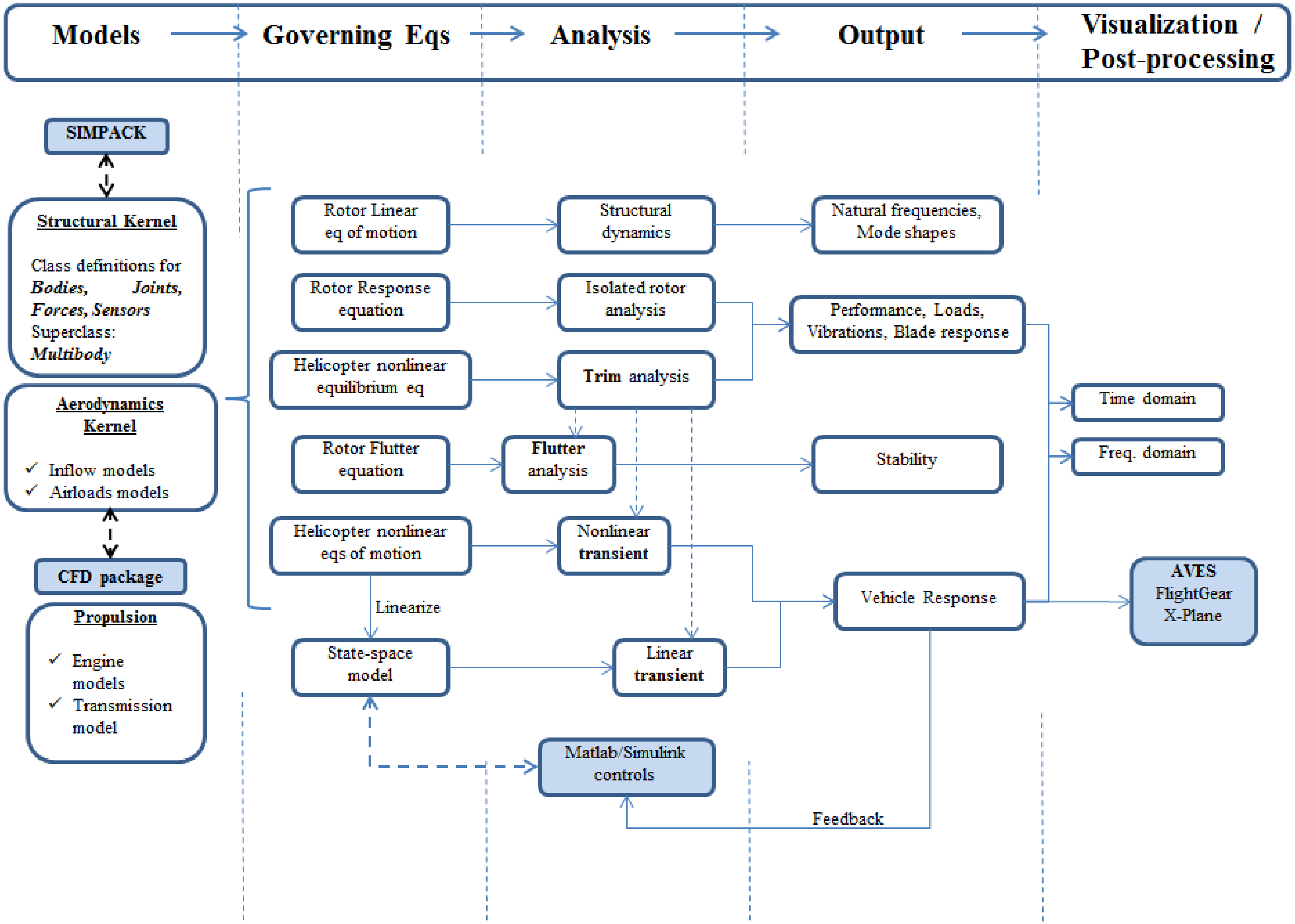
- Dynamic & arbitrary model interdependencies
- Time-dependent data (“state”) controlled at a central place (by the “solver”)
  - **No hidden data flows**
    → Maintainability
  - Allows check-pointing / restart, jumping in time, …
- No “dependencies in code”
  - Allows isolated tests of models / solvers

tight coupling (numerics) ↔ loose coupling (software)

Package dependencies



Ideas on functional architecture (engineering view)



Open questions

- Intuitive **GUI**:
  - Helps building / editing complex multi-model systems
    (2 rotors → 2 rotor models → 1 global / 2 local aerodynamic models, …)
  - Based on the VAST XML config schema file?
    (same technology as in CPACS)

- Integration of **mixed math/engineering algorithms** (“Trim”, stability analysis):
  - Currently we can do “time integration”
  - Strict distinction between maths and engineering
  - “Trim”: inverse simulation, “find input for specific periodic output”
    (auto-pilot controller “model”, general mathematical “solver”?)

DLR