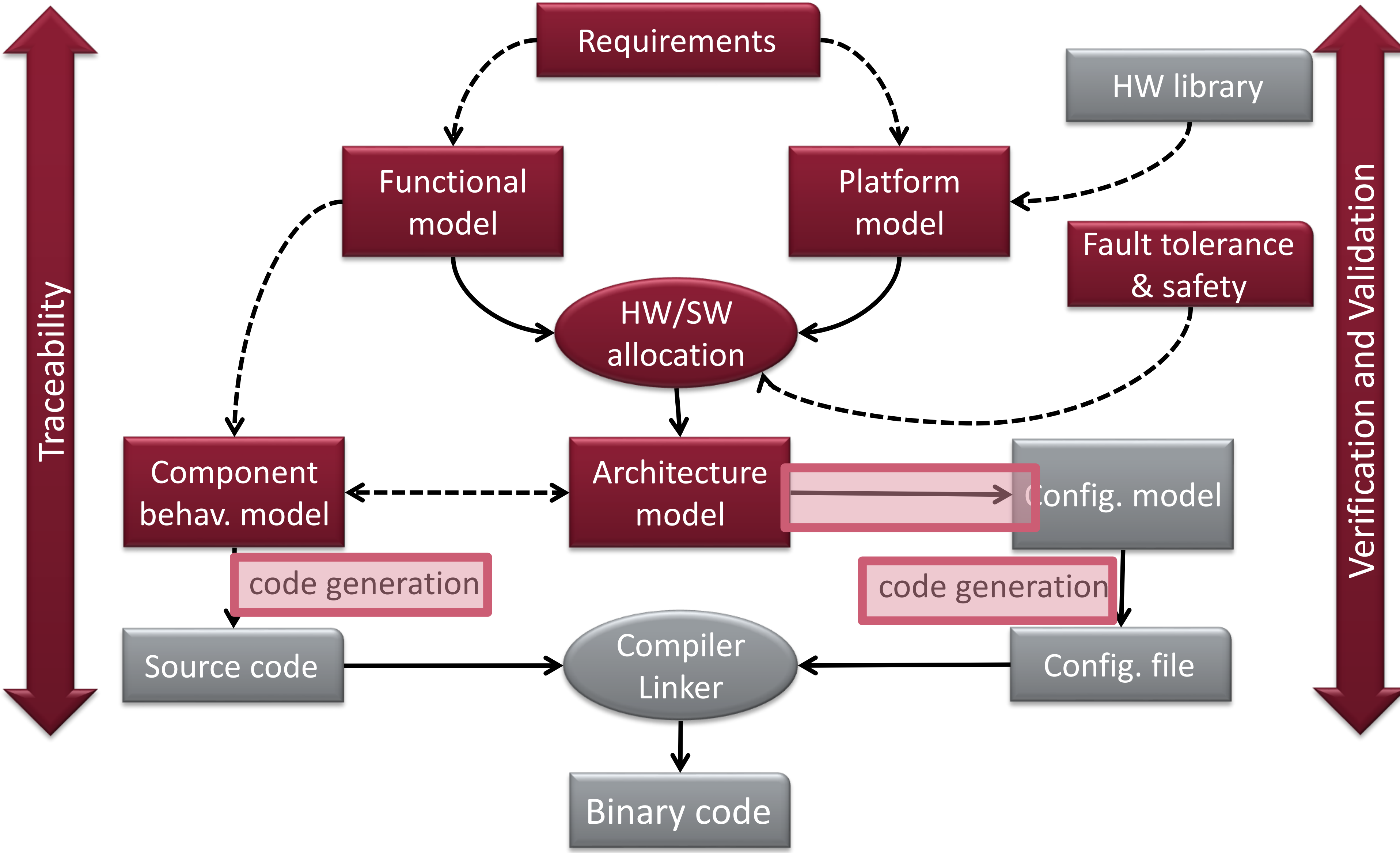
Towards Model-driven Engineering

Systems Engineering BSc Course

Platform-based systems design



Learning Objectives

Model and code generation

- Motivations
- Overview on code generation concepts

Domain-specific modeling

- Motivation and core concepts

Case study

- Complex case study from the avionics domain

Model-driven Engineering

Motivations

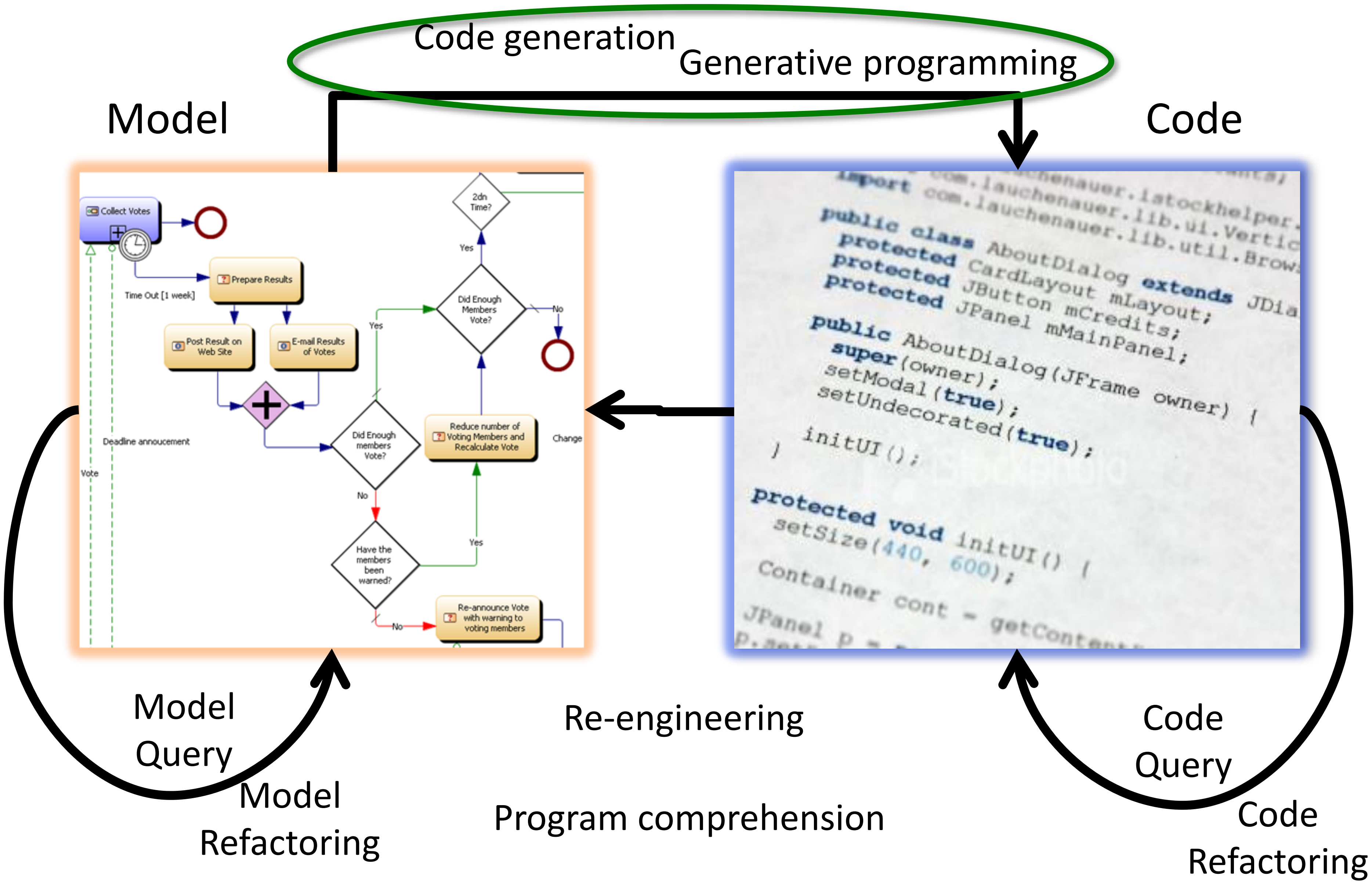
- We have valuable information in models → reuse!
  - Use our **models/requirements/plans** to derive…
    - Documentation
    - Source code
    - Configuration, communication descriptors
    - …
    - Even other models!

Model-to-text transformation (M2T)

Model-to-model transformation (M2M)

- **Model-driven Engineering:**
  - Models are the main artifacts, not code etc.
  - The rest is mostly derived / generated
  - May shorten development time and increase quality

Some Well-known MDSE Concepts

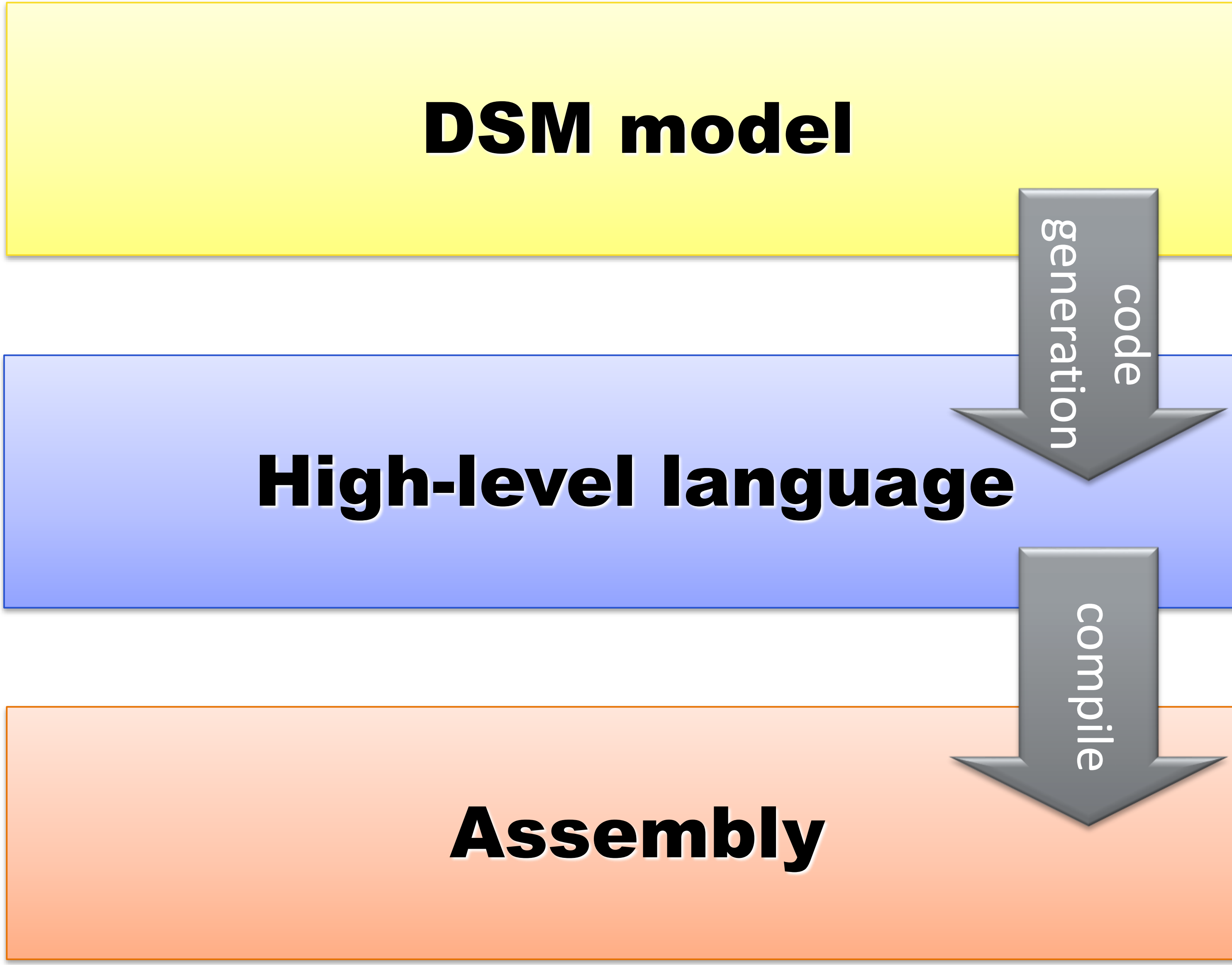


Code generation (text synthesis, M2T)

Similarity with compilers

- Mapping between abstraction levels
  - e.g., from C to assembly
- Usage of design patterns
  - e.g., arrays, function calls, loops in C
- Many similarities, NOT a strict separation
  - pl. C++ templates, automatically generated ctor+dtor
- Prediction:
  - yesterday’s design pattern → today’s code generation feature → tomorrow’s language element
- Domain-specific instead of universal languages

Example: Source Code generation in MDE



Code Generation - Major Approaches

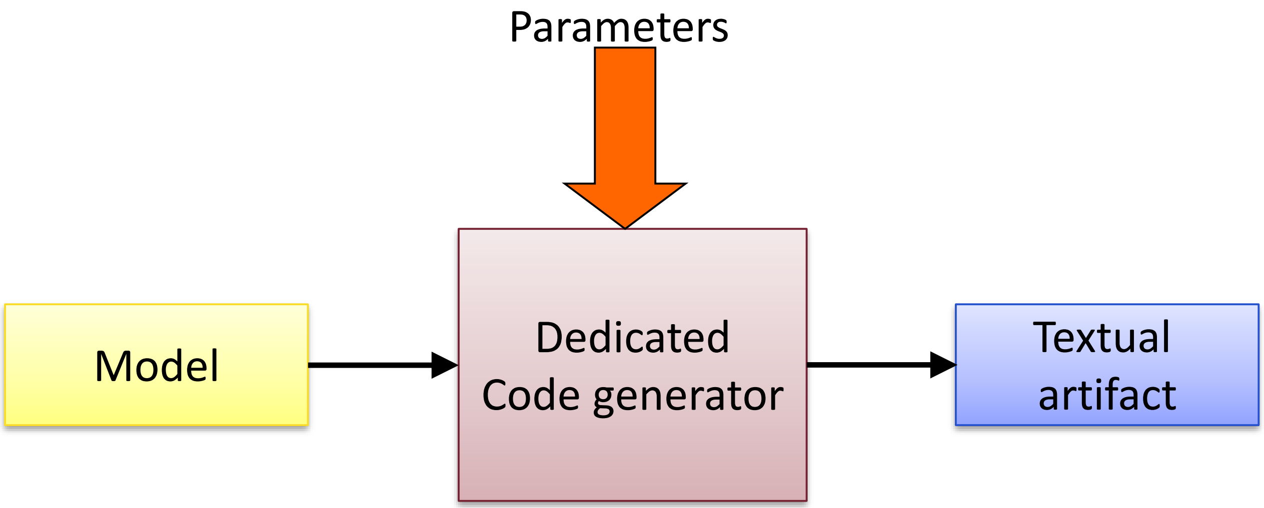
- Dediacated
  - Specific, ad-hoc
  - Using a dedicated code generator
- Template based

Specific, ad-hoc

```
sourceFile.write("    temp = ((AIDA_PARTITION_TYPE*) selfModule.partitions.elements);\n" )
i = 0
for partition in partitions:
 numPorts = getNumberOfAllCommPorts_Partition(currModuleComm, interPartitionComm, partition.partitionName)
 sourceFile.write("    temp[" + str(i) + "].partition_id = " + str(partition.partitionID) + ";\n" )
 sourceFile.write("    strcpy( &temp[" + str(i) + "].partition_name[0], \"" + str(partition.partitionName) + "\");\n")
 sourceFile.write("    temp[" + str(i) + "].ports.type = CONST_AIDA_PORTS_TYPE;\n")
 sourceFile.write("    temp[" + str(i) + "].ports.elements = &mem_ports_" + str(partition.partitionName) + "[0];\n")
 sourceFile.write("    temp[" + str(i) + "].ports.numOfElements = " + str(numPorts) + ";\n")
 sourceFile.write("\n")
 i = i + 1
## end for
sourceFile.write("\n")
```

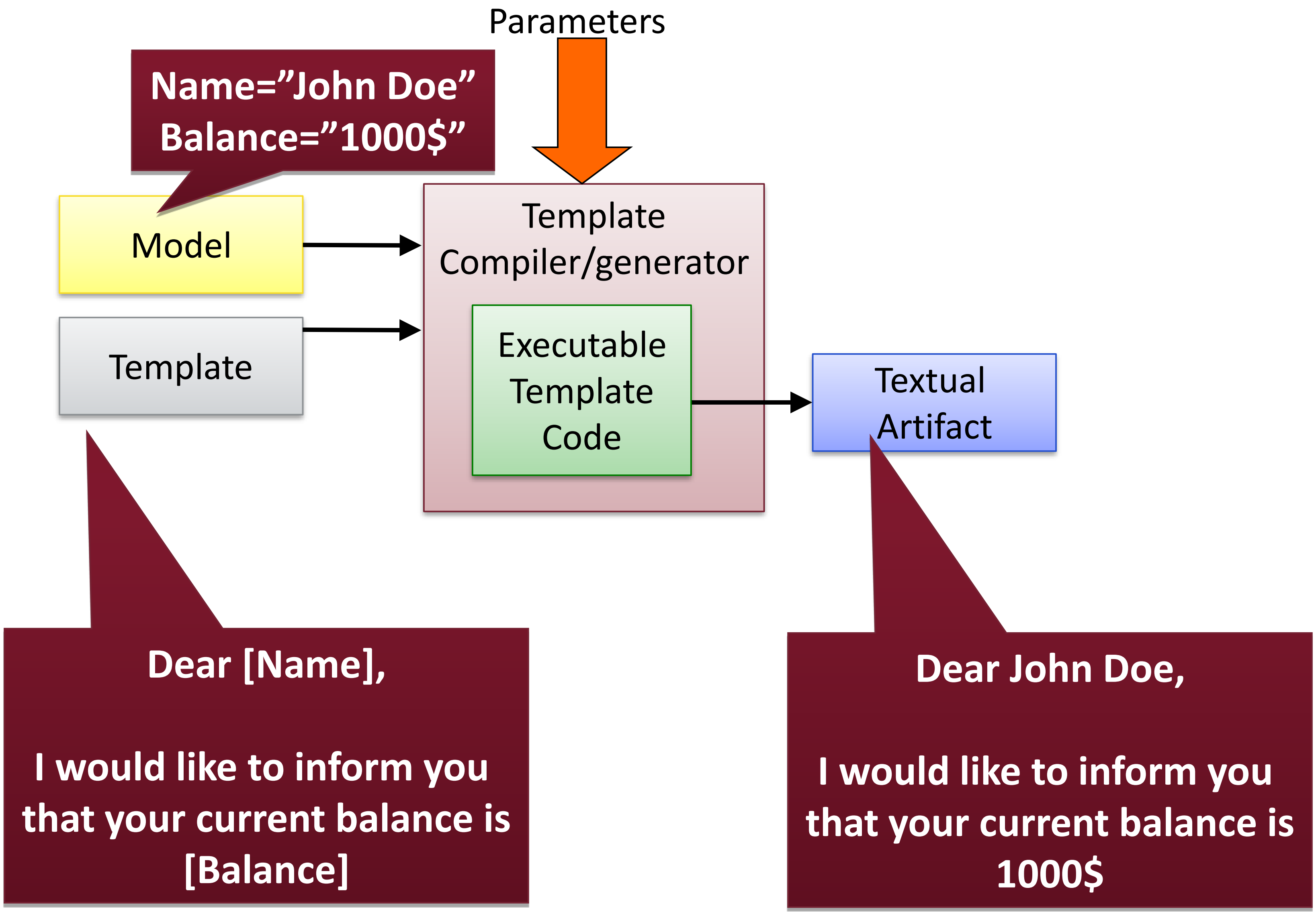
- Designed for the specific problem domain:
  - Best performance
  - Quick and dirty
  - Long development, hard maintainability
  - Zero reusability
  - Dedicated problem domains
    - Minimal changes during support cycle (safety critical embedded system, defense)
    - Certifiability
  - Example:
    - ARINC653 Multistatic configuration generator (python script)

Dedicated code generator

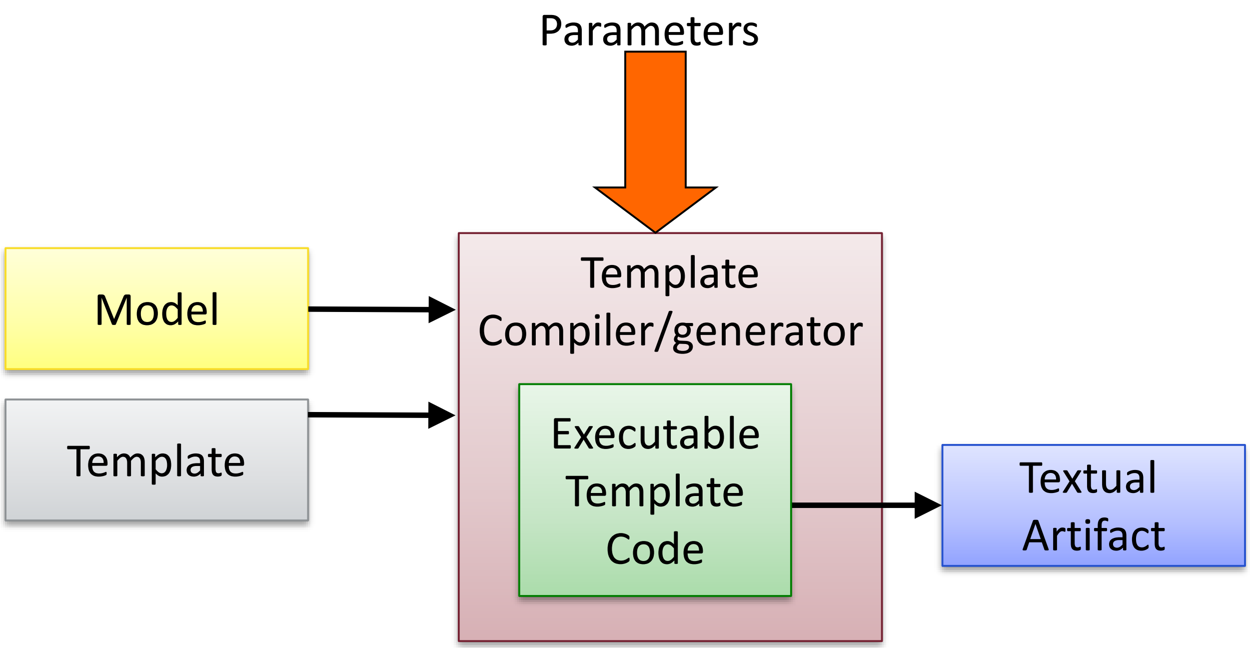


- Examples:
  - IBM Rational Software Architect
  - VASP (DO-178B Level A) Display graphics in avionics
  - Mathworks
  - Matlab Simulink
  - Esterel Scade suite
  - Yakindu Statechart Tools ☺

Template based approach



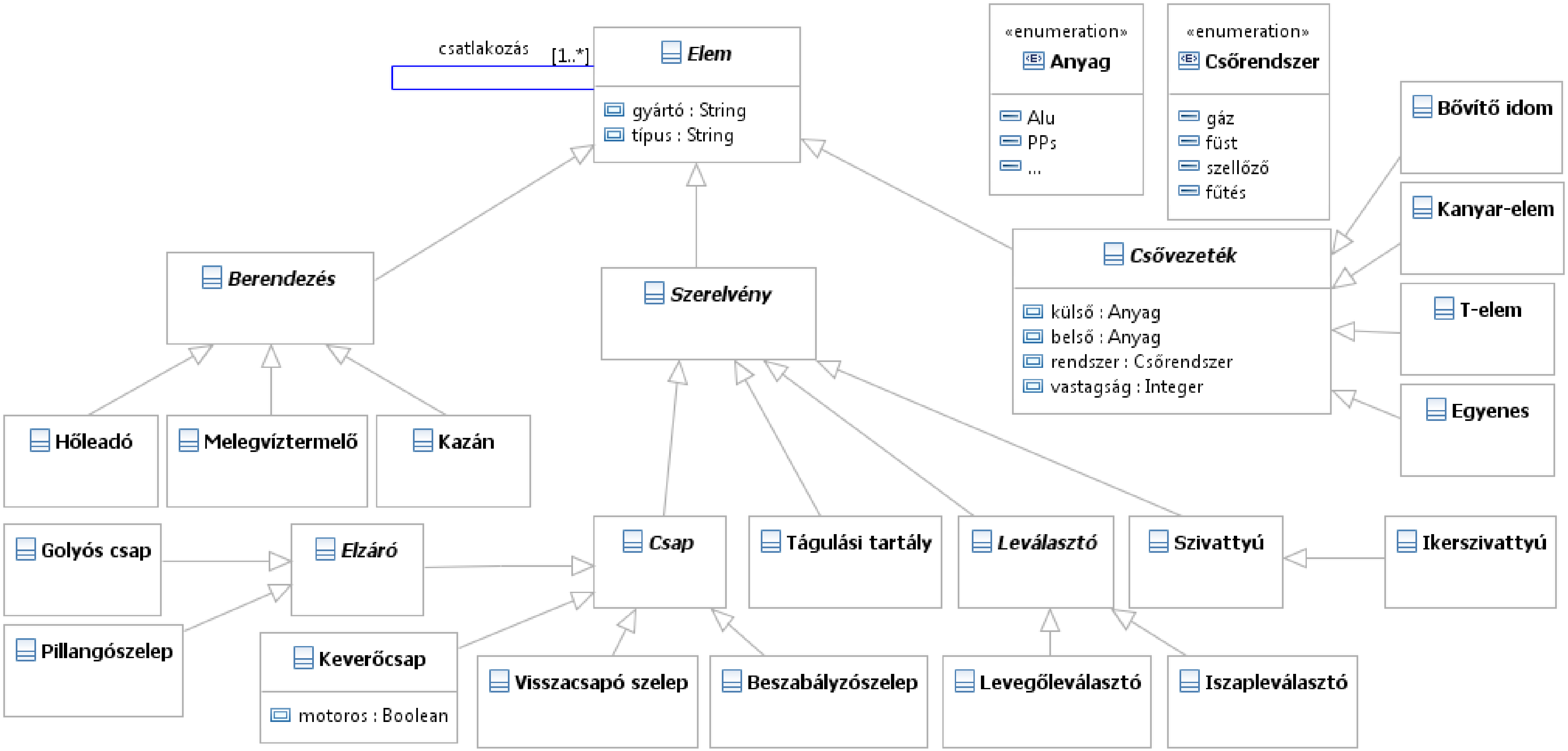
Template based approach



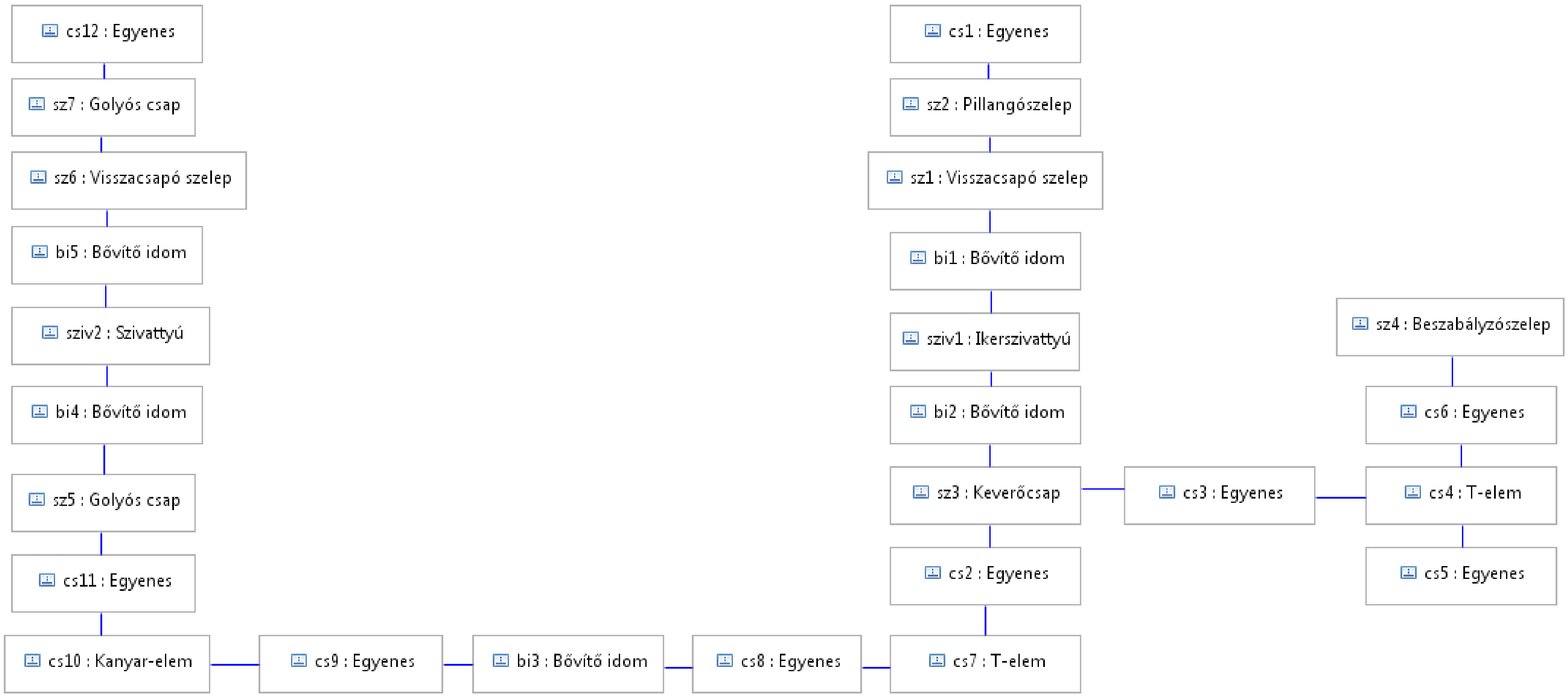
- Examples:
  - JET (for EMF models)
  - Velocity (/JSP)
  - Xtend, Acceleo (MDE approach in Eclipse)
  - AutoFilter (Kalman filters)
  - Smarty (php)

Domain-specific Modeling Languages

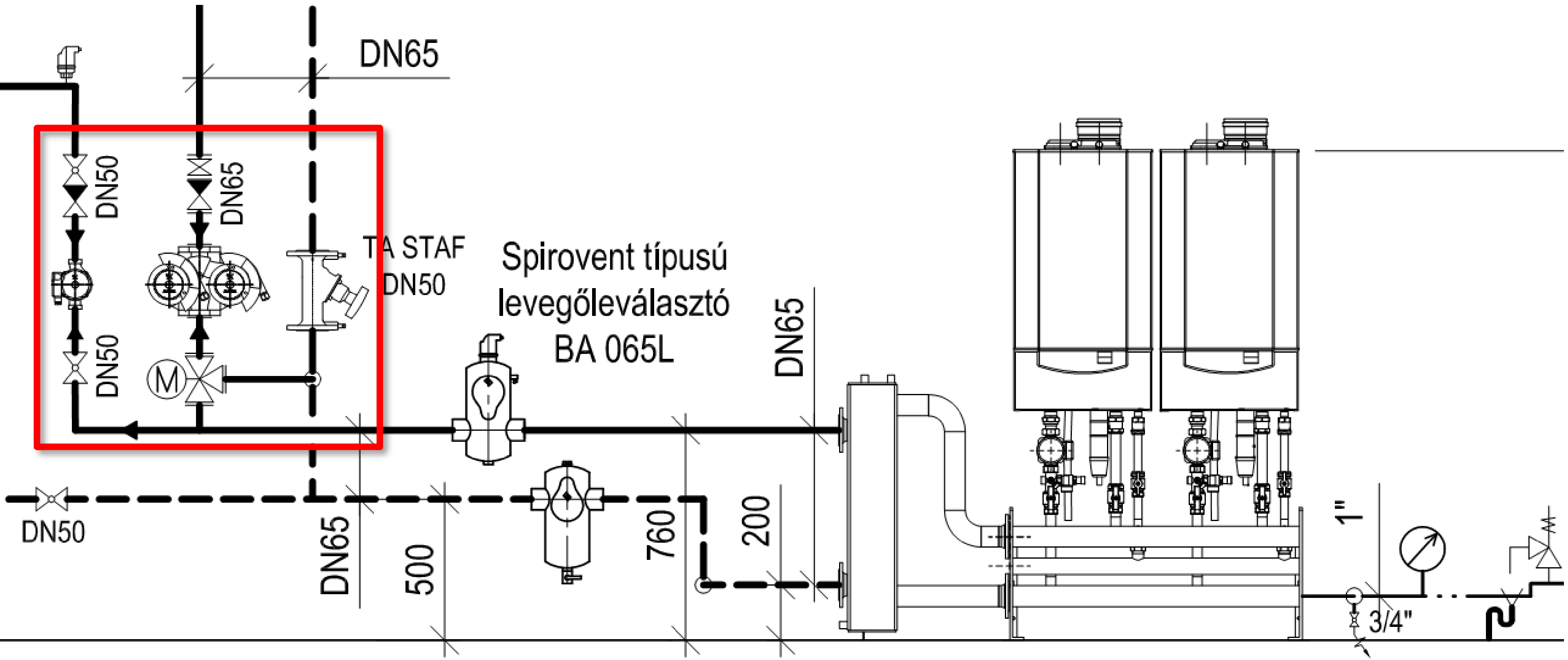
Example metamodel / profile



Instance model, abstract syntax



Instance model, concrete syntax

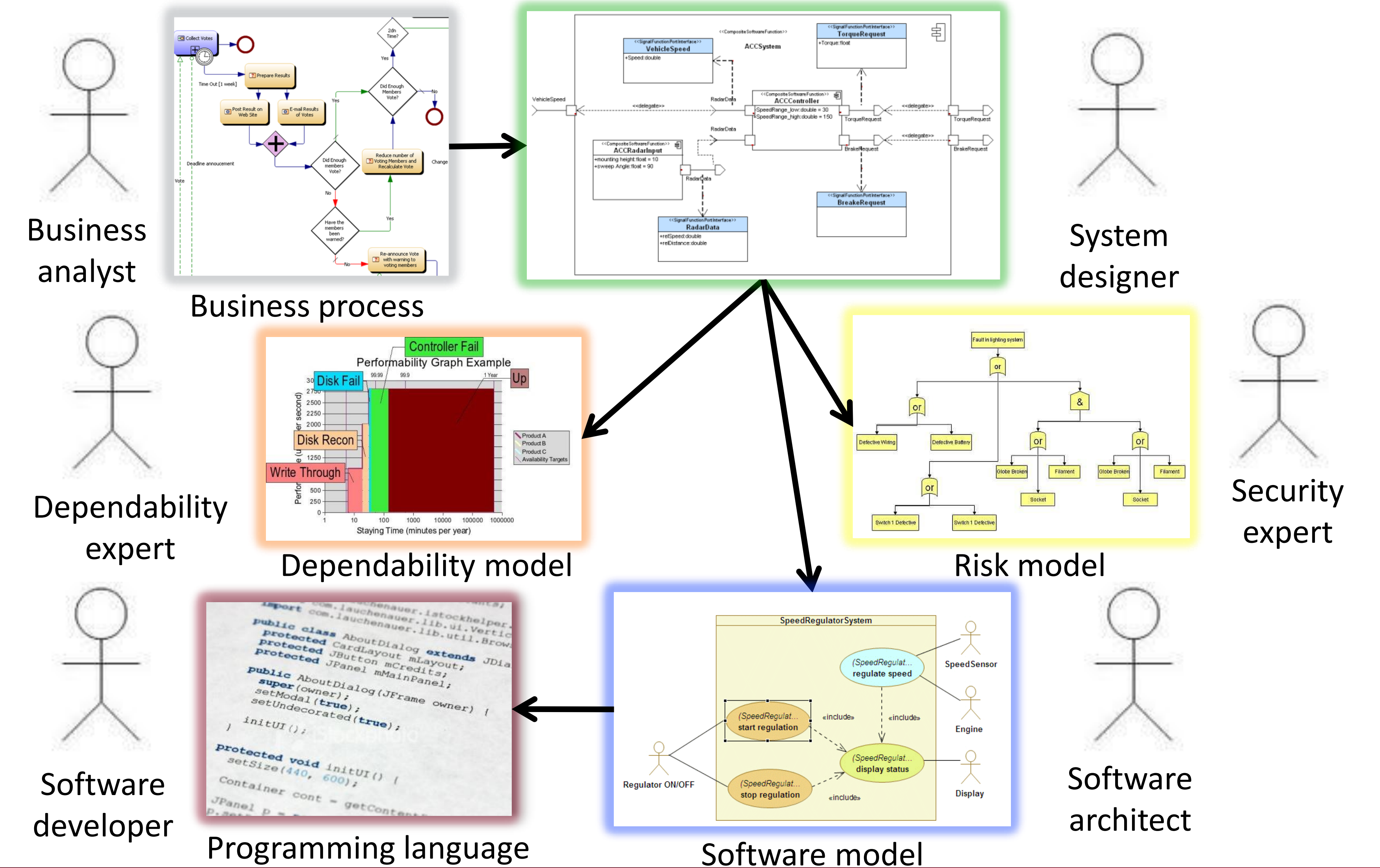


Honeywell
keverőcsap
DN50 K_{vs} 40

Spirovent típusú
iszapleválasztó
BE 065L

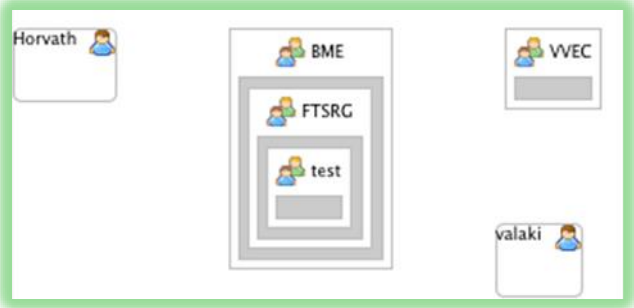
Remeha Quinta kaszkád
rendszer hidraulikus váltóval

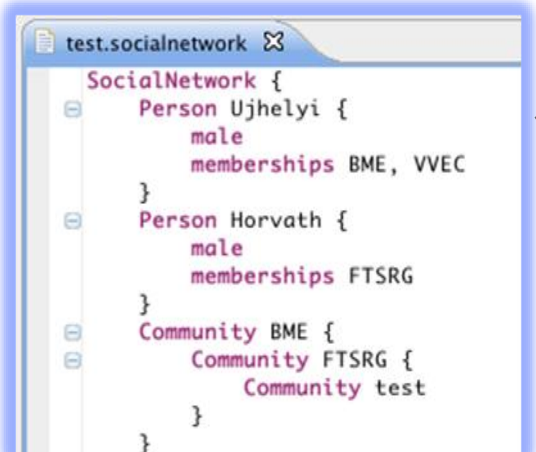
Domain specific modeling languages



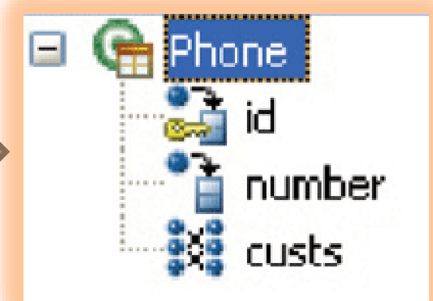
Structure of DSMs

Concrete syntax (Graphical/Textual)



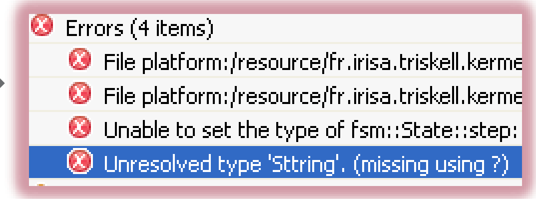


Abstract syntax (Metamodel)



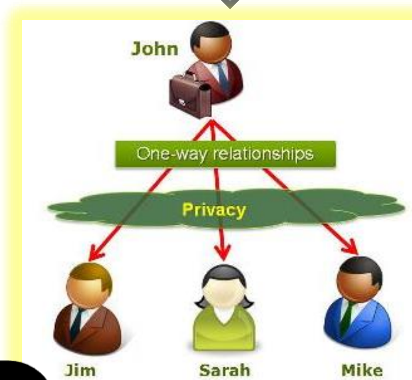
Mapping

Well-formedness constraints

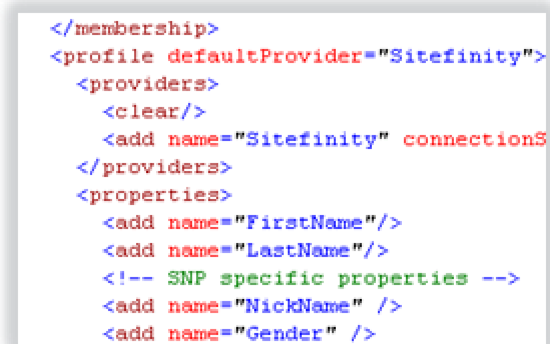


Behavioural semantics, Simulation, Refactoring

Code generation



View



Source Code (Documentation, Configuration file)

Foundations of many modern tools (design, analysis, V&V)

- Domains: avionics, automotive, business modeling, …

Aspects of Defining DSMLs

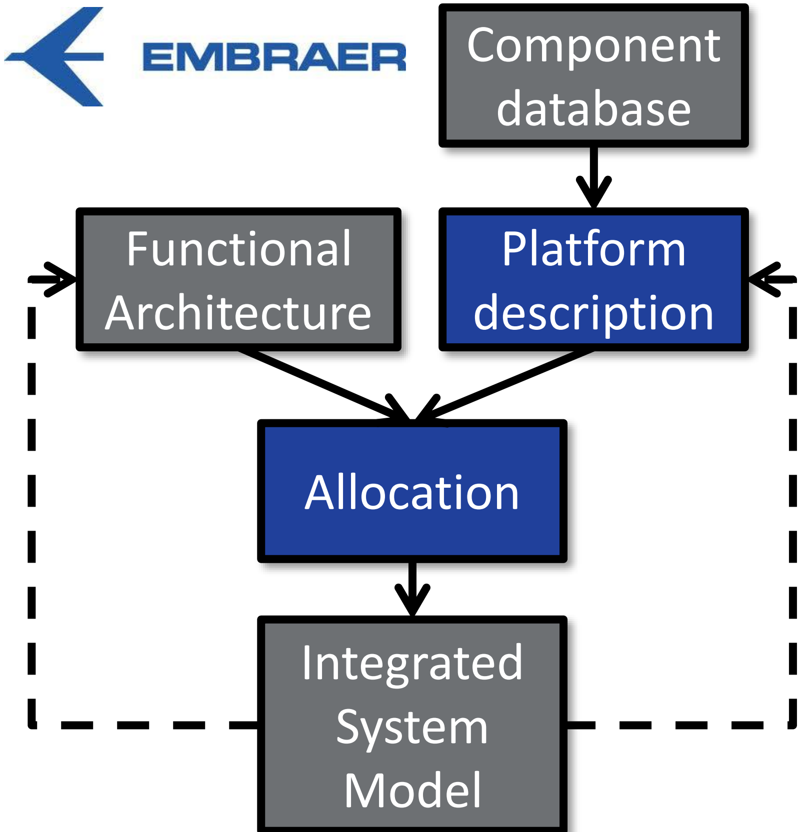
Model driven development of ARINC653 configuration tables

A case study

Recent Project

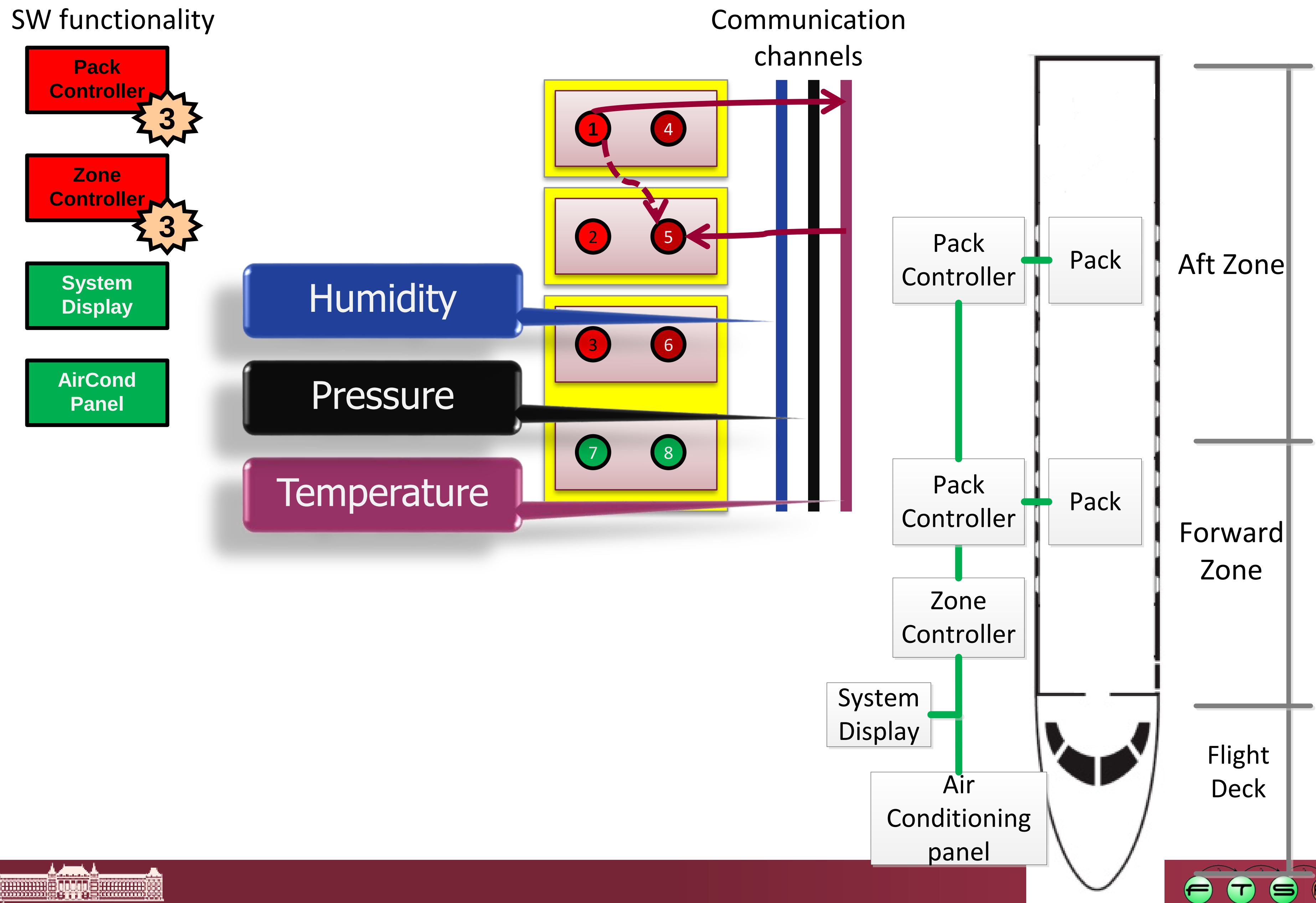
Goal: Allocate SW components to ARINC653 compliant IMA platform

EMBRAER



DECOS indexys MOGENTES
DIANA SENSORIA secure CHANGE

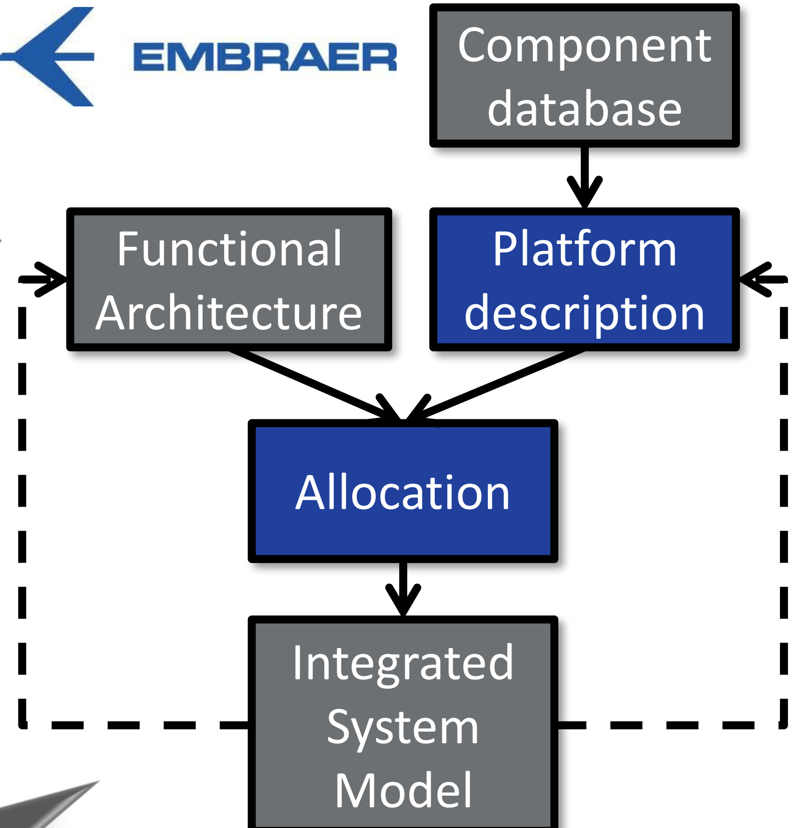
Allocating communication channels



Model Driven Development of IMA Configs

Inputs:
- Platform Independent Model (PIM) (functional + nonfunc. reqs; Simulink)
- Platform Description Model (PDM) for ARINC 653 (DSML)

EMBRAER

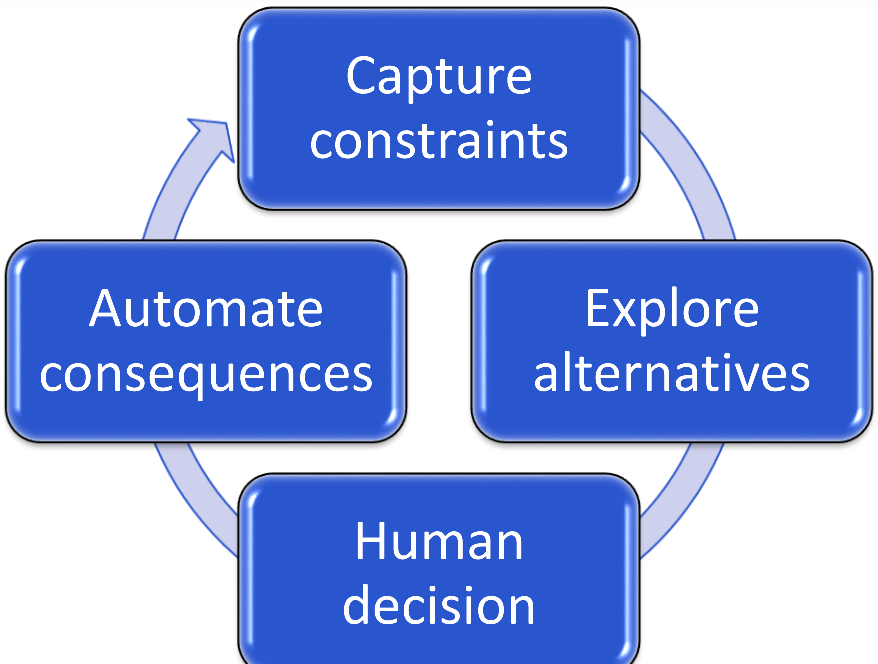


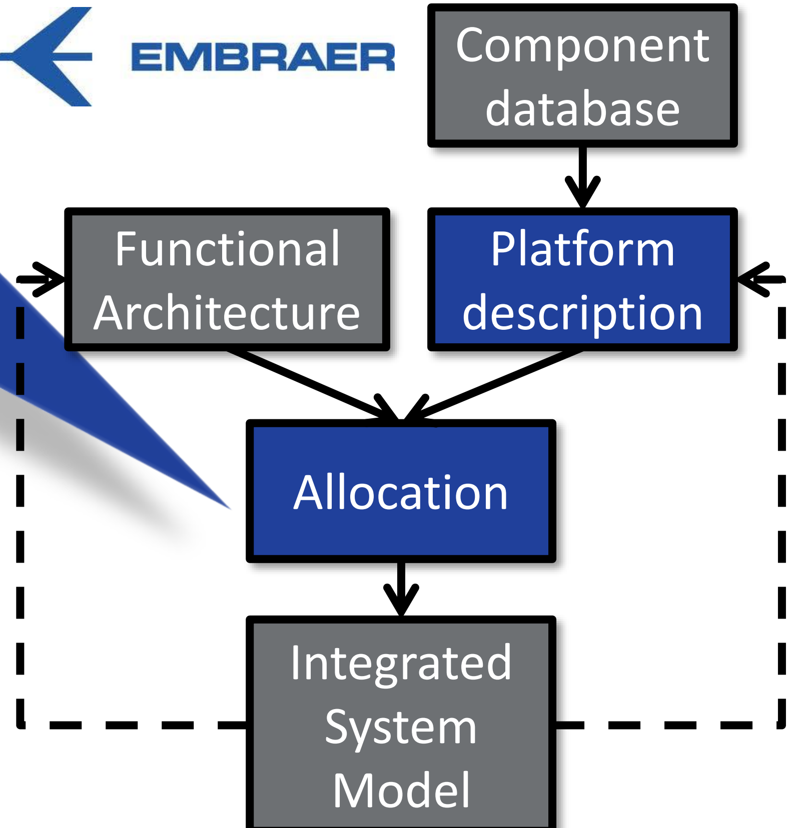
Output:
- Integrated system model
- Ready for simulation
- End-to-end traceability

Model Driven Development of IMA Configs

Model transformation chains:

- Designer-guided manual steps
- Automated steps
  - design space exploration
  - optimization
  - code generators
- Continuous validation of design rules



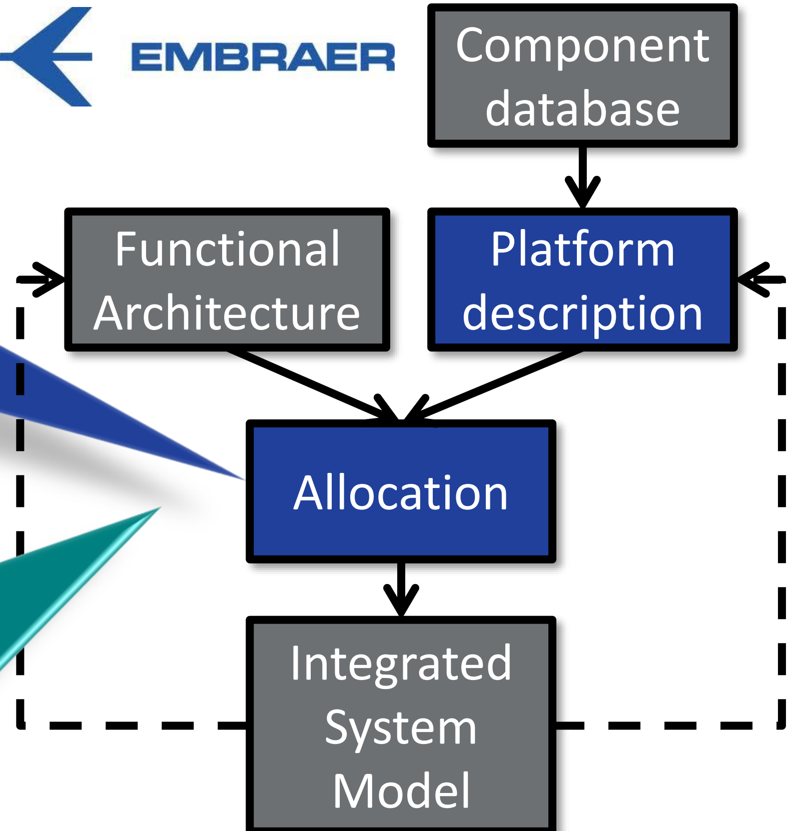


Model Driven Development of IMA Configs

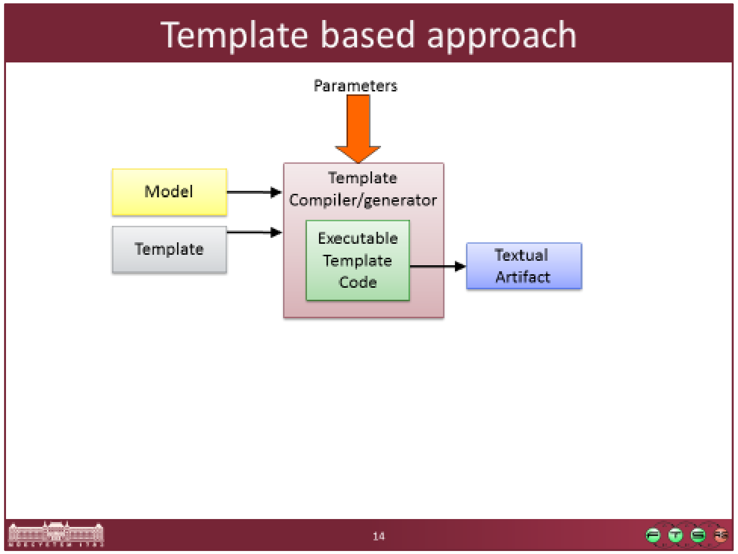
Precise development workflow:
- Aligned with certification-compliant development process
- Monitors design phases
  - completed steps
  - incomplete steps

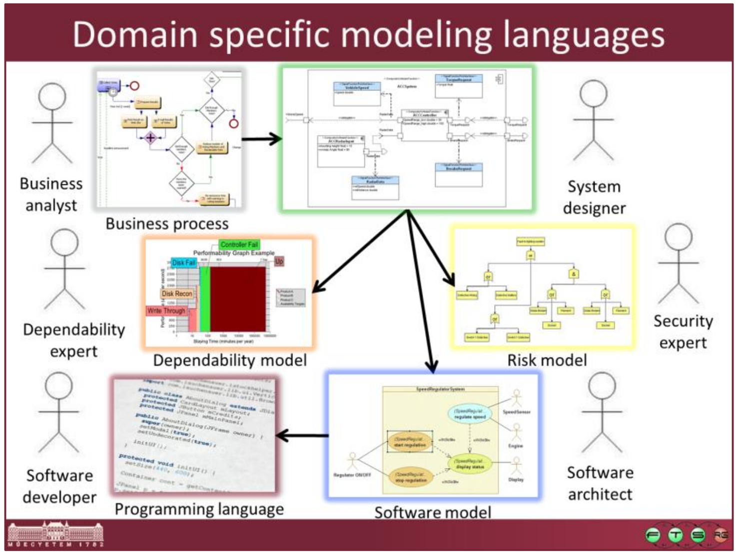
End-to-end traceability:
- Traceability models
  - linking FAM and PDM to IAM
  - integration with requirements tool (e.g. DOORS)
- Soft interconnection of models by incremental model queries

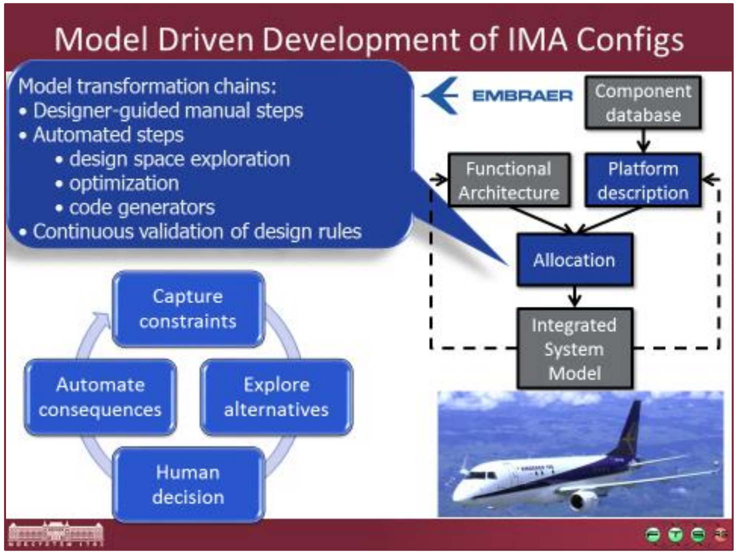
EMBRAER

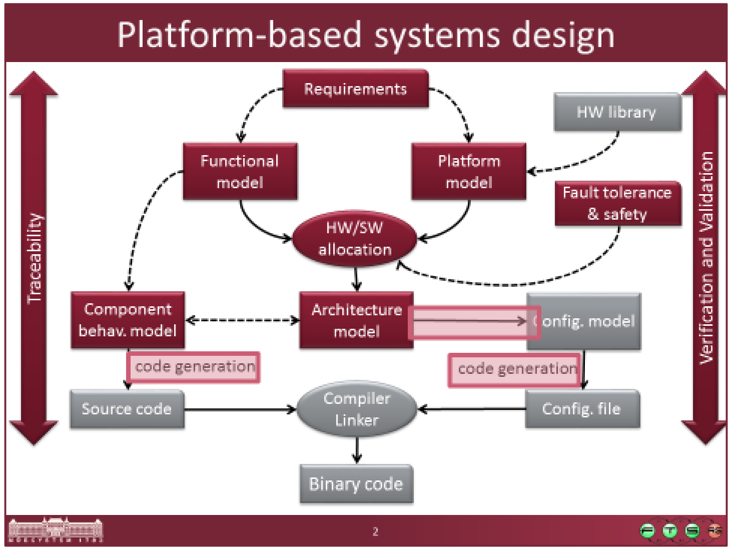


Summary


Template based approach
Parameters
Model
Template
Template Compiler/generator
Executable Template Code
Textual Artifact
14


Domain specific modeling languages
Business analyst
Business process
System designer
Dependability expert
Dependability model
Risk model
Security expert
Software developer
Programming language
Software model
Software architect


Model Driven Development of IMA Configs
Model transformation chains:
• Designer-guided manual steps
• Automated steps
• design space exploration
• optimization
• code generators
• Continuous validation of design rules
EMBRAER
Component database
Functional Architecture
Platform description
Allocation
Integrated System Model
Capture constraints
Automate consequences
Explore alternatives
Human decision


Platform-based systems design
Traceability
Verification and Validation
Requirements
HW library
Functional model
Platform model
Fault tolerance & safety
HW/SW allocation
Component behav. model
Architecture model
Config. model
code generation
code generation
Source code
Compiler Linker
Config. file
Binary code
2

Ez itt a reklám helye!

- „Kritikus rendszerek” MSc főspecializáció
  - Modell alapú rendszertervezés (BMEVIMIMA00)
    - Szakterület-specifikus modellezés
    - Modellező eszközök, kódgenerátor, M2M, stb. fejlesztése
  - Szoftver- és rendszerellenőrzés (BMEVIMIMA01)
    - V&V technikák a statikus kódellenőrzéstől a rendszertesztelésig
    - Tesztgenerálás modell és kód alapján
    - Megbízhatósági analízis
  - Kiberfizikai rendszerek (BMEVIMIMA02)
    - IoT + Cloud + Fog Computing rendszerek tervezése és megvalósítása
    - Kritikus rendszerek tervezése, biztonsági analízise
  - (közös) Formális módszerek (BMEVIMIMA07)
    - Informatikai rendszerek formális modellezése és analízise
  - …