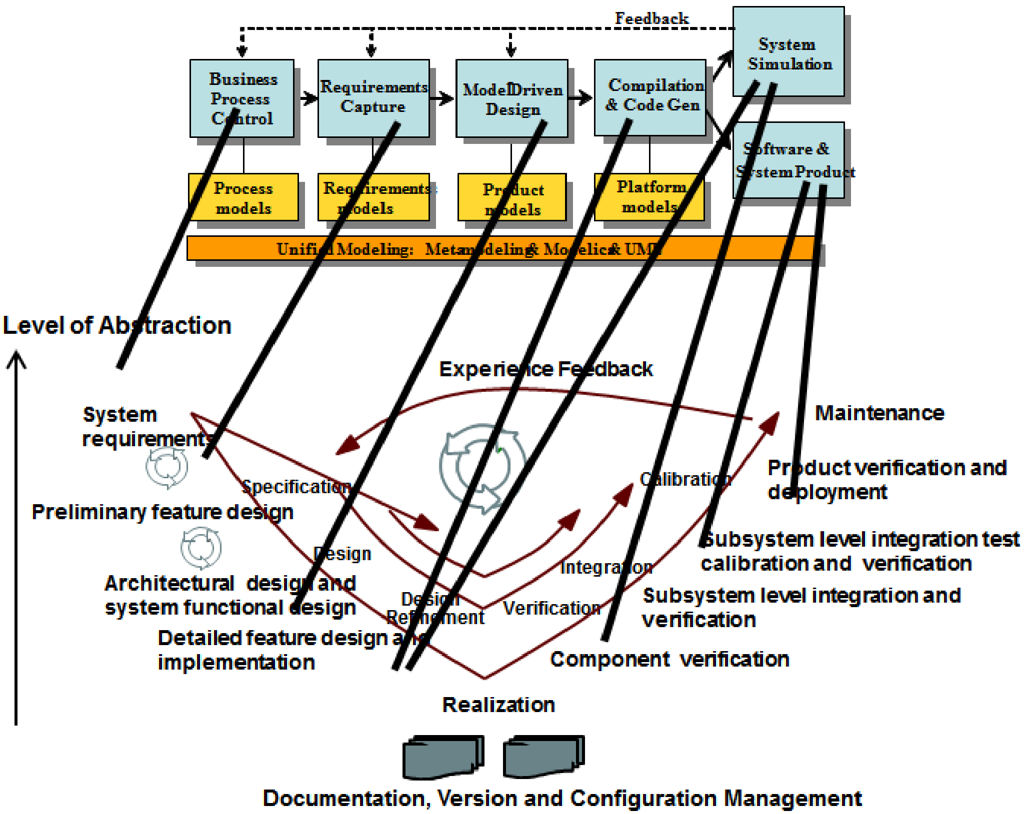
The Need for Comprehensive Whole-life-cycle Systems Engineering Tool Support for Cyber-Physical Systems



MODPROD 2017, Linköping
February 8, 2017

Daniel Bouskela, EDF, France
daniel.bouskela@edf.fr

Peter Fritzson, LIU, Sweden
peter.fritzson@liu.se

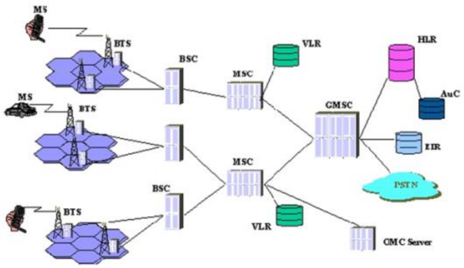
Lena Buffoni, LIU, Sweden
lena.buffoni@liu.se

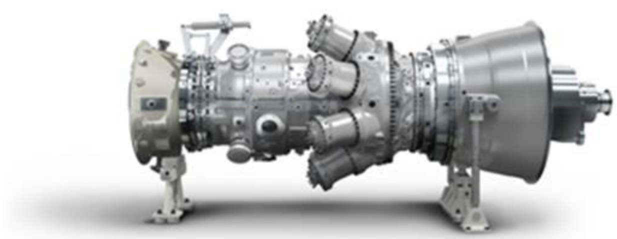
Linköpings universitet

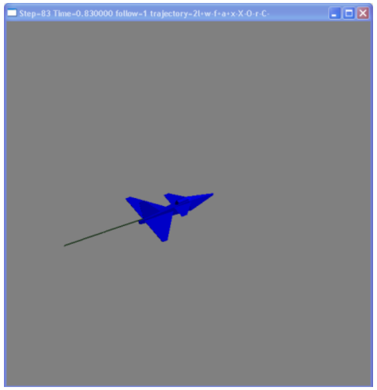
MODELICA

Industrial Challenges for Complex Cyber-Physical System Products of both Software and Hardware

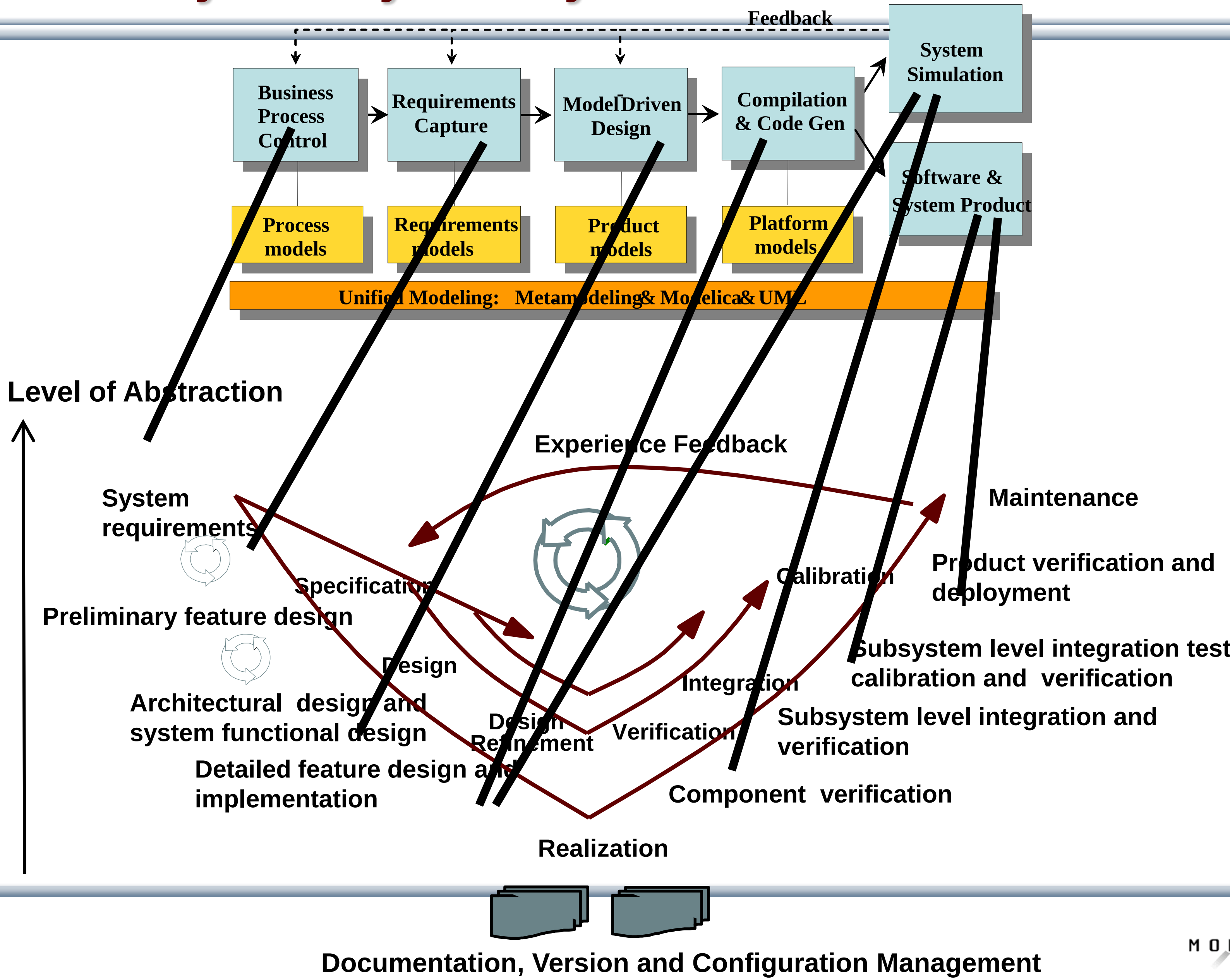
- Increased **Software** Fraction
- **Shorter** Time-to-Market
- Higher demands on effective strategic **decision** making
- **Increased** number of **stakeholders** ➔ increased uncertainties on the system and difficulties in reaching a common agreement
- **Cyber-Physical** (CPS) – Cyber (software) Physical (hardware) products







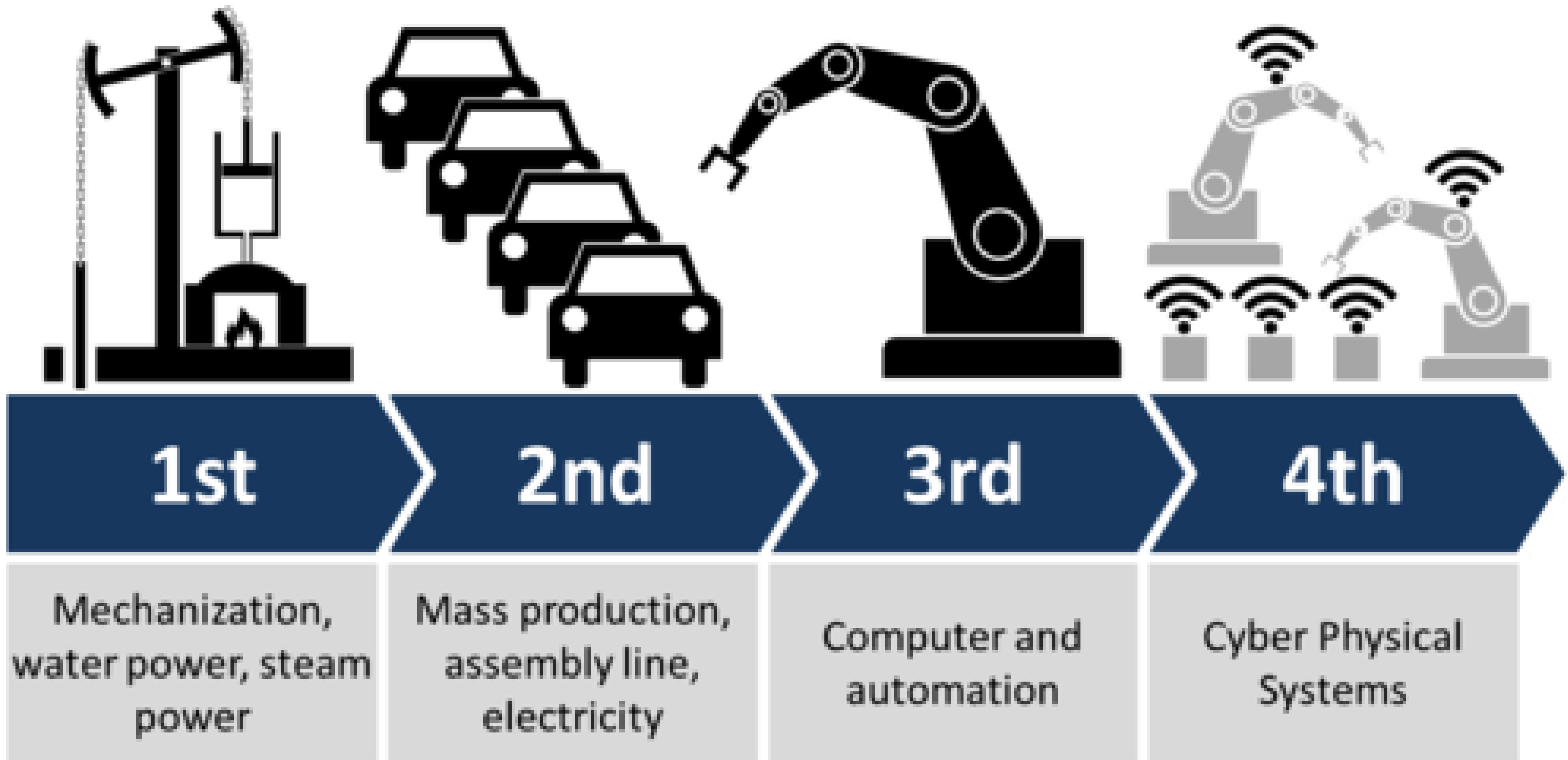
Need for Comprehensive Approach to Systems Engineering of Whole Cyber-Physical Systems Products



Overview of this Talk

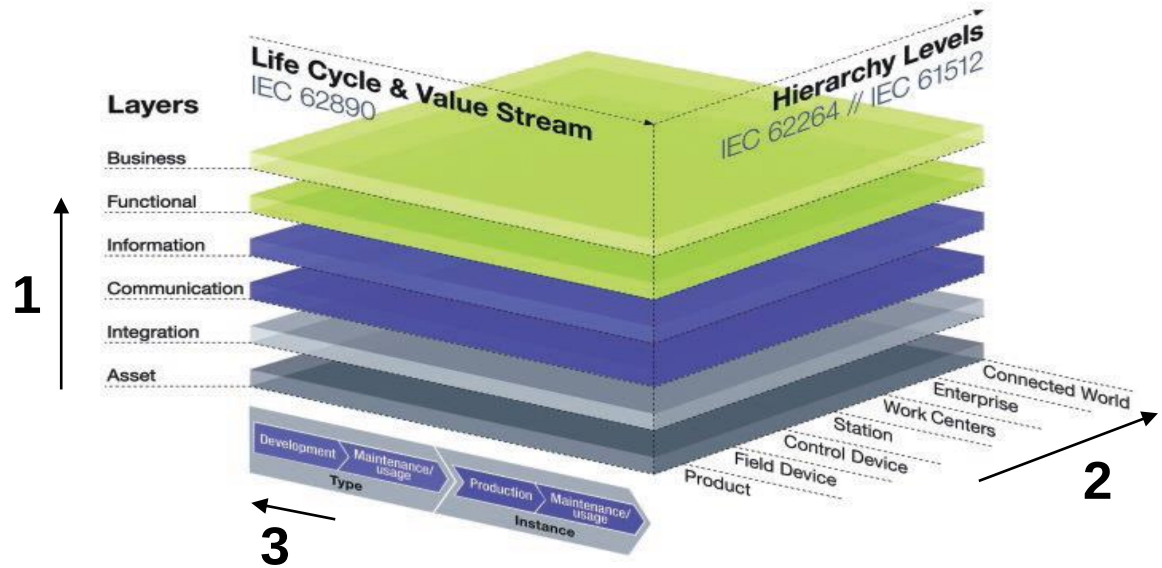
- Need of **whole** product systems engineering
- Taking into account the **assumptions** about the **environment** of the system
- **Industrie 4.0** background
- **Requirement** engineering, debugging and verification
- **Integrating** requirement engineering into CPS systems engineering
- The **whole picture** – what is needed for comprehensive systems engineering

Industrie 4.0 – The Fourth Industrial Revolution
(also called digitalization)



Industrie 4.0 Reference Architecture

- Provides a **global picture** of the engineering needs in terms of MBSE. Only a tiny fraction is achieved today.
- **Missing** in particular:
  1. **Bridging physical** (asset) layer (e.g., Modelica) to the **functional** layer (e.g., SysML) (OPENCPS)
  2. **Modelling**, debugging, **verifying requirements** upstream detailed design (Need for new requirement modeling language?)
  3. Modelling and simulation, large systems with **mode switch**ing (follow-up of MODRIO)



Industrie 4.0 Design Principles

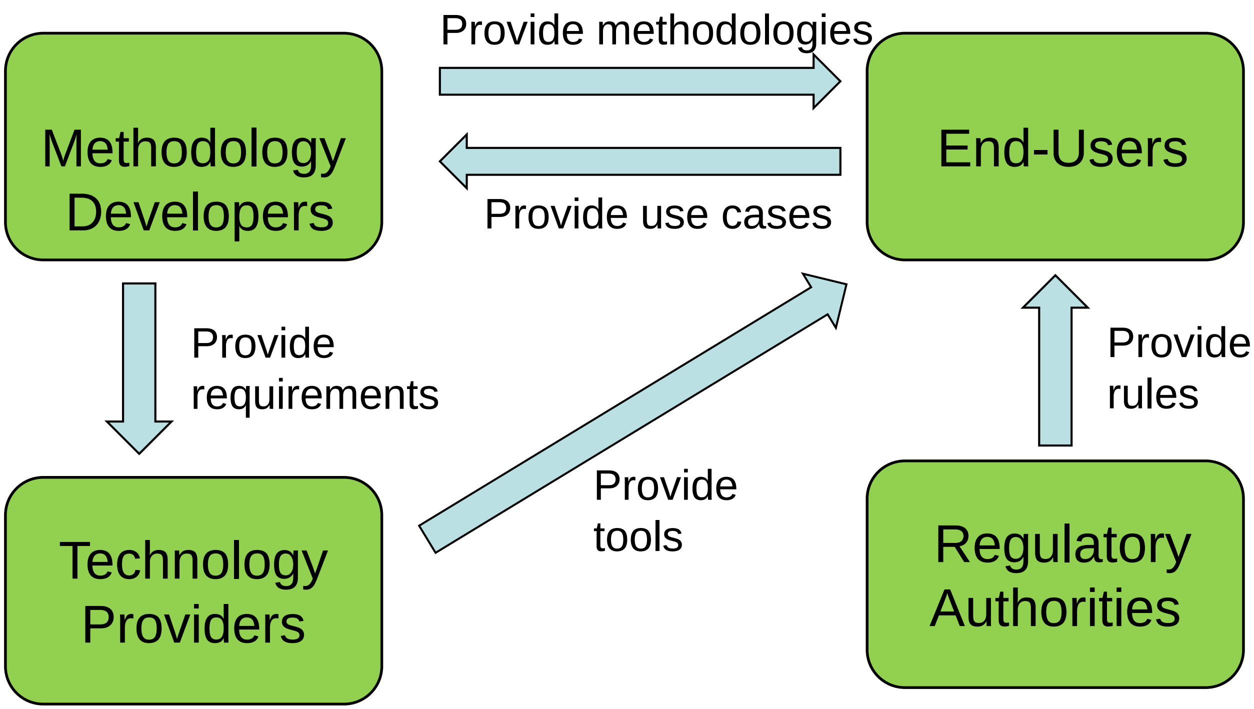
- **Interoperability** – ability of machines, devices, sensors, and people to communicate with each other via the Internet of Things
- Information **transparency** – ability of information systems to create a virtual model of the physical world e.g. by enriching digital plant models with sensor data
- Technical **assistance** – ability of assistance systems to support humans by aggregating and visualizing information comprehensibly for making informed decisions
- **Decentralized** decisions – ability of cyber physical systems to make decisions on their own and to perform their tasks as autonomously as possible.

Some Industrie 4.0 Challenges

- IT **security** issues – aggravated by need to open up previously closed production
- **Reliability** and stability – needed for critical machine-to-machine communication
- **Protect** industrial knowhow – e.g. contained in industrial automation equipment and software
- Lack of adequate skill-sets – needed for the new technology
- Loss of jobs – to automatic and IT-controlled processes

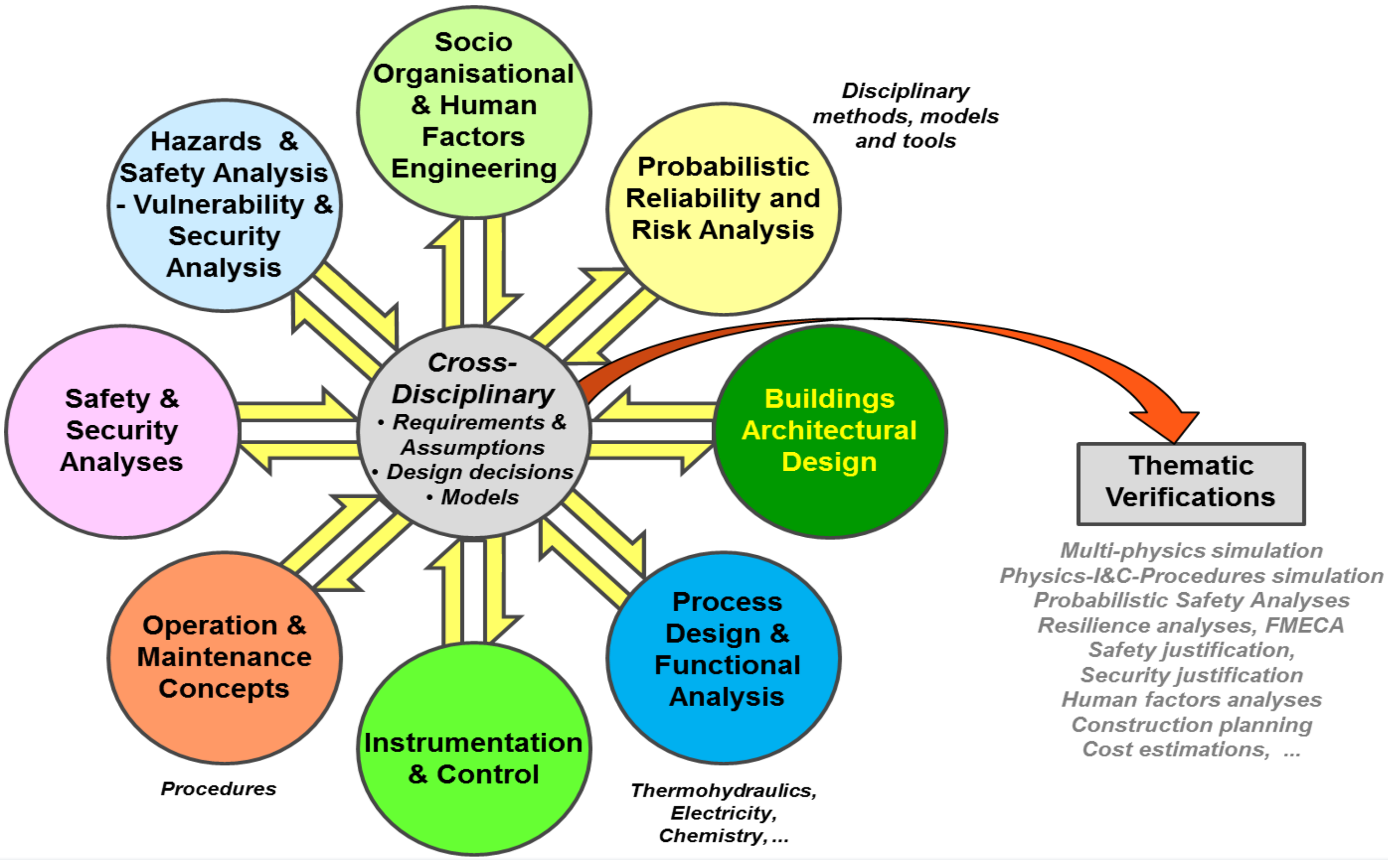
Three Roles in Value Chain of CPS Development

- **End-users** – building and operation of CPS
- **Technology providers** – provide and distribute tools
- **Methodology developers** – develop methods for model-based systems engineering
- **Regulatory authorities** – provide rules to operators for ensuring the stability of large distributed systems



System Co-Design via Requirements Modeling

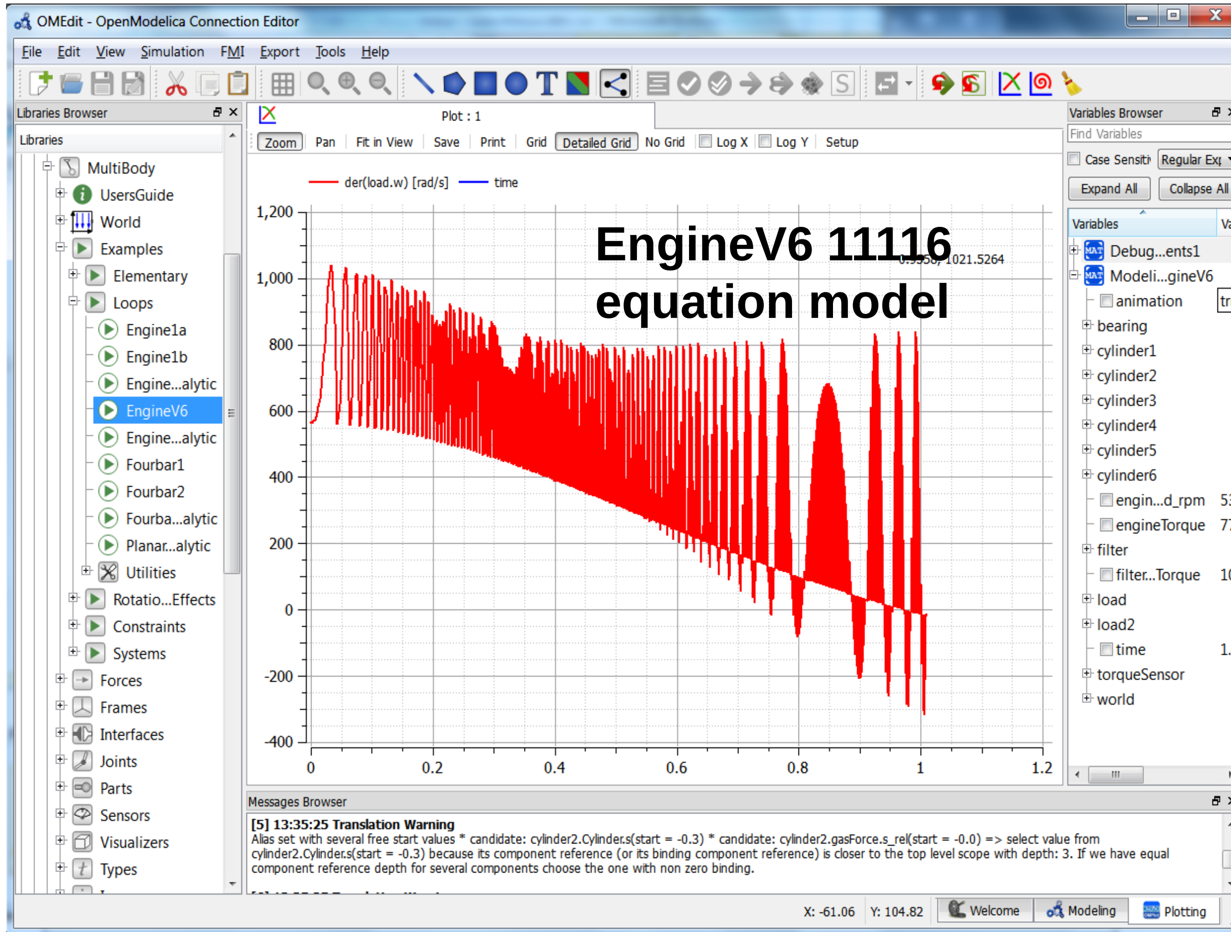
- Need for a **new neutral language** to **coordinate** all **engineering disciplines** via the **formal** expression of **requirements** and **assumptions** (formal == that may be **analyzed/simulated** at each step of the engineering process)



MODELICA

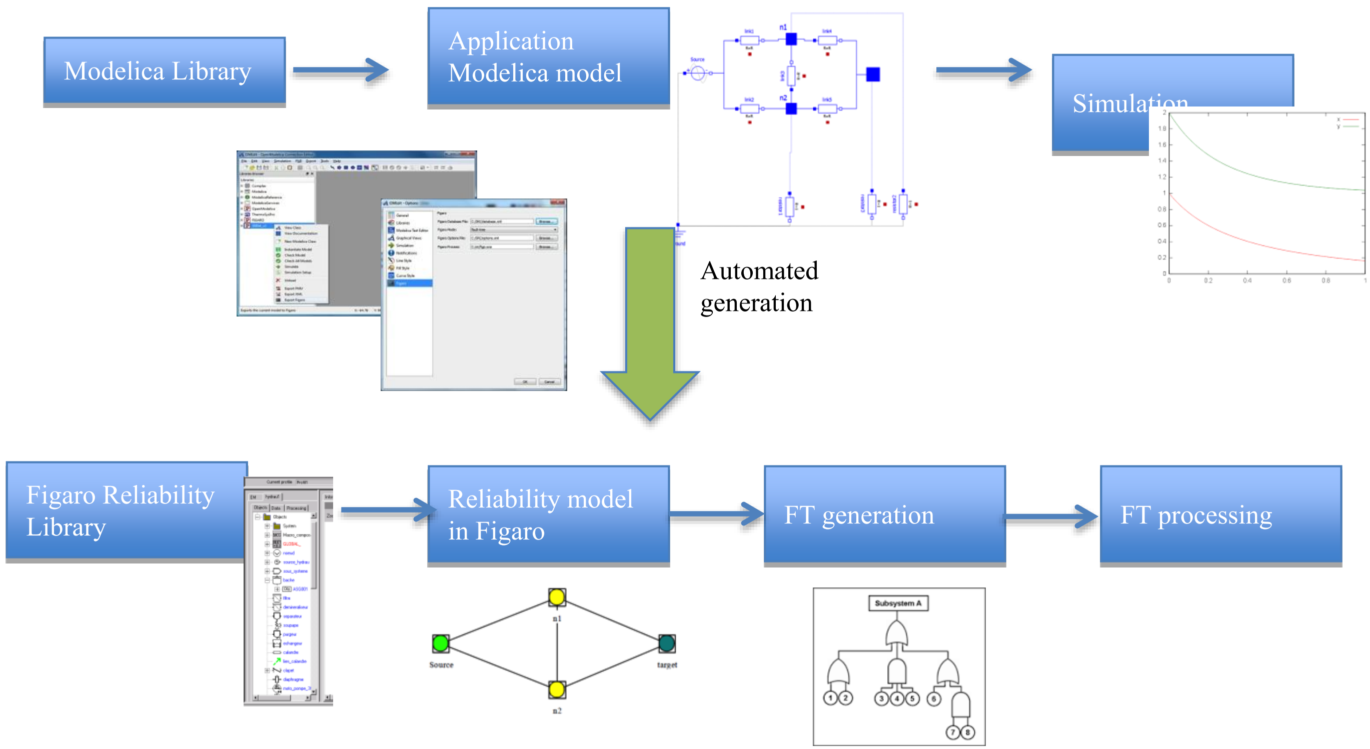
OpenModelica – Free Open Source Tool
developed by the Open Source Modelica Consortium (OSMC)

- Graphical editor
- Model compiler and simulator
- Debugger
- Performance analyzer
- Dynamic optimizer
- Symbolic modeling
- Parallelization
- Electronic Notebook for teaching

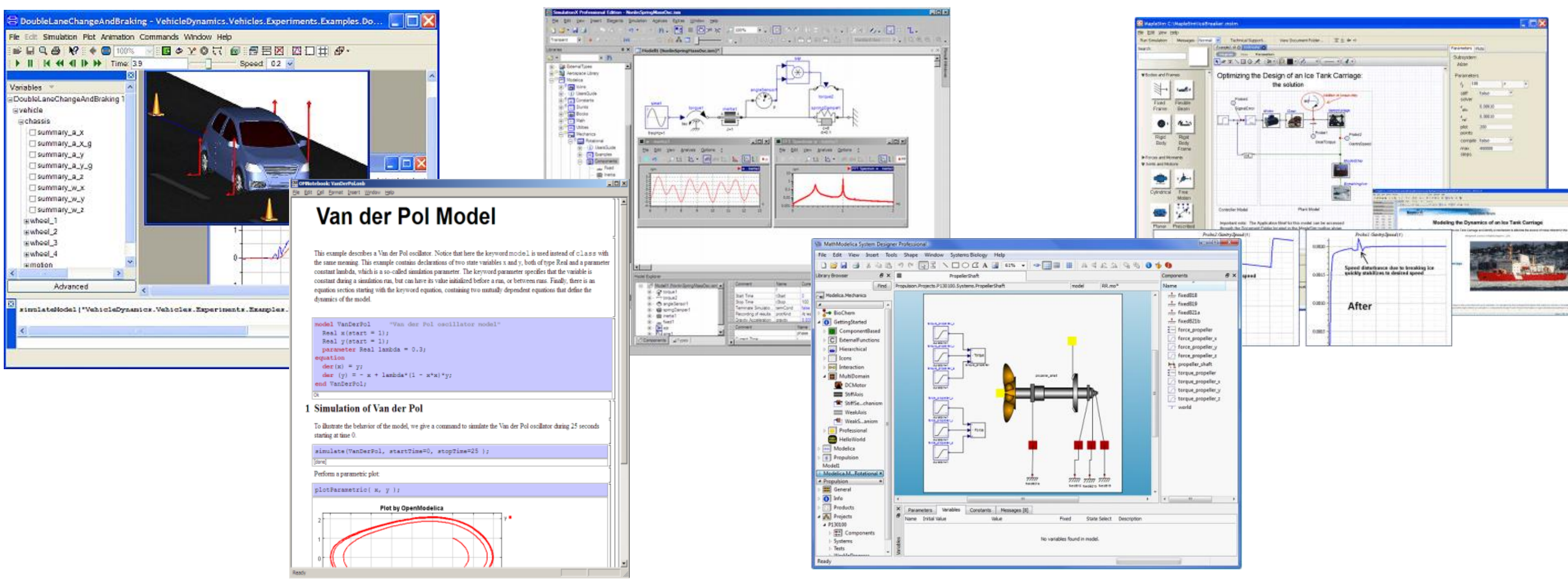


Model-based Failure Mode and Effects Analysis

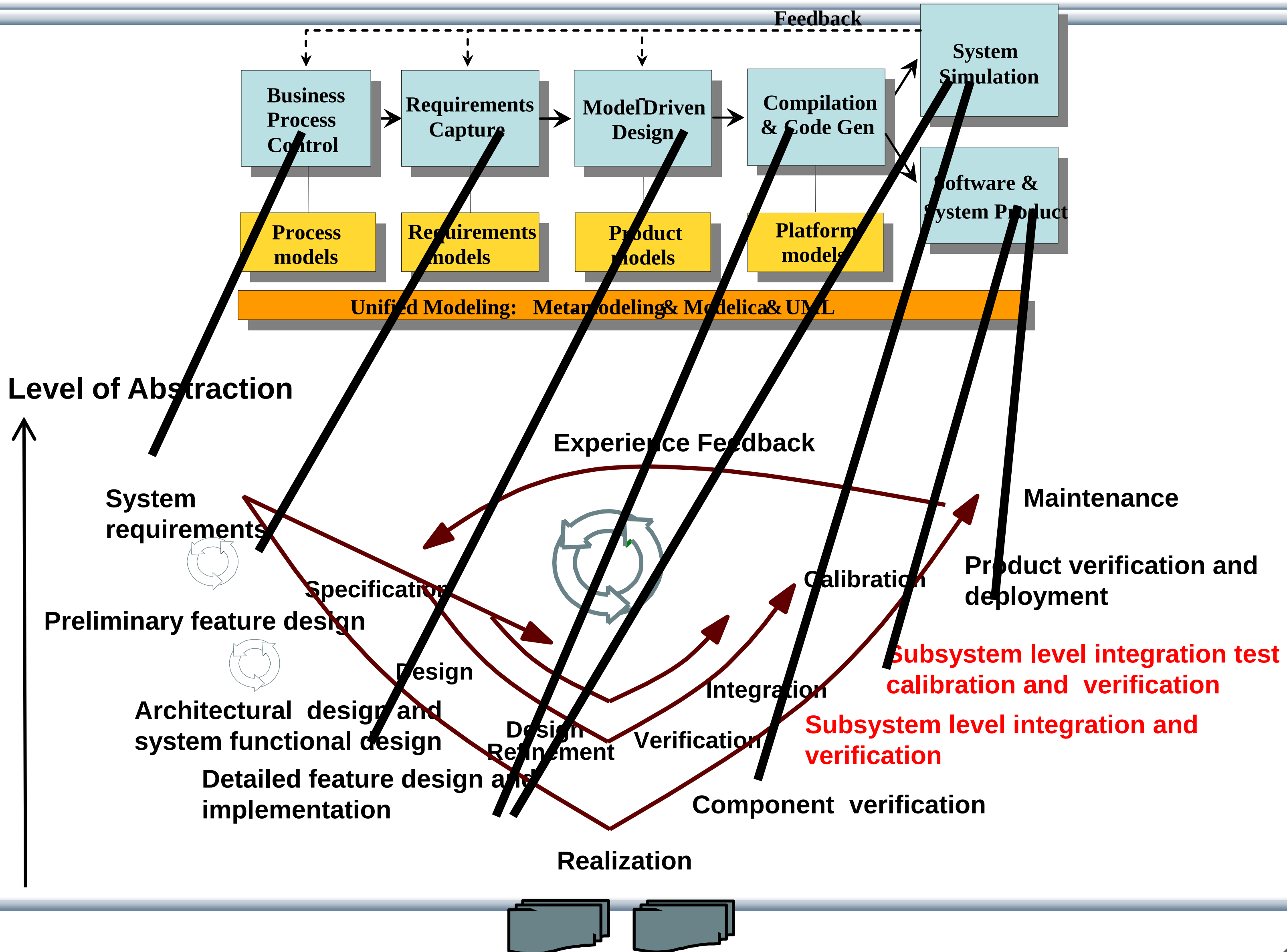
- Modelica models augmented with reliability properties can be used to generate reliability models in Figaro, which in turn can be used for static reliability analysis
- Prototype in OpenModelica integrated with Figaro tool



Dynamic Verification/Testing of Requirements vs Usage Scenario Models

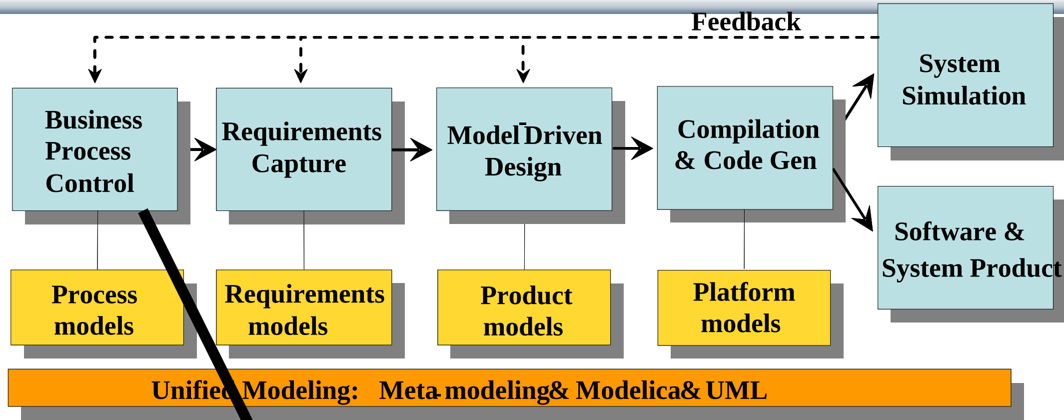


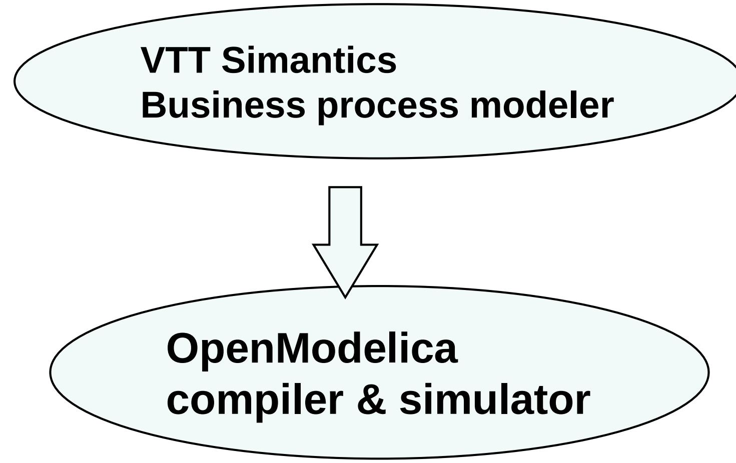
OpenModelica and Papyrus Based Model-Based Development Environment to Cover Product-Design V



Documentation, Version and Configuration Management

Business Process Control and Modeling

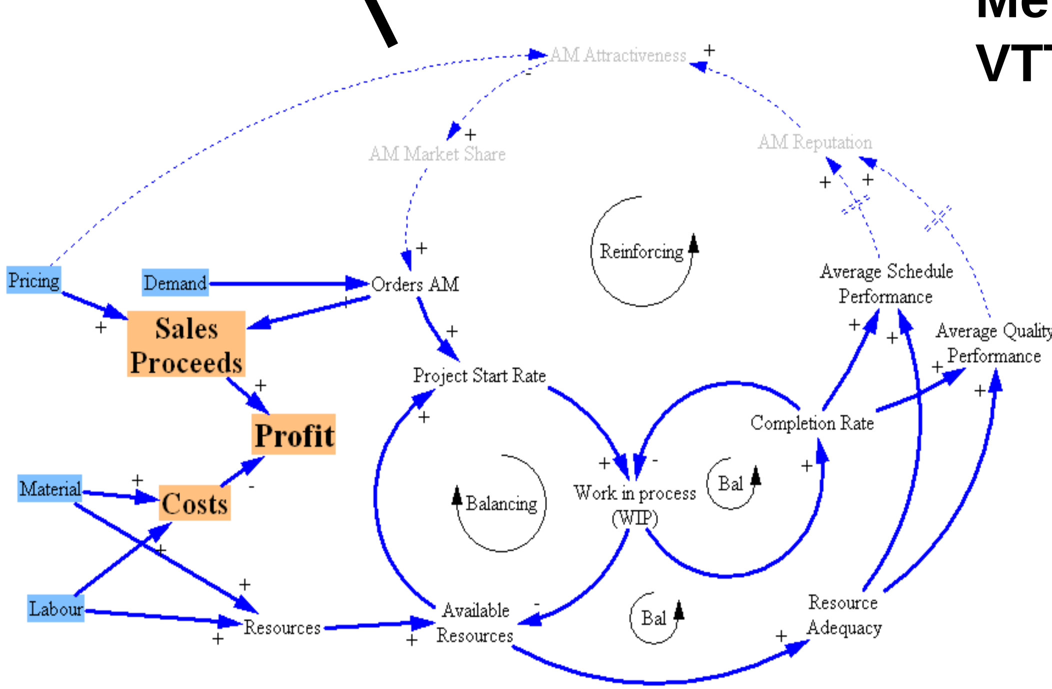




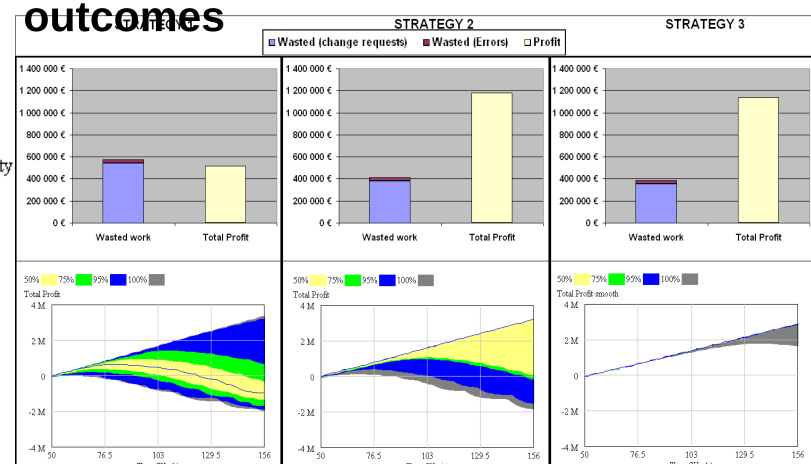
OpenModelica based simulation

Metso Business model & simulation

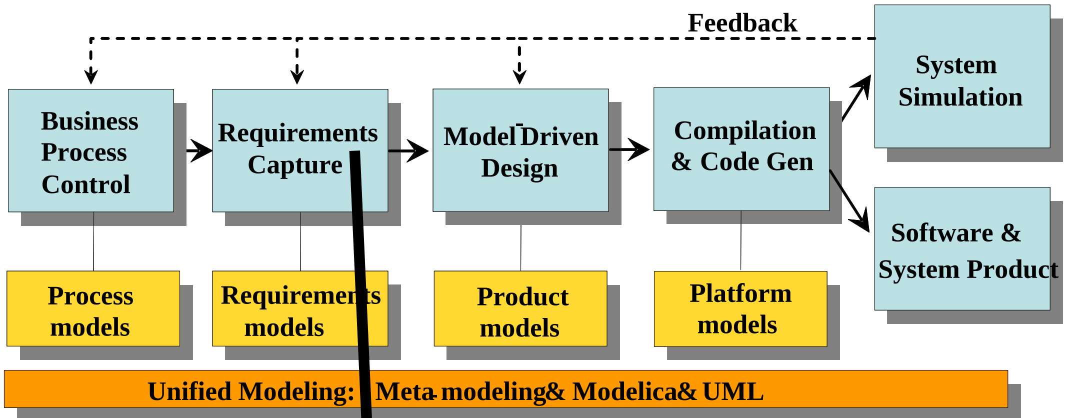
VTT Simantics Graphic Modeling Too



Simulation of 3 strategies with outcomes



Requirement Capture



vVDR (virtual Verification of Designs against Requirements)

in ModelicaML UML/Modelica Profile, part of OpenModelica

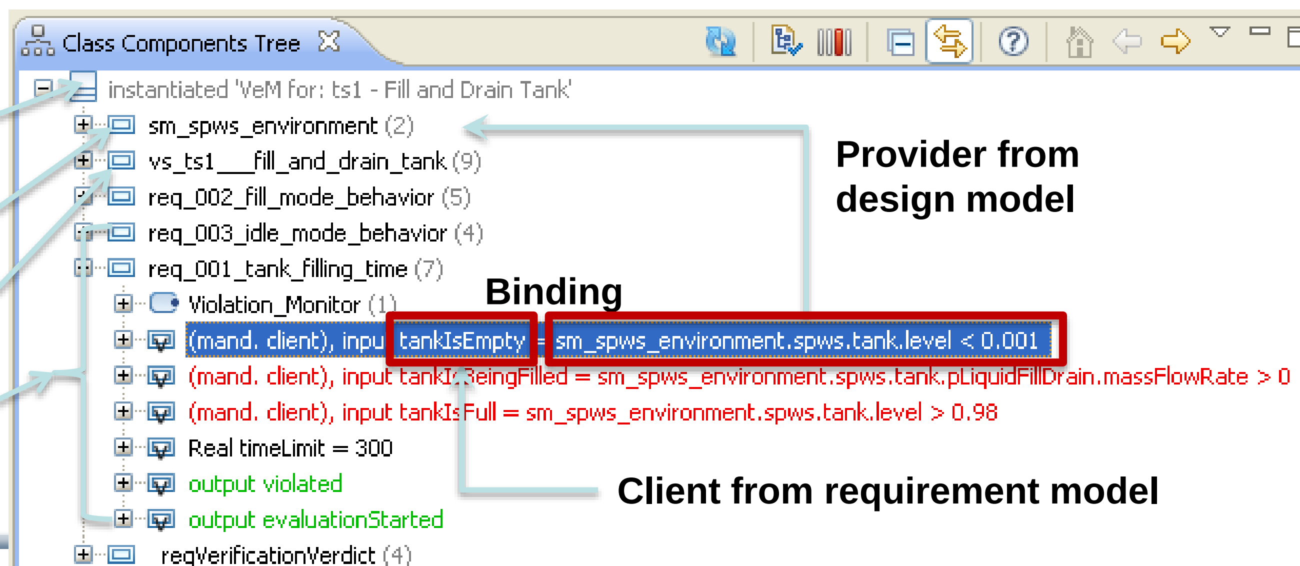
OpenModelica based simulation

Verification Model

Design Model

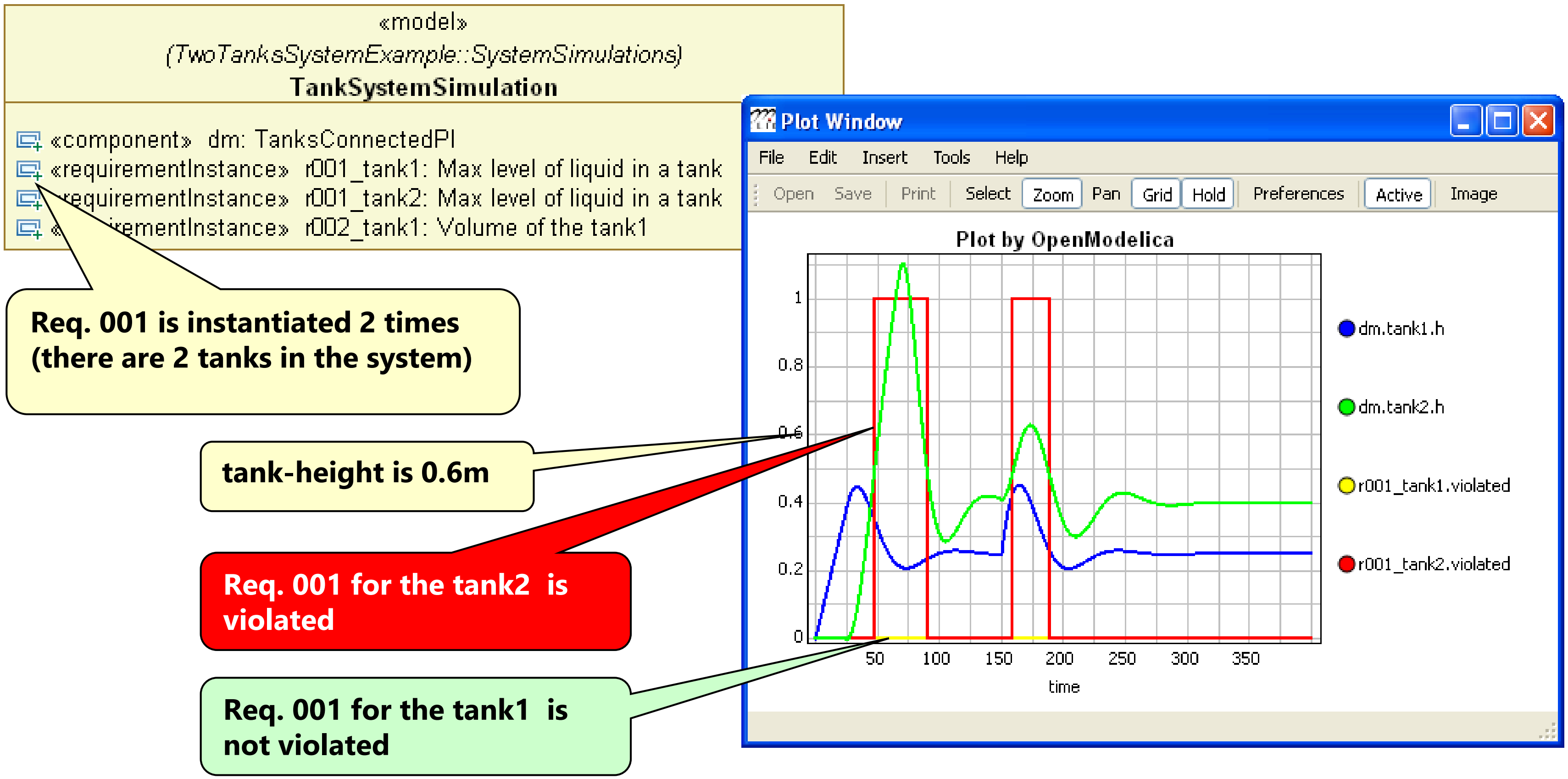
Scenario Model

Requirement Models



MODELICA

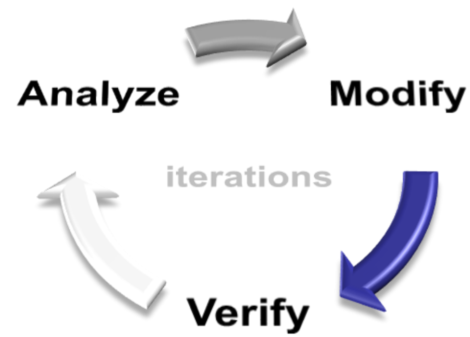
Example: Simulation and Requirements Evaluation (using the ModelicaML UML/Modelica prototype)



MODELICA

vVDR Method – virtual Verification of Designs vs Requirements

Actor	Task	Created Artifact
	Formalize Requirements	RMM Requirement Monitor Models
	Formalize Designs	DAM Designs Alternative Models
	Formalize Scenarios	SM Scenario Models
AUTOMATED	Create Verification Models *	VM Verification Models
AUTOMATED	Execute and Create Report	Reports
	Analyze Results	



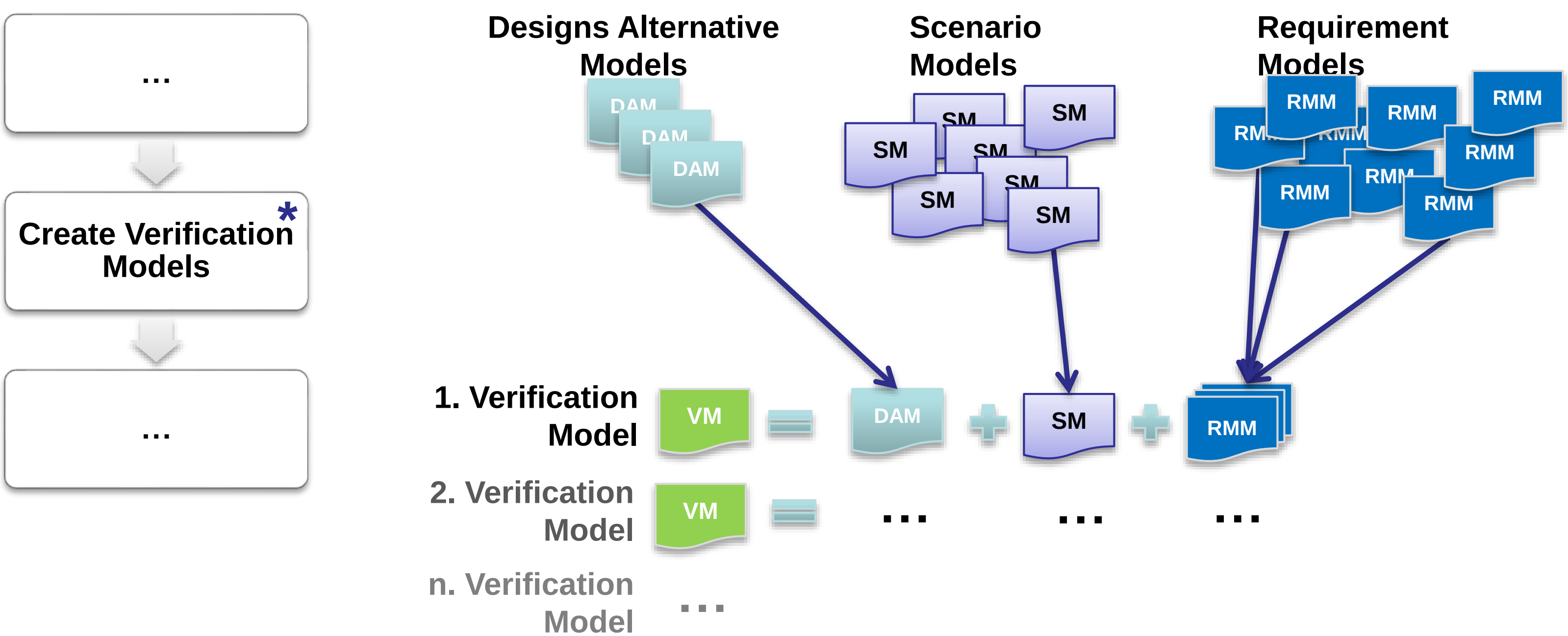
Goal: Enable on-demand verification of designs against requirements using automated model composition at any time during development.

MODELICA

Challenge

We want to verify **different design alternatives** against **sets of requirements** using **different scenarios**. Questions:

1) How to **find valid combinations** of **design alternatives**, **scenarios** and **requirements** in order to enable an automated composition of verification models?
2) Having found a valid combination: How to **bind all components correctly**?



Composing Verification Models
main idea

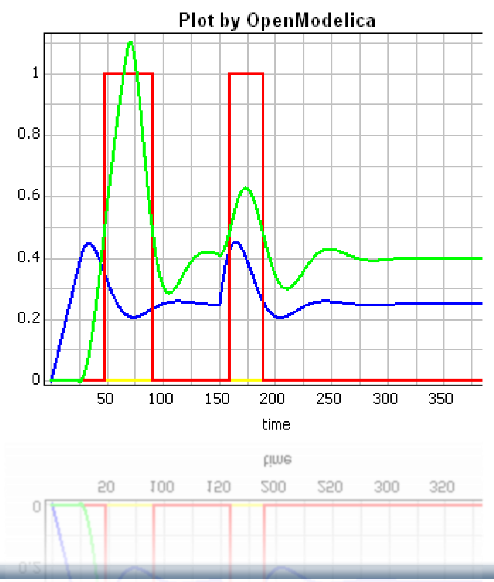
- Collect all **scenarios**, **requirements**, import **mediators**
- Generate/compose *verification models* automatically:
  - Select the **system model** to be verified
  - Find all **scenarios** that can stimulate the selected system model (i.e., for each mandatory client check whether the binding expression can be inferred)
  - Find **requirements** that are implemented in the selected system model (i.e., **check** whether for **each requirement** for all mandatory clients binding expressions can be inferred)
- Present the list of scenarios and requirements to the user
  - The user can select only a subset or scenarios or requirements he/she wishes to consider

Verification Report Generation, from Simulation of Verification Models (fr Requirements, Designs, appl scenarios)

Verification models are simulated.

The generated **Verification Report** is a prepared summary of:

- Configuration, bindings
- Violations of requirements
- etc.



Verification models number (3), executed (3), passed (0), failed (3)

Failed VeM for: s1-Fill and Drain Tank (Plot)
Failed VeM for: s2-Fill tank (Plot)
Failed VeM for: s3-Drain tank (Plot)

Failed VeM for: s1-Fill and Drain Tank (Plot)
(ModelicaMLModel::GenVeMs for: SPWS Environment_1::VeM for: s1-Fill and Drain Tank)

Settings: startTime = 0, **stopTime** = **1500**, tolerance = default, intervals = 0, outputFormat = plt

verdict allRequirementsEvaluated : **yes**
verdict someRequirementsViolated : **yes**

Model to be verified: **SPWS Environment**
(ModelicaMLModel::Design::SPWS Environment)

Verification Scenario: **s1-Fill and Drain Tank**
(ModelicaMLModel::Verification Scenarios::s1-Fill and Drain Tank)

madantory client: **vs_s1_fill_and_drain_tank.tankHeight** (**changed its value**)
Type : = ModelicaReal
Variability : = continuous
Binding code : = sm_spws_environment.spws.tank.height

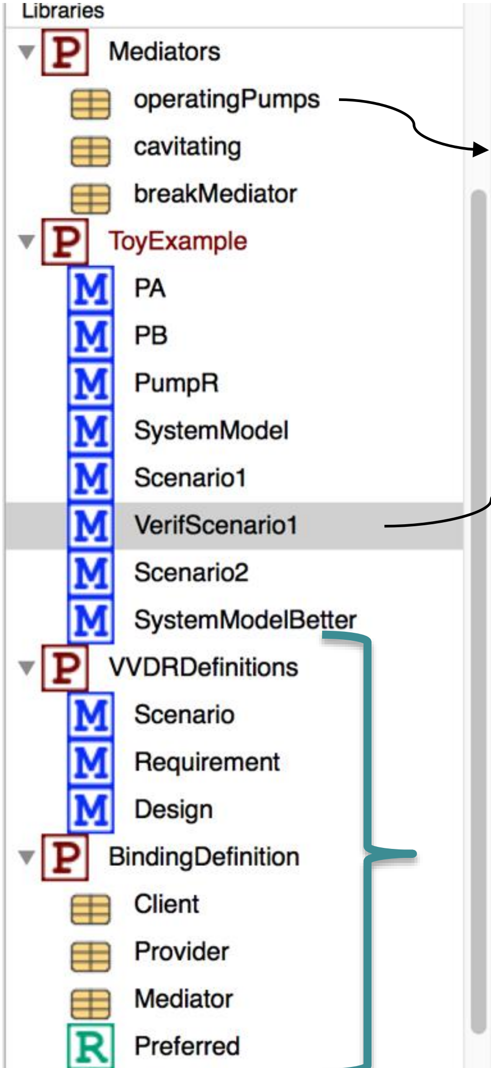
Violated Requirement: **Drain mode behavior (ID 004)**
(ModelicaMLModel::Requirements::Drain mode behavior)
Text: When the system is drained only the fill/drain valve should be open, all other valves should be closed.

verdict evaluated : **yes**
verdict violated : **yes**

madantory client: **req_004_drain_mode_behavior.fillDrainValveIsOpen** (**changed its value**)
Type : = ModelicaBoolean
Variability : = continuous
Binding code : = sm_spws_environment.spws.fillDrainValve.isFullyOpen

madantory client: **req_004_drain_mode_behavior.otherValvesAreClosed** (**changed its value**)
Type : = ModelicaBoolean
Variability : = continuous
Binding code : = if sm_spws_environment.spws.overFlowValve.isFullyClosed and sm_spws_environment.spws.supplyVavle.isFullyClosed then true else false

Support of vVDR in Modelica within OMEdit in OpenModelica



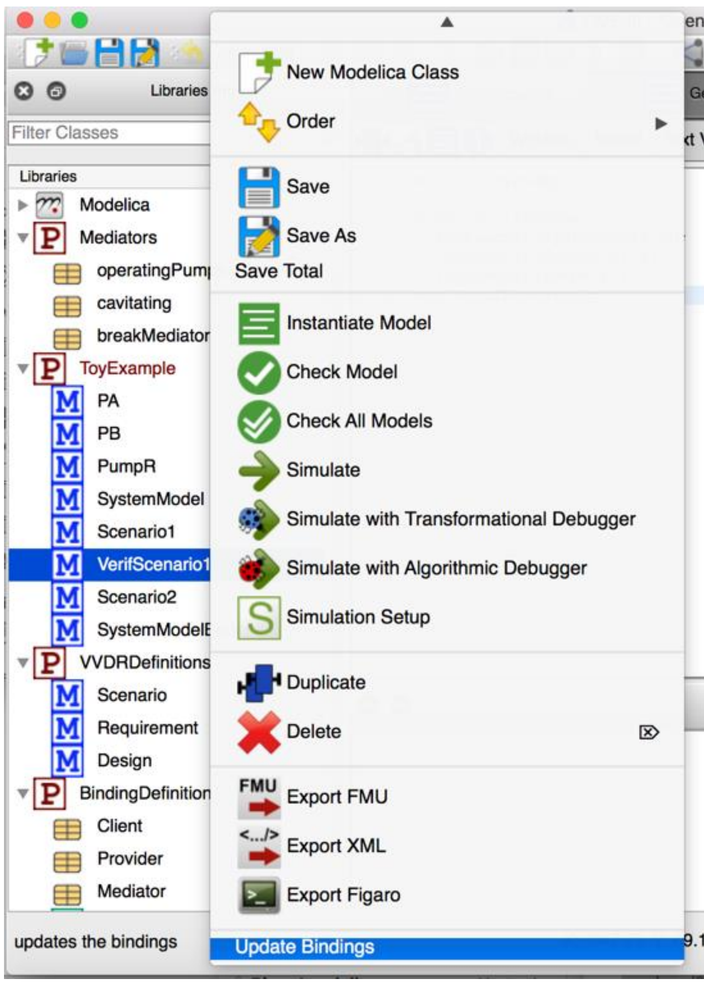

```
record operatingPumps
  extends  Mediator(mType = "Boolean",
  clients = {Client(modelID = "ToyExample.PumpR", component = "inOperation")},
  providers = {Provider(modelID = "ToyExample.PA", template = "if %getPath.on then 1 else 0"),
    Provider(modelID = "ToyExample.PB", template = "if (%getPath.volFlowRate) > 0 then 1 else 0")}
           );
end operatingPumps;
```

```
within ToyExample;

model VerifScenario1
  ToyExample.SystemModel md;
  ToyExample.Scenario1 s1;
  ToyExample.PumpR r1;
end VerifScenario1;
```

vVDR concepts in **standard Modelica**

- **mediators** mapped to **records**
- **requirements**, **design**, **scenarios** mapped to Modelica classes

Single Scenario Generation



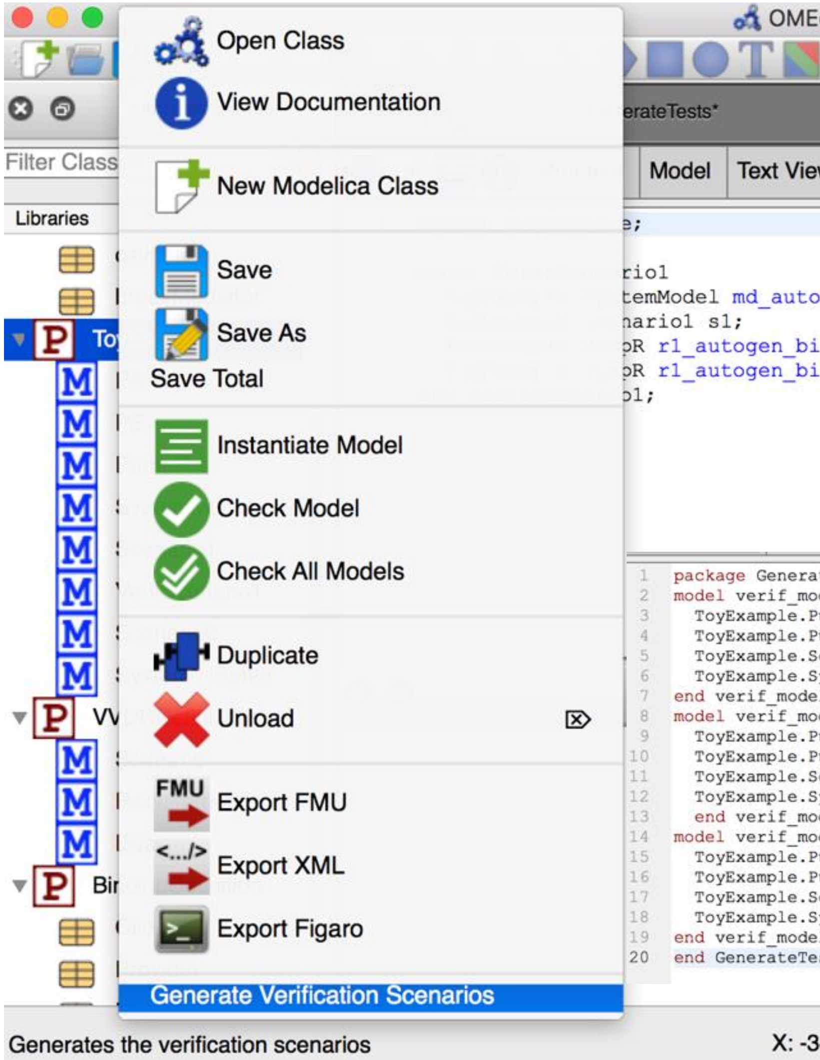

```
within ToyExample;

model VerifScenario1
  ToyExample.SystemModel md;
  ToyExample.Scenario1 s1;
  ToyExample.PumpR r1;
end VerifScenario1;
```

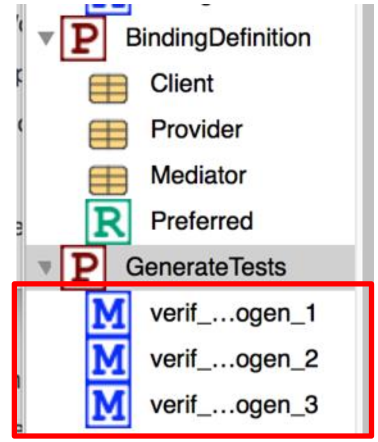
```
within ToyExample;

model VerifScenario1
  ToyExample.SystemModel md_autogen_bind_0(timeBreak = s1.timeFailure);
  ToyExample.Scenario1 s1;
  ToyExample.PumpR r1_autogen_bind_0(cavitate = md_autogen_bind_0.pa.cavitating);
  ToyExample.PumpR r1_autogen_bind_1(cavitate = md_autogen_bind_0.pb.cavitating);
end VerifScenario1;
```

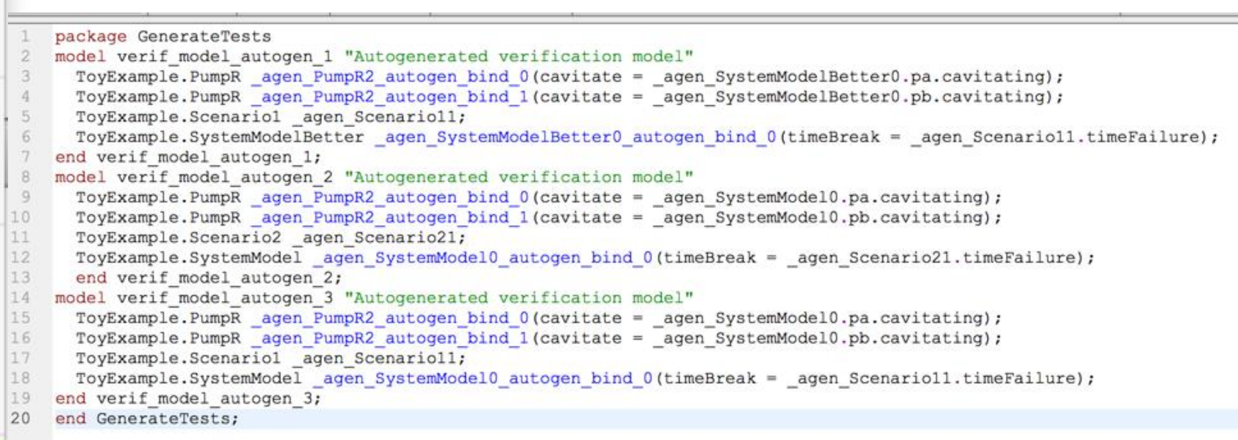
Generating correct number of requirement instances

Connecting the **design** model and the **requirement**

Batch Scenario Generation







EDF SRI Case Study of vVDR Method Conclusion and Lessons Learnt

- Showed **applicability of vVDR method** to **realistic industrial** applications
- ModelicaML is a **promising prototype** implementation of the vVDR method, needs improved usability and stability
- Lessons learnt:
  - **Formalized requirements** should be **tested separately** in order to ensure correctness
  - **Model validity asserts** must be included
  - Parameterized **requirement monitors** can be re-used as **library** components (later realized in MODRIO project)
- Later work, now ongoing
  - **Stochastic aspects** (model uncertainties, tolerances in requirements, ...) should be taken into account

What is Missing in our Systems Engineering Tool Support?

- We already have many parts of the desired environment:
  - Business process modeling
  - Requirement modeling (related to design properties)
  - Design modeling
  - Model simulation and product generation
  - Verification based on simulations

- Perhaps missing: expressing and verification of general requirements in an early phase, independent of design choices

What vs How and a Possible Requirement Language

- Desirable for **requirements** to be **independent of** particular **design** architectures
- A Design model expresses **How** to solve a problem
- A Requirement should express **What** conditions should be satisfied
- Desirable for requirements to be able to express **quantifiers**, **temporal constraints**, related to (sets of) objects that satisfy certain conditions
- A natural/neutral rule language for requirements? Close to **natural language?** or **borrow** many elements from an **existing** modeling **language**, e.g. Modelica?

Conclusions

- First step of a requirements language – **bridge/unification of UML/SysML and Modelica (ModelicaML)**, for design verification versus requirements on application scenarios
- Second step, requirements library in **Modelica** and **OpenModelica prototype** for design verification versus requirements on application scenarios
- Third step (to be realized), a formal **requirement language** to express basic requirement **rules**, for **early requirement debugging and verification** without the need for a (detailed) system architecture