Requirement Formalization in Modelica

Lena Buffoni, Linköping University
(with material from Wladimir Schamai)

LINKÖPING UNIVERSITY

Lecture Plan

- Requirements in cyber-physical systems
- How do we express requirements?
- How do we verify requirements?

Material

- Slides

- OpenModelica: openmodelica.org

- Example:
- https://gitlab.ida.liu.se/olero90/RequirementsTutorial

WHAT IS A REQUIREMENT?

Definitions

"Software requirements express the *needs* and *constraints* placed on a software product that contribute to the solution of some real-world problems."

(Kotonya and Sommerville, 2000)

"A requirement is something the product *must do* or a *quality it must have*."

(Suzanne & James Robertson, 2006)

Examples from specification

1. "*The torque of any ADGB electrical motor shall not be superior to 20 N.m for more than 1 sec.*"

2. *"At least 2 pumps shall be in operation at any time"*

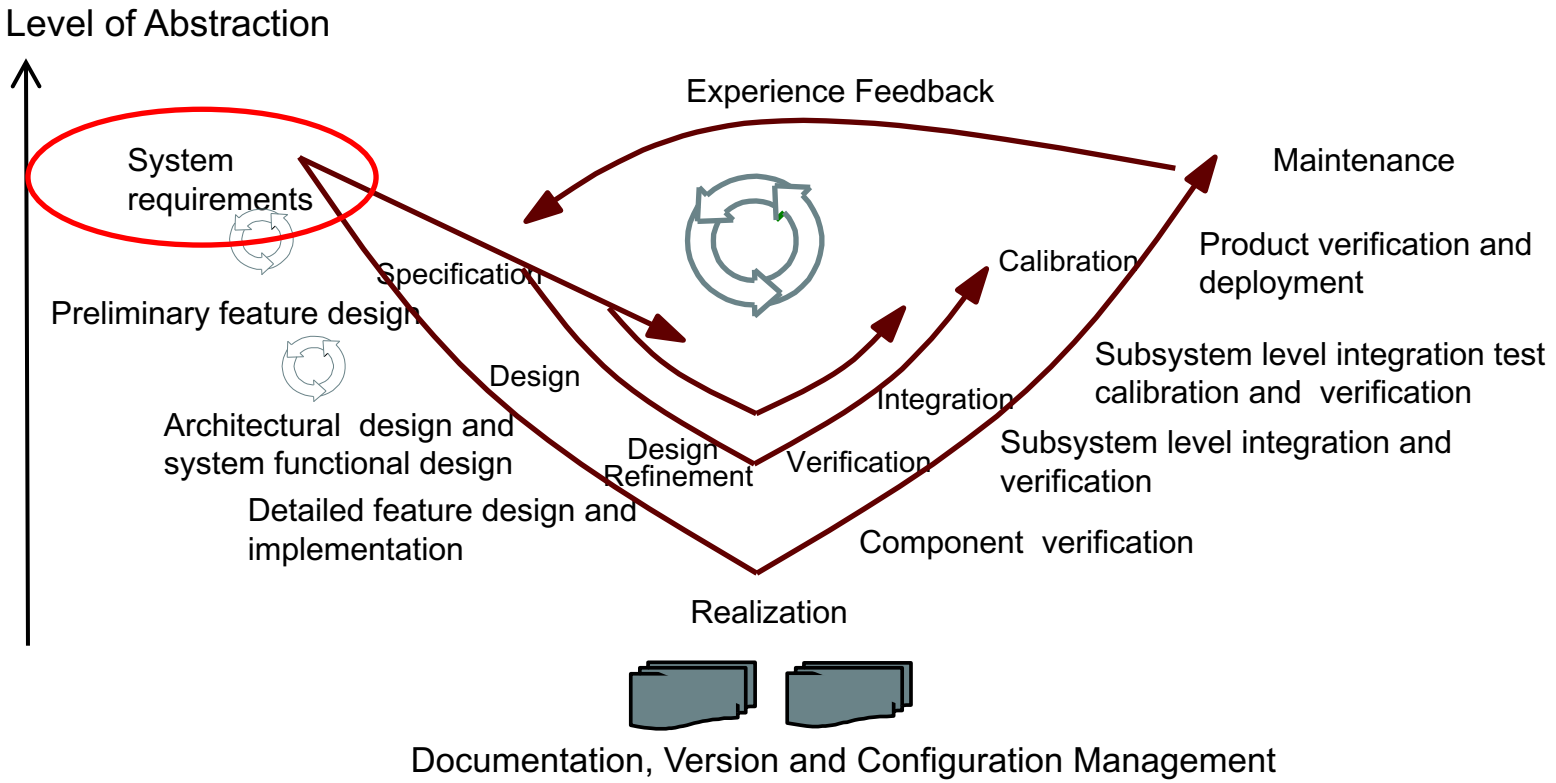
3. *"In the absence of any BPS component failure or in the presence of a single sensor failure, when the BPS is not under maintenance, and in case of MPS loss, at least two of Set2 to Set5 must be powered within 40 s "*

Some Requirements Engineering Literature

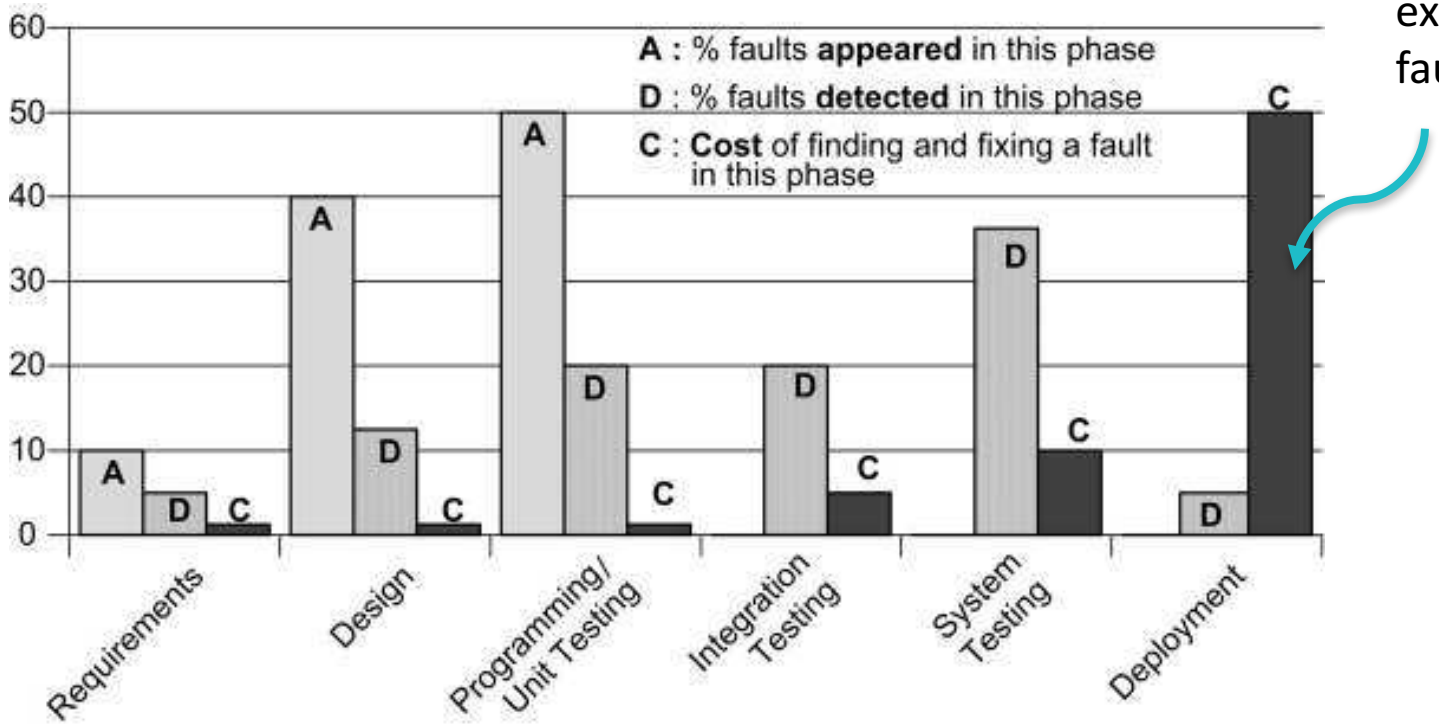
- Kotonya G. and Sommerville, I. Requirements Engineering: Processes and Techniques. Chichester, UK: John Wiley & Sons
- Software Requirements Engineering Methodology (Development) Alfor,M. W. and Lawson,J. T. TRW Defense and Space Systems Group. 1979.
- Thayer, R.H., and M. Dorfman (eds.), System and Software Requirements Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1990.
- Royce, W.W. 'Managing the Development of Large Software Systems: Concepts and Techniques', IEEE Westcon, Los Angeles, CA> pp 1-9, 1970. Reprinted in *ICSE '87, Proceedings of the 9th international conference on Software Engineering*.
- Requirements bibliography Reviewed November 10th 2011
- Sommerville, I. *Software Engineering*, 7th ed. Harlow, UK: Addison Wesley, 2006.
- Ralph, Paul (2012). "The Illusion of Requirements in Software Development". *Requirements Engineering*.

WHY DO WE WANT TO MODEL REQUIREMENTS?

Requirement formalization in product life-cycle



Why are requirements important?



50 times more expensive to fix a fault at this stage!

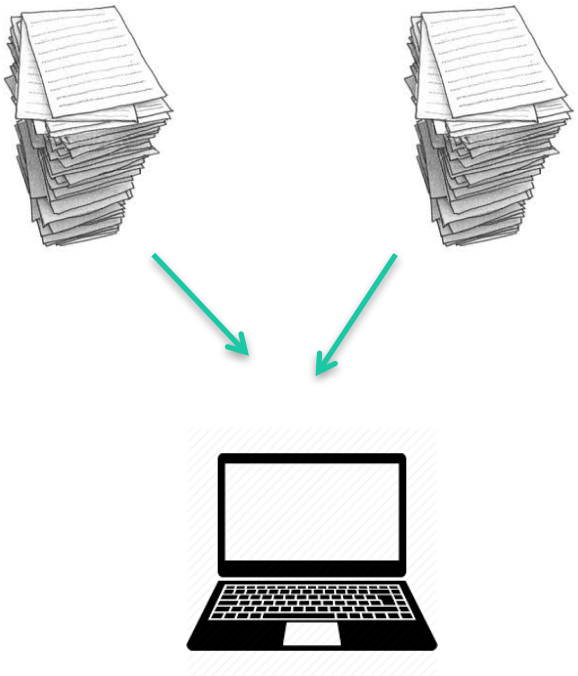
Errors in system requirements cause large cost increase in product development, or even completely failed projects

From theory to practice

- Requirements engineering should be an integrated part of the development tool chain

- Some resources:
  - International Requirements engineering conference home page: ***http://requirements-engineering.org/***
  - Example of requirements engineering journal: ***Requirements Engineering***, Springer Verlag, http://link.springer.com/journal/766

What is requirement analysis used for?



- Detect and resolve conflicts between requirements and different levels of requirement specification
- Define the bounds of normal system behaviour
- Define interaction with the environment
- Break down high-level requirements into more detailed requirements

How do we represent requirements?

Can all requirements be formalized?

The maximum rate of liquid flow in the pump should be below a certain threshold

Natural language specification

lflow_rate < lflow_rate_max

Computable/machine-readable form

Can all types of requirements be formalized?

Functional: specify a behavior or function

- *"The pump shall not cavitate"*

Non-functional: system constraints performance constraints

- *"The controller code should be in C"*
- *"The response time should be less than 0.1s"*

Why model requirements in Modelica?

- We can have them alongside the component Model
- Declarative style of Modelica well suited to requirement expression
- Integrated in the tool chain
- Can be verified during simulation

What do need to represent requirements?

- We need to identify requirements ()
  - a special **interface**
- We must know how to evaluate a requirement
  - Requirements must have a **status**
- Requirements should **not modify** the physical model
  - The status is the only output variable

- Requirement models can (possibly) contain extensions for expressing requirements in a more requirement-designer friendly way (FORM-L macros for example)

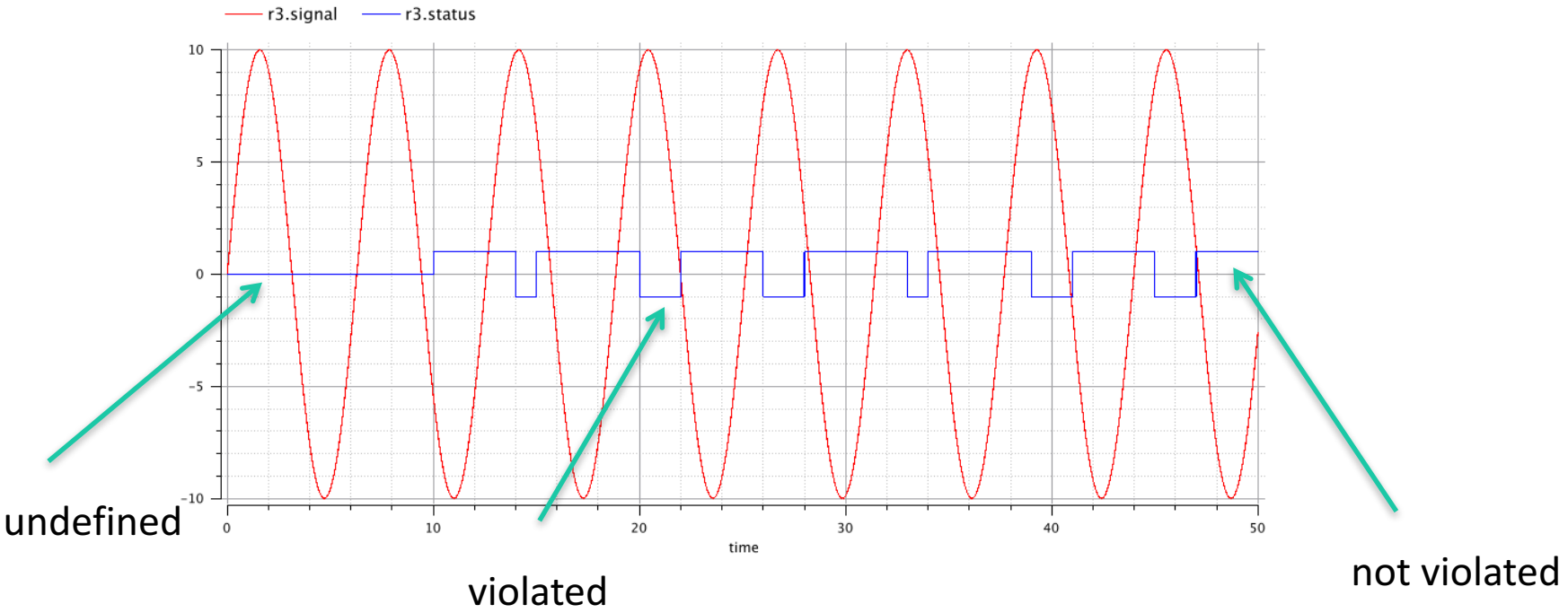
Status of a requirement

Example requirement: **when the pump is on the flow must be above a minimum threshold**

The **status** variable is 3-valued and applies to a specific instant in time:

- **Violated** : the requirement is applicable and the conditions of the requirement are not fulfilled – the pump is on and the flow is below the minimum required
- **Satisfied** : the requirement is applicable and the conditions of the requirement are fulfilled – the pump is on and the flow is above the minimum required
- **Not_applicabl**e : the requirement cannot be applied or is not relevant - the pumps are off

Status over simulation time



Status changes over time, so we need to track it over the course of the simulation

Example 1 - LimitInFlow


```
model LimitInFlow "A2: If pump is on then in-flow is
                                   less than maxLevel"
    extends Requirement;
    //qOut from the Source component
    input Real liquidFlow;
    input Boolean pumpOn;
    parameter Real maxLevel = 0.05;
equation
   if (pumpOn) then
     if (liquidFlow < maxLevel) then
       status = Status.not_violated;
     else
       status = Status.violated;
    end if;
    else status = Status.not_applicable; end if;
  end LimitInFlow;
```

☑ Standard Modelica

Modeling approaches

- Standard Modelica
- State-machines
- Dedicated libraries or languages

Use case:
Requirement specification for a tank system

- Req. 001: The volume of each tank shall be at least 2 m3.

- Req. 002: The level of liquid in a tank shall never exceed 80% of the tank height.

- Req. 003: After each change of the tank input flow, the controller shall, within 20 seconds, ensure that the level of liquid in each tank is equal to the reference level with a tolerance of ± 0.05 m.

Req 001: standard Modelica

```
within TwoTanksExample.Requirements;

model LiquidLevel "The level of the liquid shall never exceed 80% height"
 extends VVDRlib.Verification.Requirement;
 input Real waterLevel;
 //input Real tankVolume;
 parameter Real tankVolume = 2;
 equation
  status = if (waterLevel < 0.8*tankVolume) then
   VVDRlib.ReqStatus.NOT_VIOLATED else VVDRlib.ReqStatus.VIOLATED;
end LiquidLevel;
```

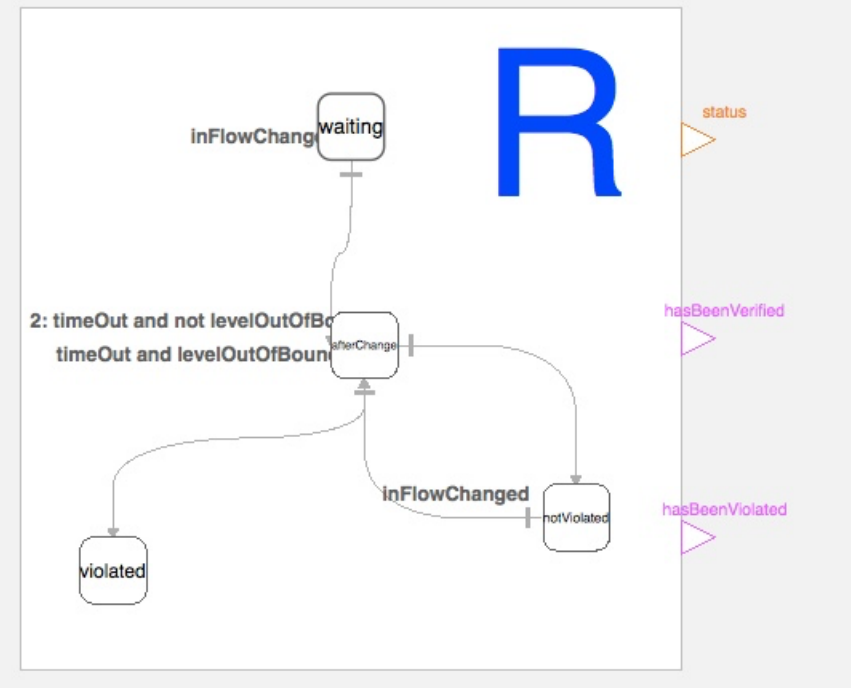
Req 002: standard Modelica

```
within TwoTanksExample.Requirements;

model Volume_of_a_tank "The volume of each tank shall be at least 2 m3.
"
extends VVDRlib.Verification.Requirement;
input Real tankVolume;
parameter Modelica.SIunits.Volume requiredVolume = 2;
equation
 status = if tankVolume < requiredVolume
  then VVDRlib.ReqStatus.VIOLATED else VVDRlib.ReqStatus.NOT_VIOLATED;
end Volume_of_a_tank;
```

Req 002: state machines

"After each change of the tank input flow, the controller shall, within 20 seconds, ensure that the level of liquid in each tank is equal to the reference level with a tolerance of ± 0.05 m."



Dedicated language: FORM-L

- FORM-L a language for property expression proposed by EDF
- Aims to make requirement expression easier for engineers
- Example :

```
 Form-L: during (On and (MPSVoltage > V170))
check no (Off becomes true);
```

- Integrate it with Modelica to breach semantic gap between how engineers and modelers express requirements

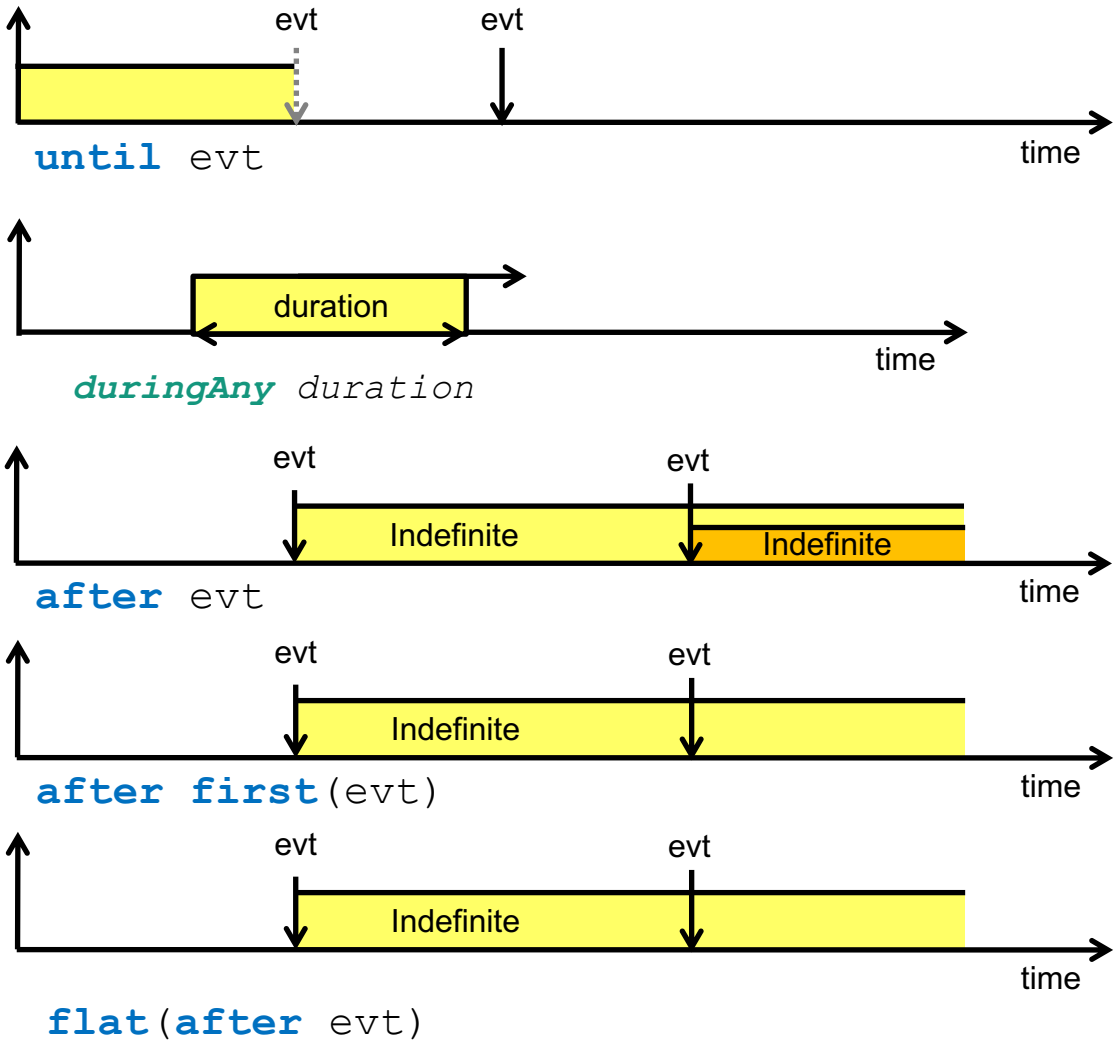
Property modeling: structure of a property

- Properties are divided into 4 main parts:

  WHERE / WHEN / WHAT / HOW_WELL

- WHERE: spatial locator. Builds the sets of objects defined by static properties.
- WHEN: temporal locator. Locate properties in time on sets built with spatial locators.
- WHAT: condition to be verified at the specified WHERE and during the specified WHEN.
- HOW_WELL: probabilistic condition of the condition to be verified

Time Locators



Special blocks that define when a requirement should be verified (after, until, afterFor…)

Many more specified in the FORM-L documentation by N. Thuy

Textual requirement modeling – Using TimeLocator WithinAfter

```
model R3 "requirement R3 is during any s6 check avg(signal) <  20;"
   extends Requirement;
   input Real signal;
 equation
   if time < 6 then
     status = undefined;
   else
     if DuringAnyAverage(timeWindow = 6, value = signal, thr = 2).out
       then
       status = not_violated;
     else
       status = violated;
     end if;
  end if;
 end R3;
```

DuringAnyAverage time locator

```
 block DuringAnyAverage "Returns true if the average of the values in a
sliding window of sizeWindow is below the threshold"
    extends TimeLocator;
    parameter Real timeWindow = 10;
    parameter Real sampleFrequency = 1;
    parameter Real thr;
    input Real value;
  protected
    Real buffer[size];
    Integer counter;
    parameter Integer size = integer(floor(timeWindow /
sampleFrequency)) + 1;
  algorithm
    when sample(0, sampleFrequency) then
      buffer[counter + 1] := value " store values in buffer";
      counter := mod(counter + 1, size-1);
      if time > timeWindow then
        out := average(buffer) <= thr;
      else
        out := true;
      end if;
    end when;
  end DuringAnyAverage;
```

Requirements library

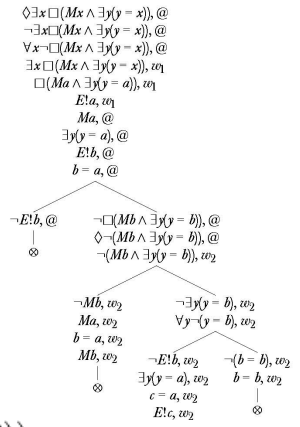
https://github.com/modelica/Modelica_Requirements

- Development led by M. Otter at DLR
- Contains implementations for all temporal locators
- Under development

How do we use requirements for verification?

Formal model checking methods

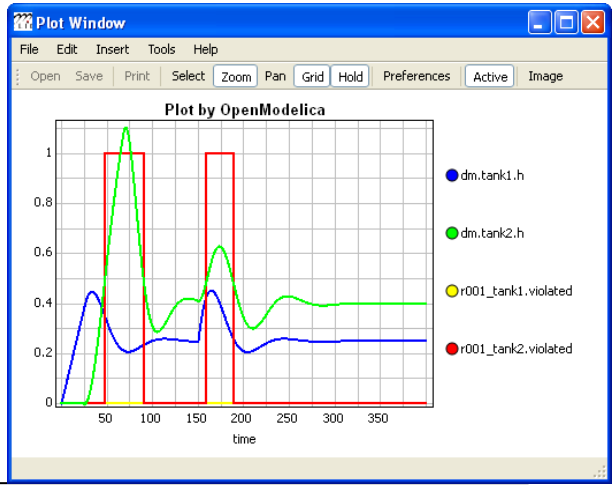
- Formal language specification (ESTERL, petri nets, temporal languages)
- Verification tools (Coq, certified code generators…)
  - + Prove correctness theoretically
  - + Give theoretical guarantees
  - - Heavy/expensive to use
  - - What can be verified is limited



Tautology:	$P \to (Q \to (P \wedge Q))$
Necessitation:	$\Box(P \to (Q \to (P \wedge Q)))$
Modal axiom:	$\Box(P \to (Q \to (P \wedge Q))) \to (\Box P \to \Box(Q \to (P \wedge Q)))$
Modus ponens:	$\Box P \to \Box(Q \to (P \wedge Q))$
Modal axiom:	$\Box(Q \to (P \wedge Q)) \to (\Box Q \to \Box(P \wedge Q))$
Classical arg:	$\Box P \to (\Box Q \to \Box(P \wedge Q))$
Classical arg:	$(\Box P \wedge \Box Q) \to \Box(P \wedge Q)$

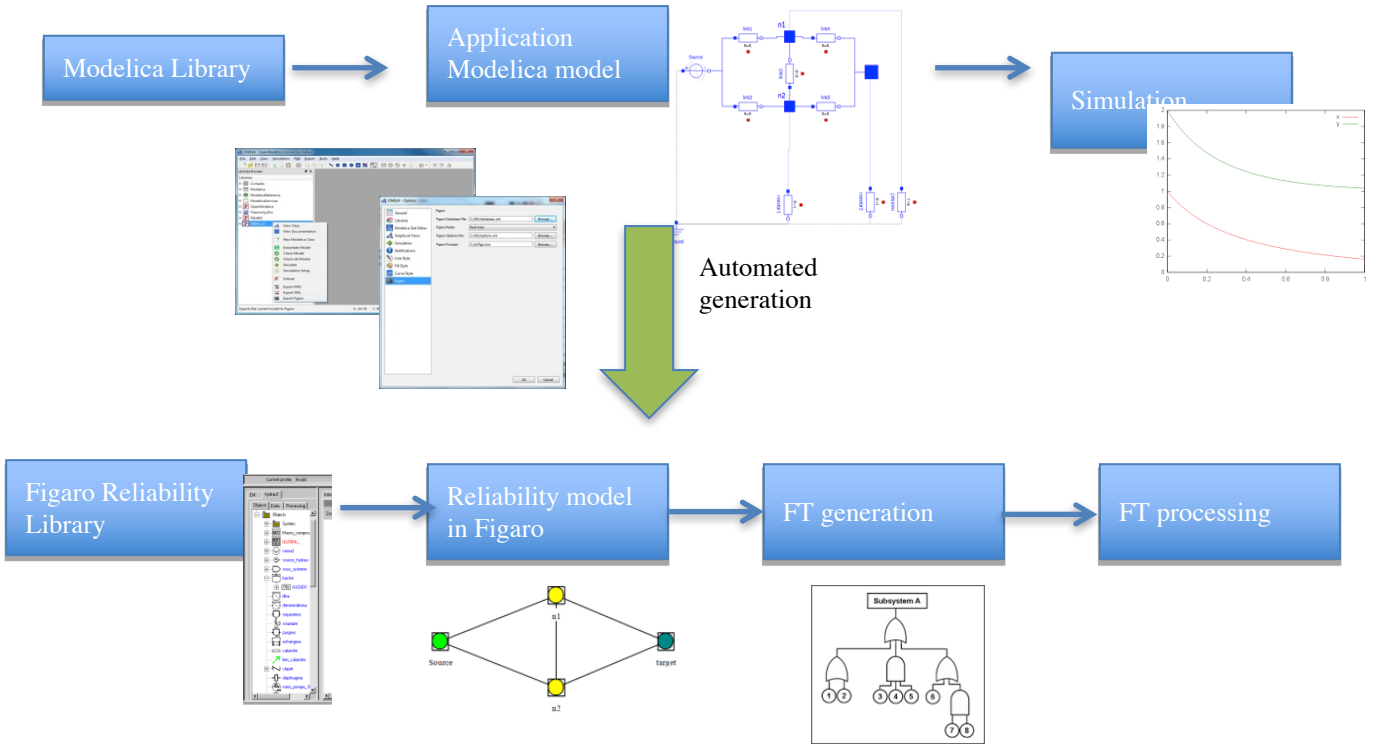
Simulation based model checking methods

- Simulate a certain number of scenarios and verify requirements dynamically
- More flexible
- Find the presence not the absence of bugs
- Can still give good guarantees

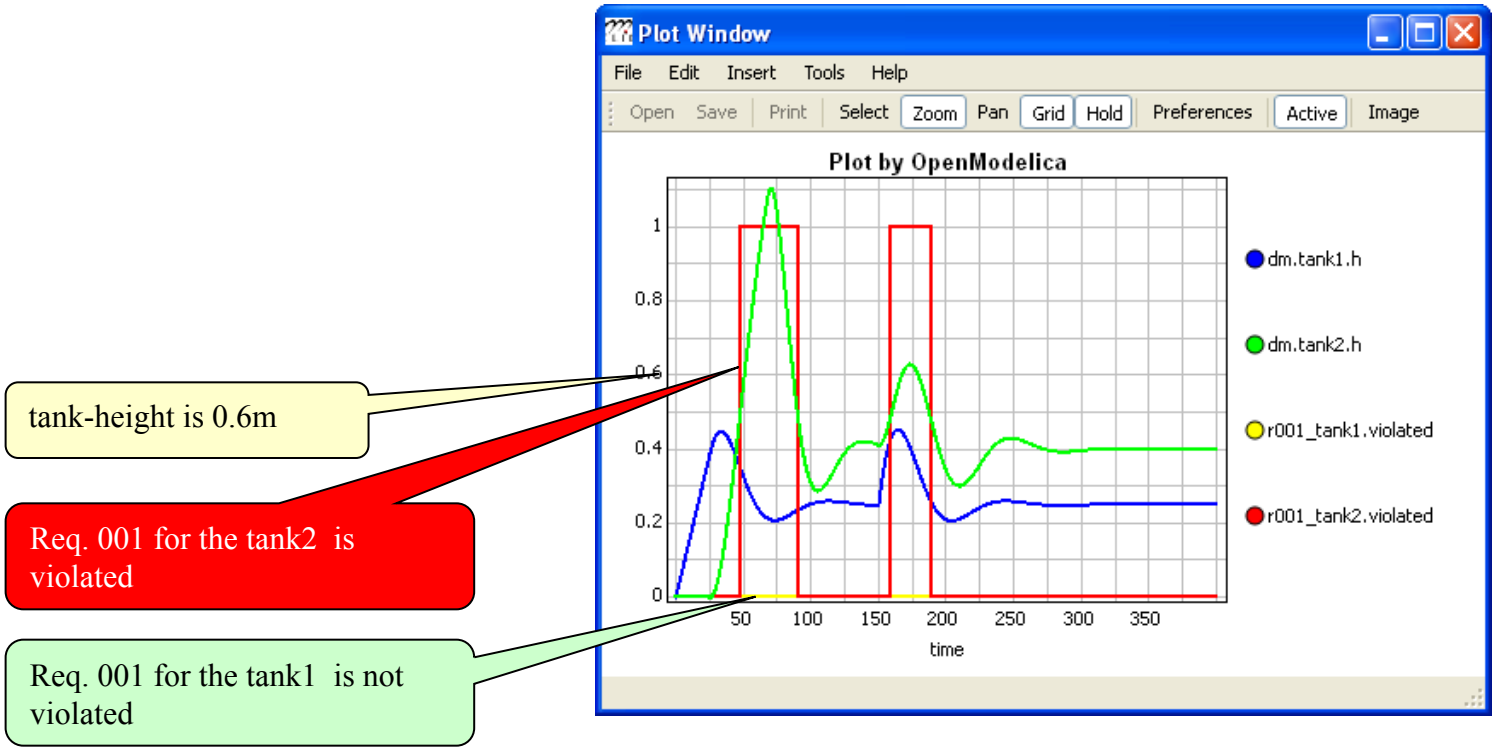


Failure Mode and Effects Analysis (FMEA)

- 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 (which is becoming open-source)



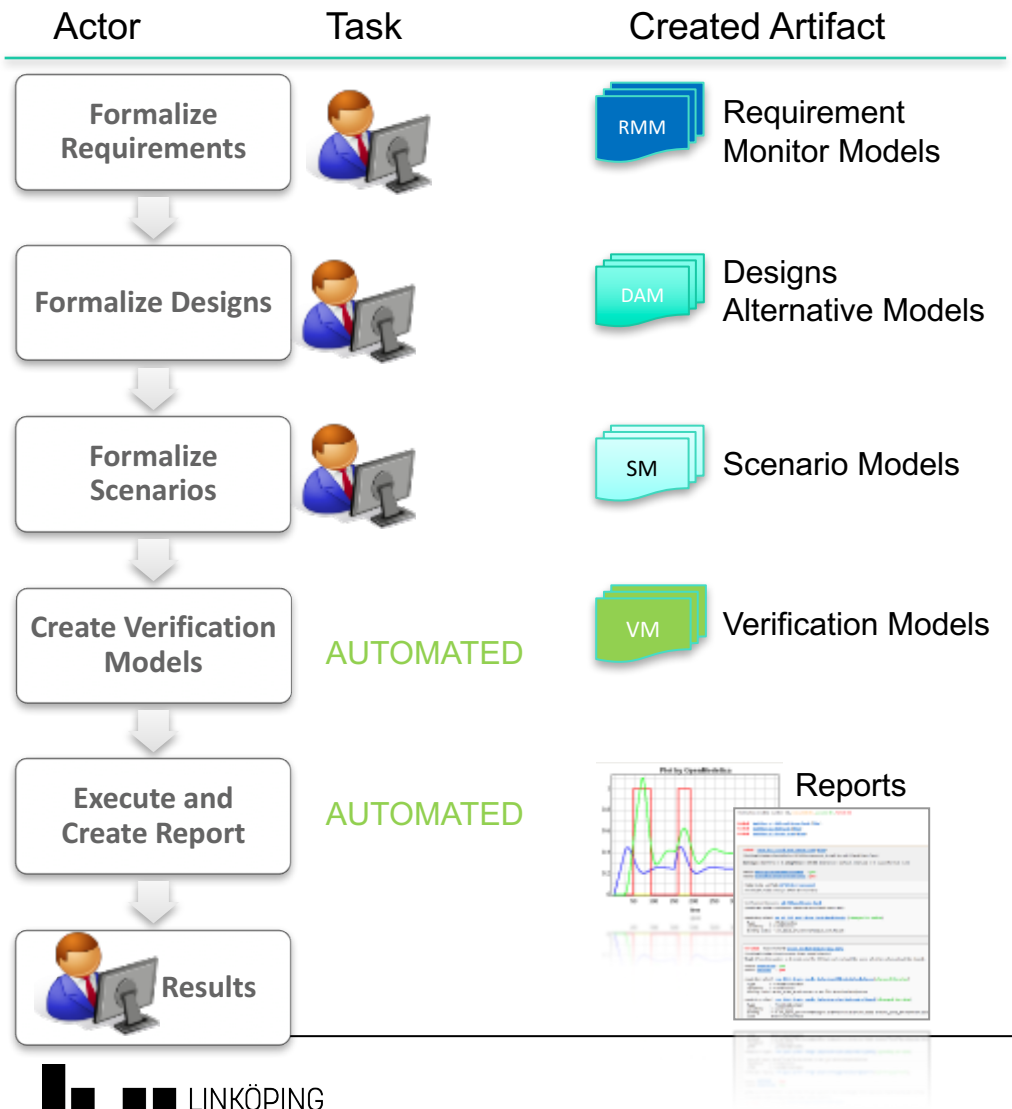
Dynamic Requirement Evaluation

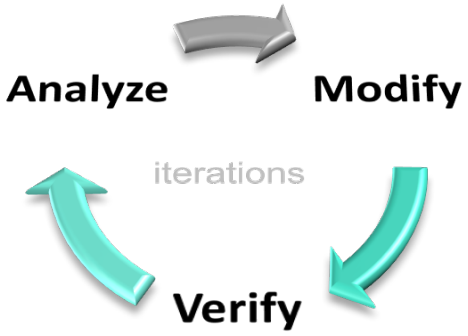


Dynamic Requirement Verification wrt System Design

- **Formalized Requirements** that should be verified
- **System Model, i.e., Design Alternative Model,** for which the requirements should be verified
- **Application scenarios** with the system model for which the requirements should be verified

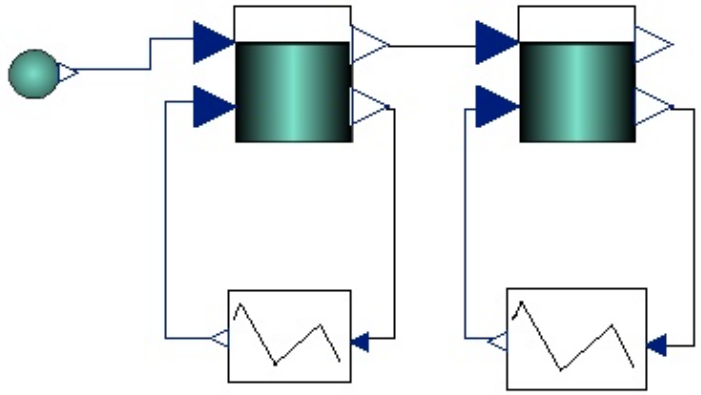
vVDR Method – virtual Verification of Designs vs Requirements





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

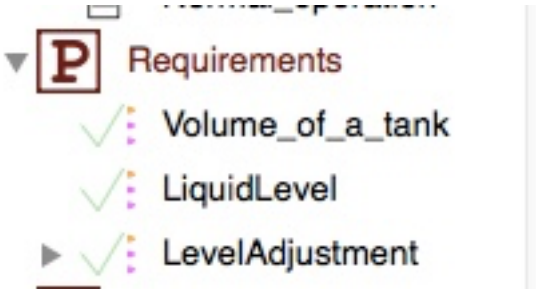
Two tank system



System model: Two connected tanks with controllers

System requirements: reminder

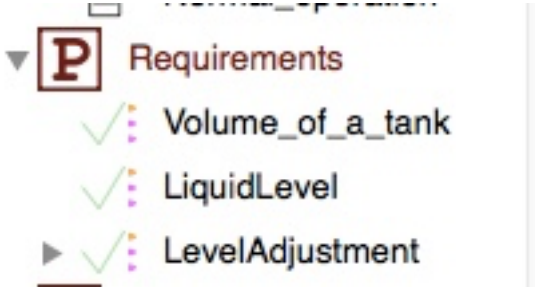
- Req. 001: The volume of each tank shall be at least 2 m3.
- Req. 002: The level of liquid in a tank shall never exceed 80% of the tank height.
- Req. 003: After each change of the tank input flow, the controller shall, within 20 seconds, ensure that the level of liquid in each tank is equal to the reference level with a tolerance of ± 0.05 m.

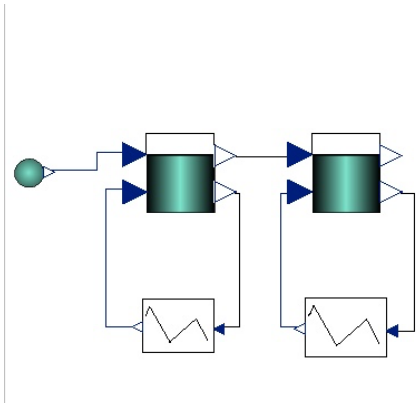


Scenarios

- In which conditions do we want to test the system?
  - Eg. Constant flow, flow change, rapid flow change

How do we combine these elements for verification?

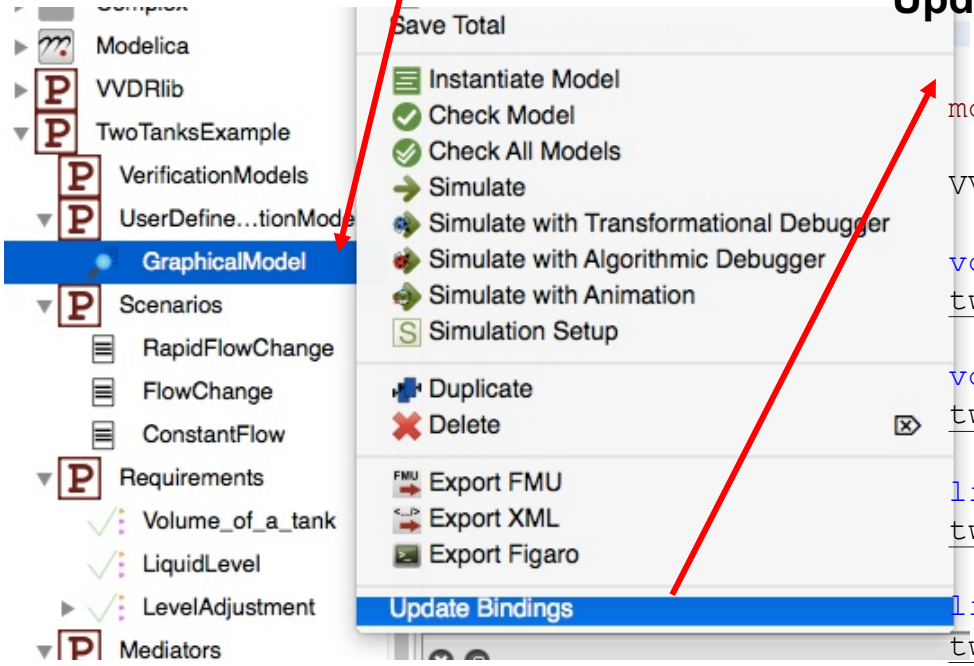




Automatic model composition

Initial AnalysisModel:

```
model GraphicalModel
  extends VVDRlib.Verification.VerificationModel;
  Requirements.Volume_of_a_tank volume_of_a_tank1;
  Requirements.LiquidLevel liquidLevel1;
  Design.TwoTanksDesign;
end GraphicalModel;
```



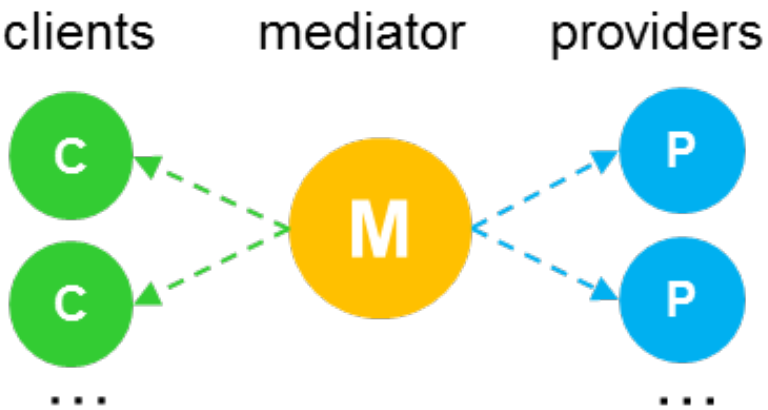
Updated AnalysisModel model with a binding:

```
model GraphicalModel
  extends
VVDRlib.Verification.VerificationModel;
  TwoTanksExample.Requirements.Volume_of_a_tank
volume_of_a_tank1_autogen_bind_0(tankVolume =
twoTanksDesign1.tank1.volume);
  TwoTanksExample.Requirements.Volume_of_a_tank
volume_of_a_tank1_autogen_bind_1(tankVolume =
twoTanksDesign1.tank2.volume);
  TwoTanksExample.Requirements.LiquidLevel
liquidLevel1_autogen_bind_0(waterLevel =
twoTanksDesign1.tank1.levelOfLiquid);
  TwoTanksExample.Requirements.LiquidLevel
liquidLevel1_autogen_bind_1(waterLevel =
twoTanksDesign1.tank2.levelOfLiquid);
  TwoTanksExample.Design.TwoTanksDesign
twoTanksDesign1;
end GraphicalModel;
```

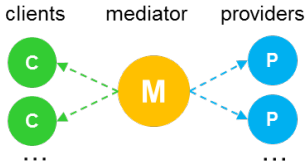
Binding:

client instance reference = binding expression

A **binding** is a causal relation which specifies that, at any simulated time, the value given to the referenced client instance shall be the same as the value computed by the right-hand expression.



How do we capture the information necessary for automatic generation?



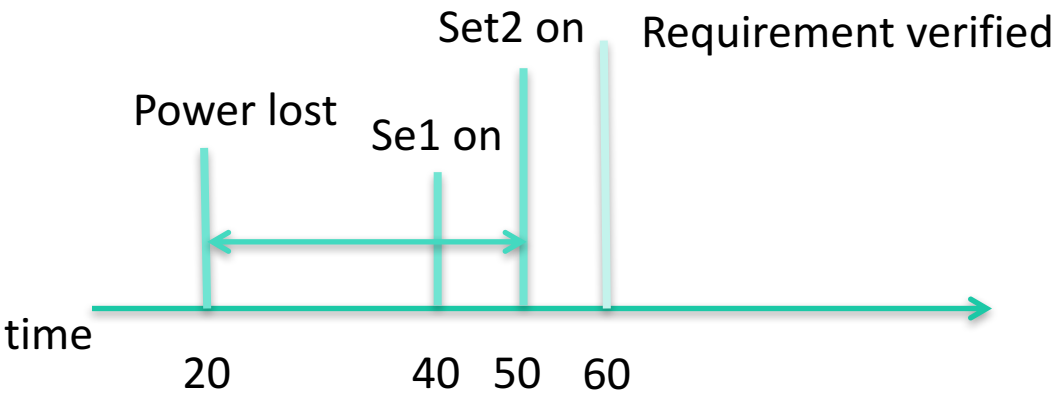

```
record volumeLevel
  extends  Mediator(mType = "Real",
  clients = {Client(modelID =
"TwoTanksExample.Requirements.Volume_of_a_tank",
component = "tankVolume")});
end volumeLevel;
```

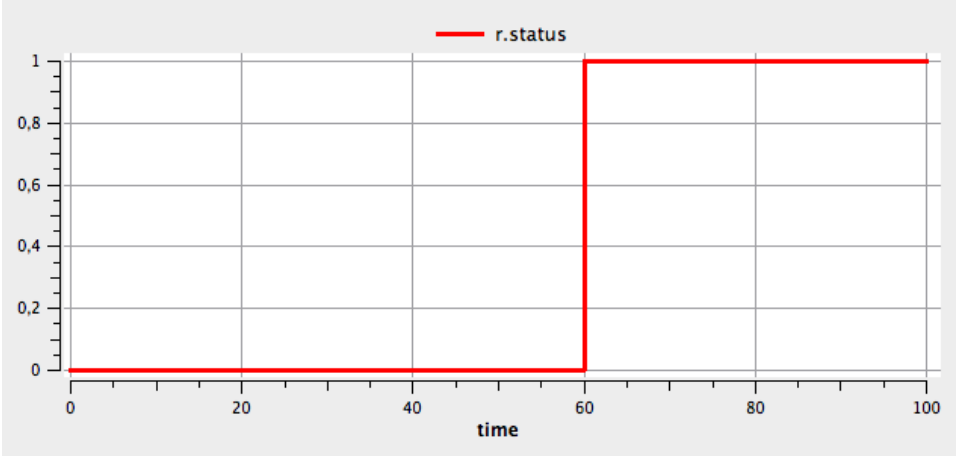
```
record volumeLevel
  extends  Mediator(mType = "Real",
  clients = {Client(modelID =
"TwoTanksExample.Requirements.Volume_of_a_tank",
component = "tankVolume")},
  providers = {Provider(modelID =
"TwoTanksExample.Design.Components.Tank",
    template = "%getPath.volume")});
end volumeLevel;
```

Verification example 2

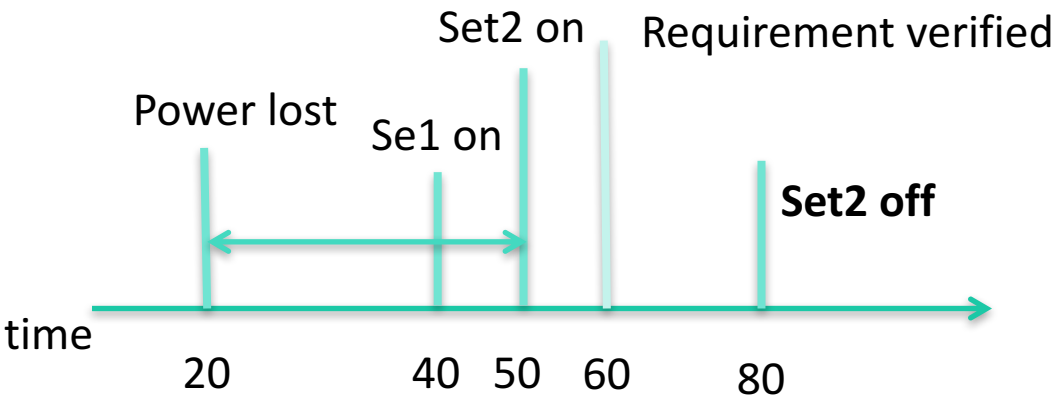
```
block R2 "after the power is lost, at least 2 powerSets
must be on within 40s"
        extends Requirement;
    input Boolean[5] isOn;
    input Boolean powerLoss;
    output Integer status(start = 0);
    Boolean wA;
    equation
        wA = withinAfter(40, powerLoss);
        when wA then
          status = if countTrue(isOn) >= 2 then
not_violated else violated;
        elsewhen not wA then
           status = undefined;
        end when;
end R2;
```

Scenario 1

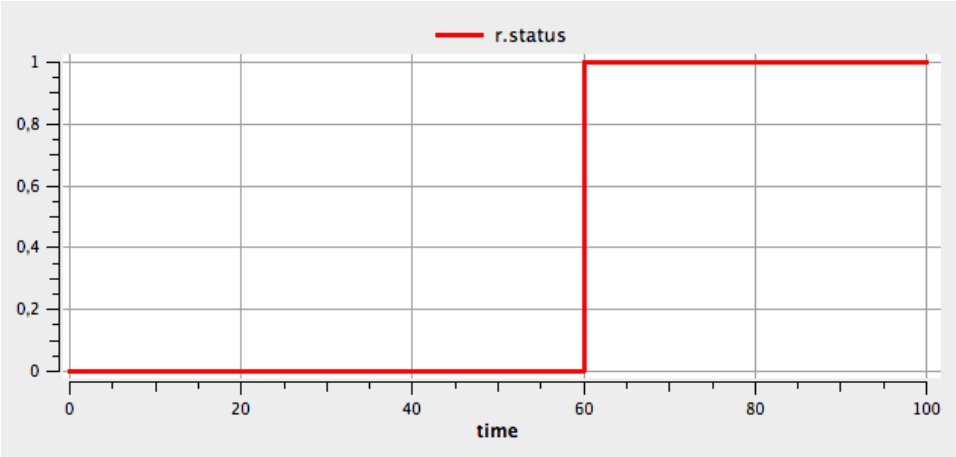

Set2 on
Requirement verified
Power lost
Se1 on
time
20
40
50
60


r.status
1
0,8
0,6
0,4
0,2
0
0
20
40
60
80
100
time

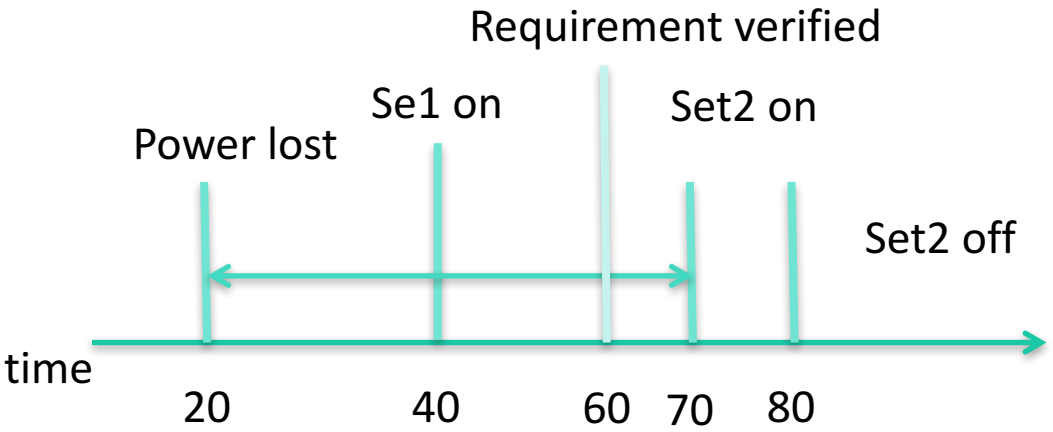
Scenario 1

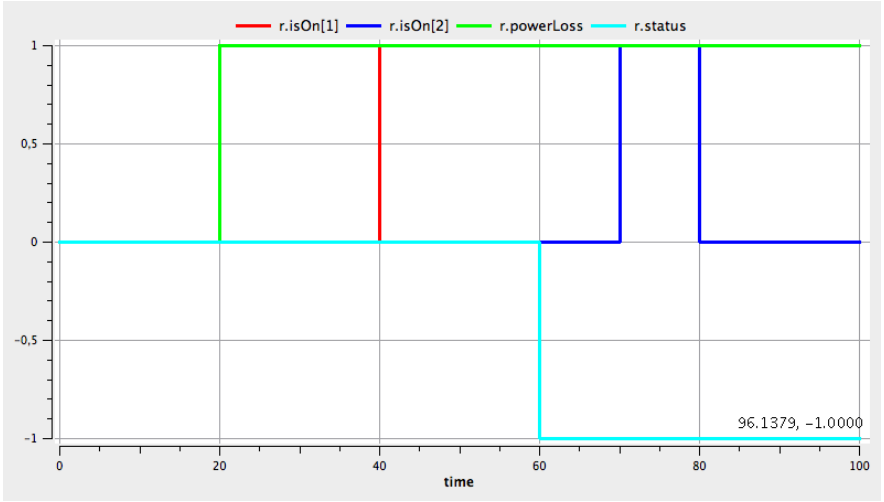


Attention!



Scenario 2

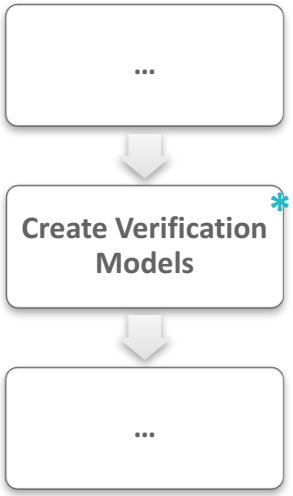

Requirement verified
Se1 on
Set2 on
Power lost
Set2 off
time
20
40
60
70
80

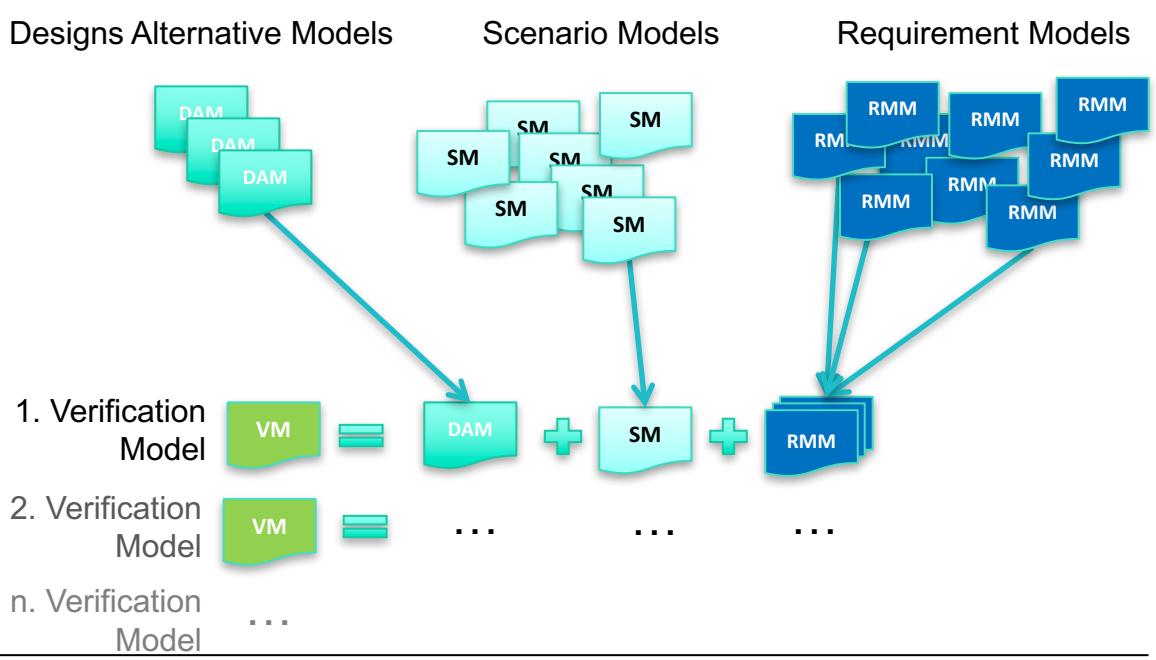

r.isOn[1]
r.isOn[2]
r.powerLoss
r.status
1
0,5
0
-0,5
-1
96.1379, -1.0000
0
20
40
60
80
100
time

Next step

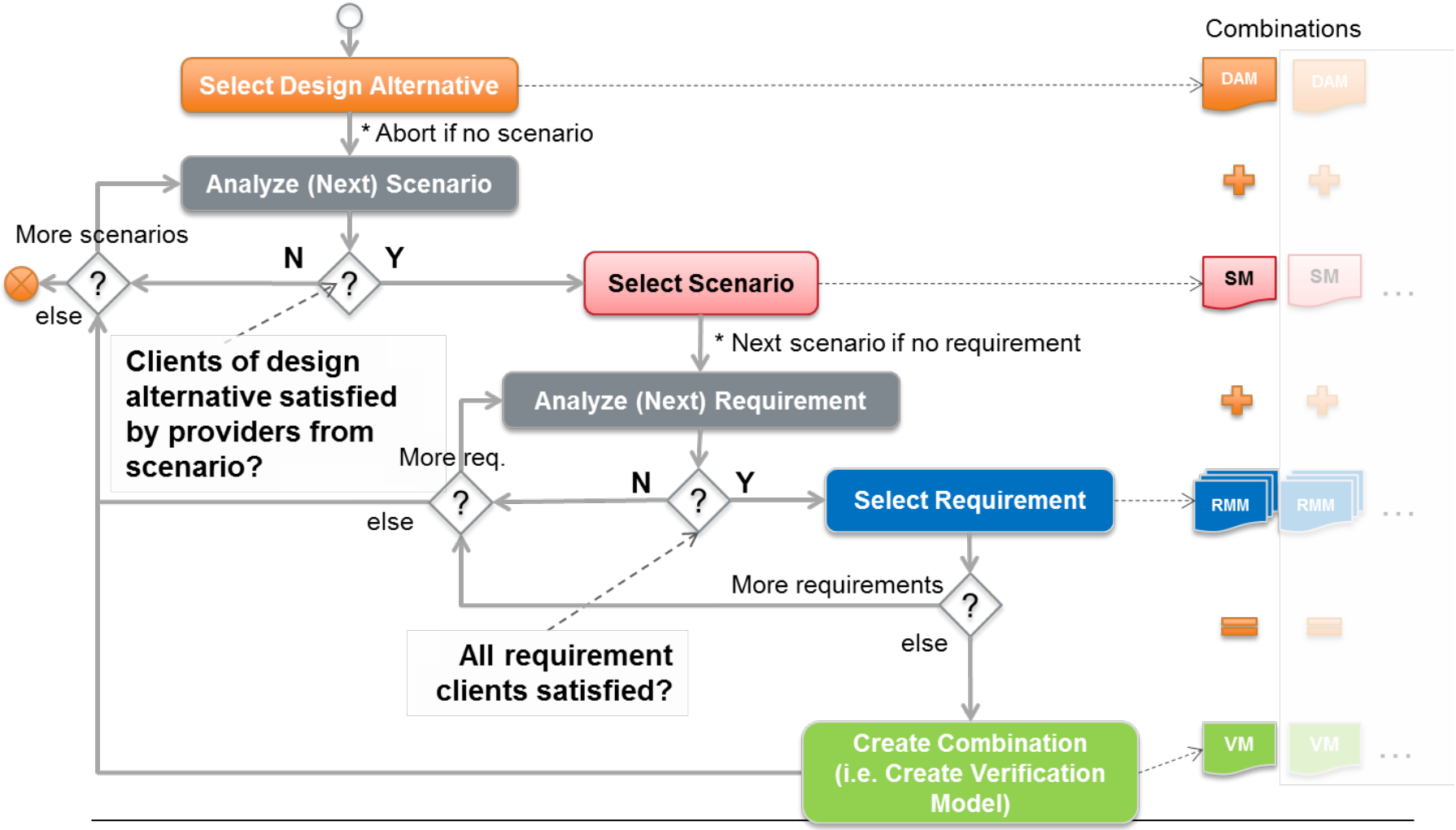
We want to verify **different design alternatives** against **sets of requirements** using **different scenarios**. Questions:

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**?





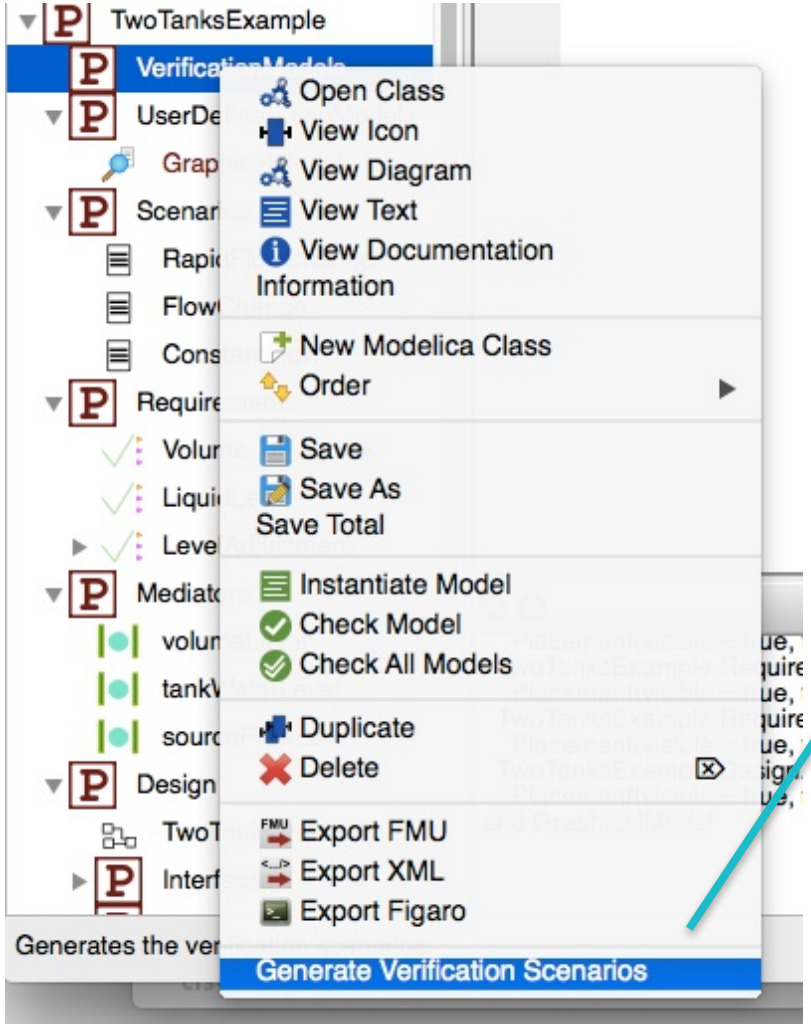
Generating/Composing Verification Models



Composing Verification Models

- 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


TwoTanksExample
Open Class
View Icon
View Diagram
View Text
View Documentation
Information
New Modelica Class
Order
Save
Save As
Save Total
Instantiate Model
Check Model
Check All Models
Duplicate
Delete
Export FMU
Export XML
Export Figaro
Generate Verification Scenarios


VerificationModels
verif_mo...utogen_2
verif_mo...utogen_1
verif_mo...utogen_0

Auto generated model

```
within TwoTanksExample.VerificationModels;

model verif_model_autogen_2 "Autogenerated verification model"
  TwoTanksExample.Requirements.Volume_of_a_tank
_agen_Volume_of_a_tank4_autogen_bind_0(tankVolume =
_agen_TwoTanksDesign1.tank1.volume);
  TwoTanksExample.Requirements.Volume_of_a_tank
_agen_Volume_of_a_tank4_autogen_bind_1(tankVolume =
_agen_TwoTanksDesign1.tank2.volume);
  TwoTanksExample.Requirements.LiquidLevel
_agen_LiquidLevel3_autogen_bind_0(waterLevel =
_agen_TwoTanksDesign1.tank1.levelOfLiquid);
  TwoTanksExample.Requirements.LiquidLevel
_agen_LiquidLevel3_autogen_bind_1(waterLevel =
_agen_TwoTanksDesign1.tank2.levelOfLiquid);
  TwoTanksExample.Requirements.LevelAdjustment
_agen_LevelAdjustment2_autogen_bind_0(inFlow =
_agen_TwoTanksDesign1.tank1.levelOfLiquid);
  TwoTanksExample.Requirements.LevelAdjustment
_agen_LevelAdjustment2_autogen_bind_1(inFlow =
_agen_TwoTanksDesign1.tank2.levelOfLiquid);
  TwoTanksExample.Design.TwoTanksDesign _agen_TwoTanksDesign1(source.flowLevel =
_agen_RapidFlowChange0.flowLevel);
  TwoTanksExample.Scenarios.RapidFlowChange _agen_RapidFlowChange0;
end verif_model_autogen_2;
```

To + Up:

- Requirement modeling is a methodology NOT a tool
- How to model requirement?
  - State machines, standard Modelica, libraries
- How to use the requirement models?
  - Composition methodology, tool-support
- Next step
  - Batch analysis of results, report generation…

That’s all! Thank you! Questions?

www.liu.se

LINKÖPING UNIVERSITY