Specifying and designing a test mean alla Capella
A typical test means design process ...

Document-based process ... with typical pains
Specify the Test Means with a strong link to
- the VV Objectives & VV strategy
- the architecture of the system

Expected benefits
- **Agility**: allow to adapt the VV strategy to
  - feedbacks from planning of deliveries & risk analysis
  - design changes
- **Lead time**:
  - start the development of the test means earlier (Logical)
  - Build the right means : no more / no less
- **Productivity**: work directly on the source information
- **Quality**: less manual error prone activities and interpretations
Find the limit and balance between

• what shall be implemented in Capella
• what shall be implemented in a dedicated tool

Guidance:

• Use viewpoints when the additional data shall be consolidated with architecture data in a short-loop
• Use export when the additional data are loosely coupled
The solution

V&V Objectives Viewpoint

Identify detailed V&V objectives related to the sub-system of interest

Contribute to the V&V plan

Identify the Units under Test

Organize VVO into TestRequests

Request a Test Mean

Identify High Level V&V objectives

Collect TestSpec Request

Allocate Unit under Test to a Test Mean

Allocate TestRequest to a TestMean

Specify the requirements of a Test Facility

Test Mean Spec / Design Viewpoint

Review the Facility specifications

Specify the requirements of a Simulation Product
**V&V Objectives Viewpoint**

- **V&V Plan**: Identify the Units under Test
  - **Properties**: verif type, category, ...

- **Test Request**: Makes the link between one UuT and a set of consistent VVOs
  - **Properties**: id, owner, expected results, ...

- **Logical Component**: Is allocable to a Test Mean

- **Idem Physical**
Demo
TestMean Spec & Design Viewpoint

Makes the distinction btw real / sim

Support TestRequests for multiple UuT
Demo
Describe the System to be simulated
- Functional and organic architecture
- Real interfaces
- Malfunctions / probes

Specify the simulation
- Identify simulation models
- Specification of the simulation models
  - Functional scope & fidelity
  - Interfaces

Design the simulation
- Design the simulation execution platform
  - Infrastructure
  - Runtime Environment
- Select the models & allocate to Runtime

Integrate & deploy simulation
- Receive simulation models: verif & patch
- Integrate simulation models
  - Connections
    - Format / deformat mgt
    - Automatic algo & scripts
  - Initialisation
- Deploy: configuration generation

System Architecture
V&V Strategy
- V&V Objectives
- V&V Plans
- Test Means identification

Test Mean Spec
- Functional scope definition
- Allocation to Simulation

Design simulation models
- Integration of building blocks
- Modeling authoring tools bootstrap
- Compliance checking

A bigger picture with Citrus
Citrus – simulation design

System parts allocated to simulation models

---

**FMI Internal**

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Causality Type</th>
<th>Variability Type</th>
<th>Can Handle Mu</th>
<th>Initial Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>nand1.y</td>
<td>Boolean output signal</td>
<td>Modelica.Blocks...</td>
<td>local</td>
<td>discrete</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>booleanTable.table[2]</td>
<td>Vector of time points. At every tim...</td>
<td>Real [fmiTypes]...</td>
<td>parameter</td>
<td>fixed</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>and1.u[1]</td>
<td>Vector of Boolean input signals</td>
<td>Boolean [fmiTyp...</td>
<td>local</td>
<td>discrete</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>r5FlipFlop.pre.pre_u.start</td>
<td>Start value of pre(u) at in...</td>
<td>Boolean [fmiTyp...</td>
<td>fixed</td>
<td>fixed</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>set1.expr[2]</td>
<td>If y = u[1] then expr[i] else y defau...</td>
<td>Boolean [fmiTyp...</td>
<td>constant</td>
<td>fixed</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>booleanPulse1.period</td>
<td>Time for one period</td>
<td>Modelica.Simulat...</td>
<td>parameter</td>
<td>tunable</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>falling.y</td>
<td>Boolean output signal</td>
<td>Modelica.Blocks...</td>
<td>local</td>
<td>discrete</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>sampleTriggerSet.y</td>
<td>Connector of Boolean output signal</td>
<td>Modelica.Blocks...</td>
<td>local</td>
<td>discrete</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>xor1.y</td>
<td>Boolean output signal</td>
<td>Modelica.Blocks...</td>
<td>local</td>
<td>discrete</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>rising.u</td>
<td>Boolean input signal</td>
<td>Modelica.Blocks...</td>
<td>local</td>
<td>discrete</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>or2.y</td>
<td>Boolean output signal</td>
<td>Modelica.Blocks...</td>
<td>local</td>
<td>discrete</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>showValue3.showActive</td>
<td>Vector of Boolean input...</td>
<td>Modelica.Blocks...</td>
<td>local</td>
<td>discrete</td>
<td></td>
<td>exact</td>
</tr>
<tr>
<td>or1.u[1]</td>
<td>Vector of Boolean input signals</td>
<td>Modelica.FmiTyp...</td>
<td></td>
<td></td>
<td></td>
<td>exact</td>
</tr>
</tbody>
</table>
Citrus – simulation design
Backlog

• Transition to a dedicated project for TestMean design
• Malfunctions & Observability points
• VVO refinement & cascading
• Compliance btw TestRequest and TestMeans
• TestRequest & TestMeans versions
• Test procedures specifications
• Integration V&V Planning