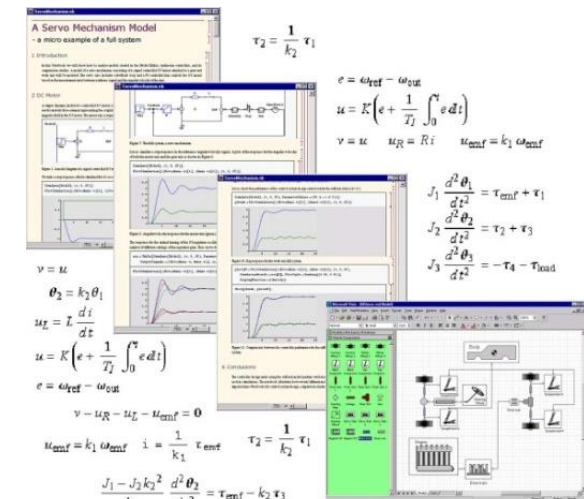
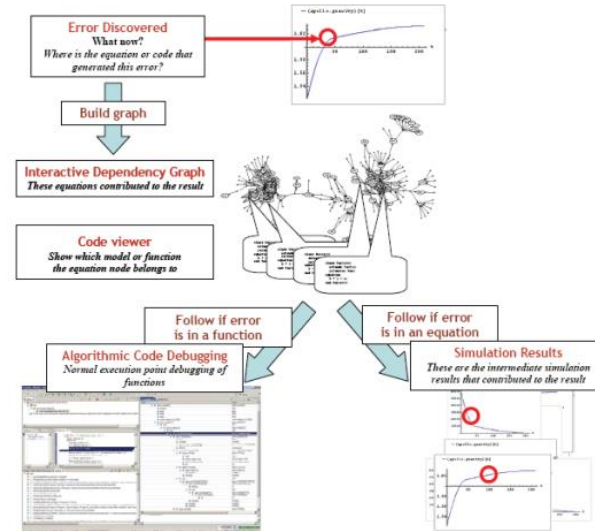
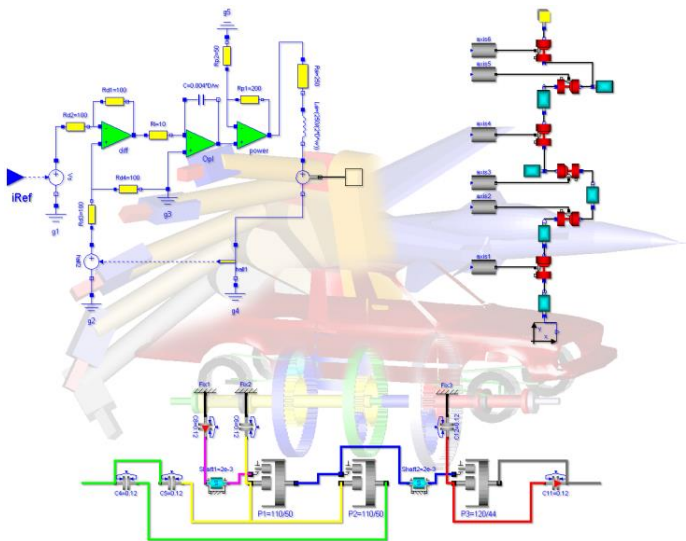


OpenModelica - The Common Requirement Modelling Language (CRML) Integration



Adrian Pop, Lena Buffoni, Audrey Jardin, Daniel Bouskela

2025-02-05

Open Source Modelica Consortium
PELAB, Linköping University
EDF, Électricité de France



- What is CRML
 - The Common Requirement Modelling Language
- CRML Tooling
 - The CRML Compiler
- CRML Integration
 - OMEdit & VSCode & Online
 - Status
- Future work

- The **C**ommon **R**equirement **M**odelling **L**anguage
 - Language for Verifying Realistic Dynamic Requirements
- Started at  **EDF** around 2006
- Further developed during the ITEA3  project

Ambition: Effective Engineering of Large CPS



Scope: Cyber-Physical Systems (CPS), especially energy systems

Characteristics

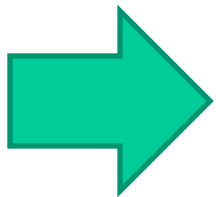
- CPS Projects have often strong **social and environmental impacts**
- They are **long lasting** projects involving numerous stakeholders
- They should obey to **multiple even conflicting requirements**
- **Project performance is a key** as large over costs may be induced quickly due to financial charges (discount rate)

Challenges

- How to focus on conceiving systems more sustainable, trusty and resilient?
- How to solve over-constrained problems? How to coordinate stakeholders efficiently?
- How to specify the right need without going into realization details?
How to reconcile innovation with what already exists?
- How to propagate changes in assumptions all over the system design cycle?
- How to evaluate design alternatives efficiently?
- How to perform failure modes, effects, and criticality analysis (FMECA) all along design lifecycle?
- How to justify and document design choices for future generations?

Examples of Challenges - Related to Energy Systems

- Interconnected systems with stringent physical constraints to ensure grid balancing
- Long system lifecycles: new solutions built on existing ones (they are not created from scratch)
- Compliance with strict safety and environmental rules
- Compliance with dependability and availability constraints (to ensure security of energy supply)
- Involvement of multiple stakeholders: clients, regulatory authorities, grid operators, energy providers, insurers, urban and land-use planning, plant operators..., with different and possibly contradictory objectives
- Moving context with increasing uncertainties (due to geopolitical tensions, energy market instabilities, climate change, lack of energy policy coordination between countries, evolution of demand wrt. new usages...)



Energy systems are globally over constrained.
New generation of methods & tools are needed to help engineers
**find the best compromise for covering multiple “what-if” operational
situations (incl. variabilities and hazards)**

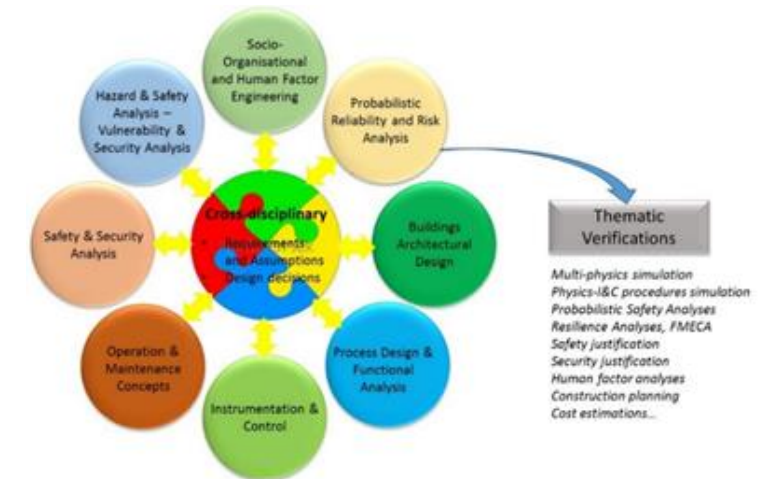
What Should Be Improved in CPS Engineering?

■ Today

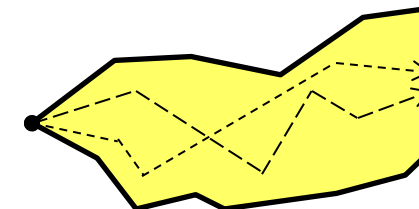
- **system evaluation** is performed mostly with **static models** (or dynamics are considered too late)
- **most verifications** are performed manually (or with domain-specific tools) and hence not as often as necessary
- **information is difficult to share** between disciplinary engineering teams

➔ oversizing, late error detections, and eventually delays and cost overruns

- There is a need for more rigorous engineering method to
 - **Be more effective assessing the impact of each solution** all along the system lifecycle including during preliminary design phases
➔ guide and justify design choices also for non-experts
 - **Open the solution space to innovative products or services**
➔ specify only “what is needed”



Figures:
T. Nguyen



Idea =

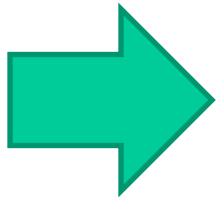
Use of **realistic dynamic behavioral models** to better handle multi-physics & systems' interactions → e.g. **Modelica**



Use of **formal dynamic requirement models** to automate verifications and evaluate multiple “what-if” scenarios → **CRML**

Rationale

- Consideration of “System Dynamics” as time may be part of new solutions to cover non-regular situations and hence source of cost reductions
- Formal verifications since for many CPS demonstration that the system operates safely is as important as the design itself

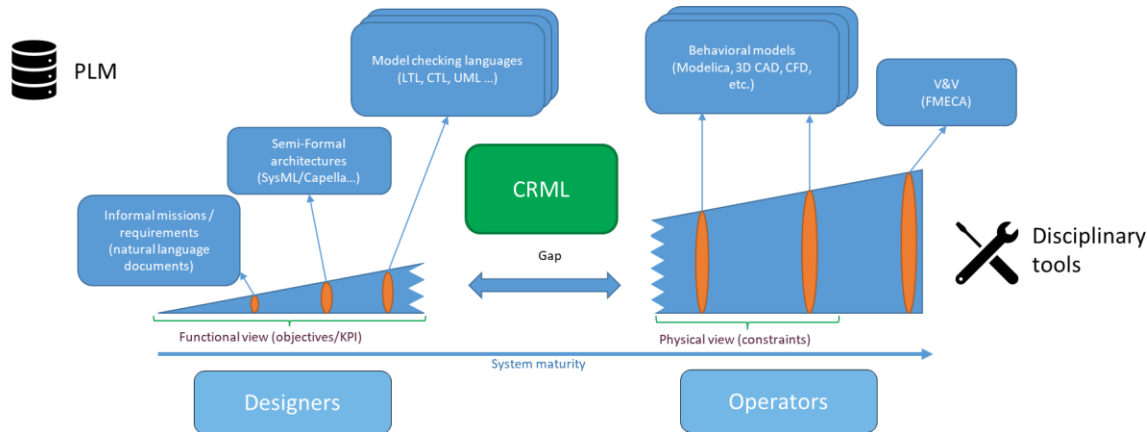


Scope of ITEA EMBrACE Project
“An enabler for making the best decisions at each step of the project cycle”

CRML: A Language for Verifying Realistic Dynamic Requirements

Why a new language?

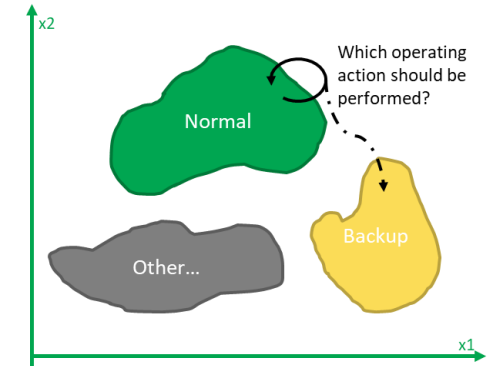
- Main principles from « System Engineering »
- Tools exist but are incomplete or essentially made for software design
- Native difficulty to address requirements that are « realistic » for systems with strong physical aspects
- In particular to study their dynamical interactions with their environments



CRML positioning vs. State-of-the-Art :
a bridge between the physical & the functional views

**A typical realistic
dynamical requirement
is multiple and stochastic**

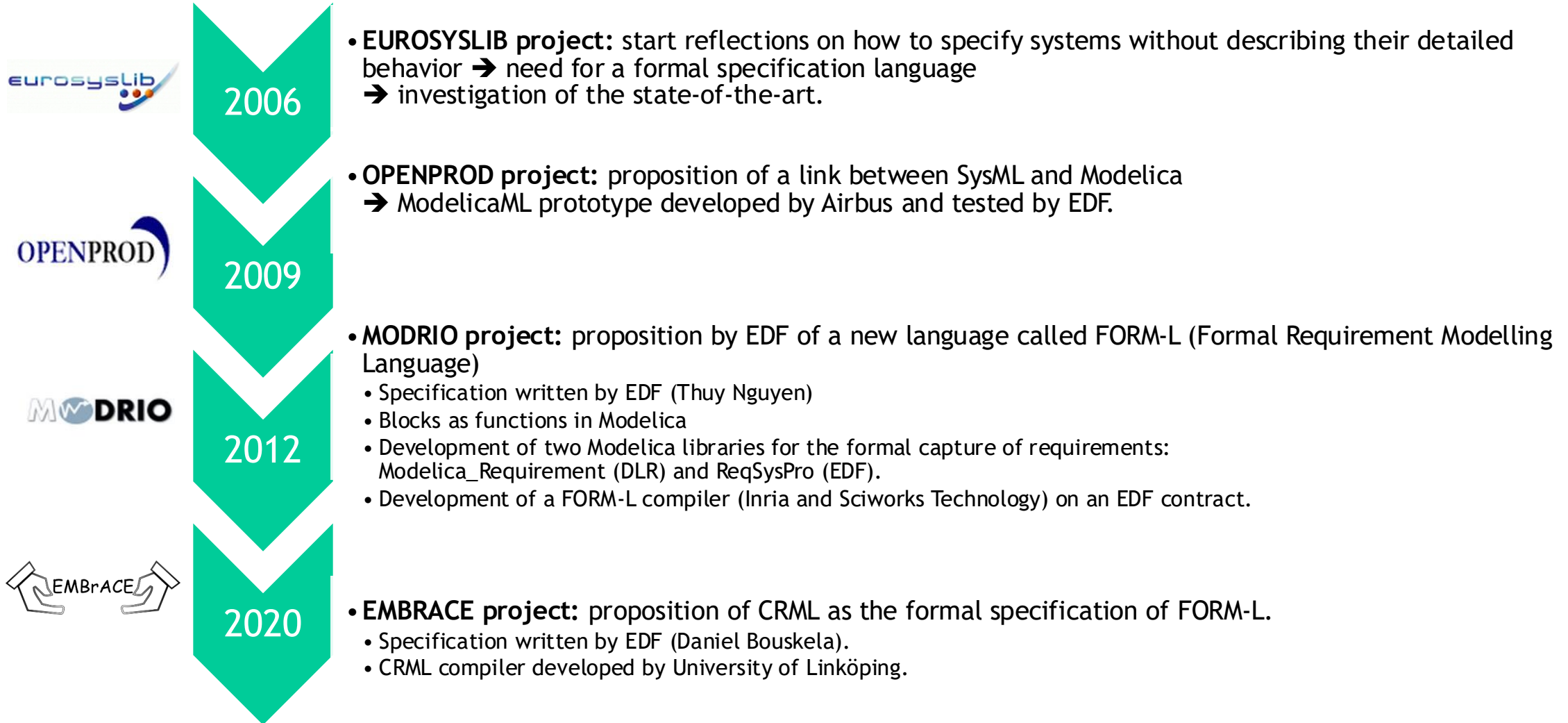
...



1. The system should stay within its normal operating domain.
2. If partial requirement 1 above fails, then the system should go back to its normal operating domain within a given time delay.
3. If partial requirement 2 above fails, or if partial requirement 1 fails with a too high failure rate, then the system should go to a safe backup state within a given time delay.
4. The complete requirement made of the conjunction of partial requirements 1, 2 and 3 should be satisfied with a given probability (e.g., > 99.99%).

**... and a typical project quickly sees its
complexity increase with the number of
requirements/stakeholders and evolution
over time**

CRML: a Long-Lasting History



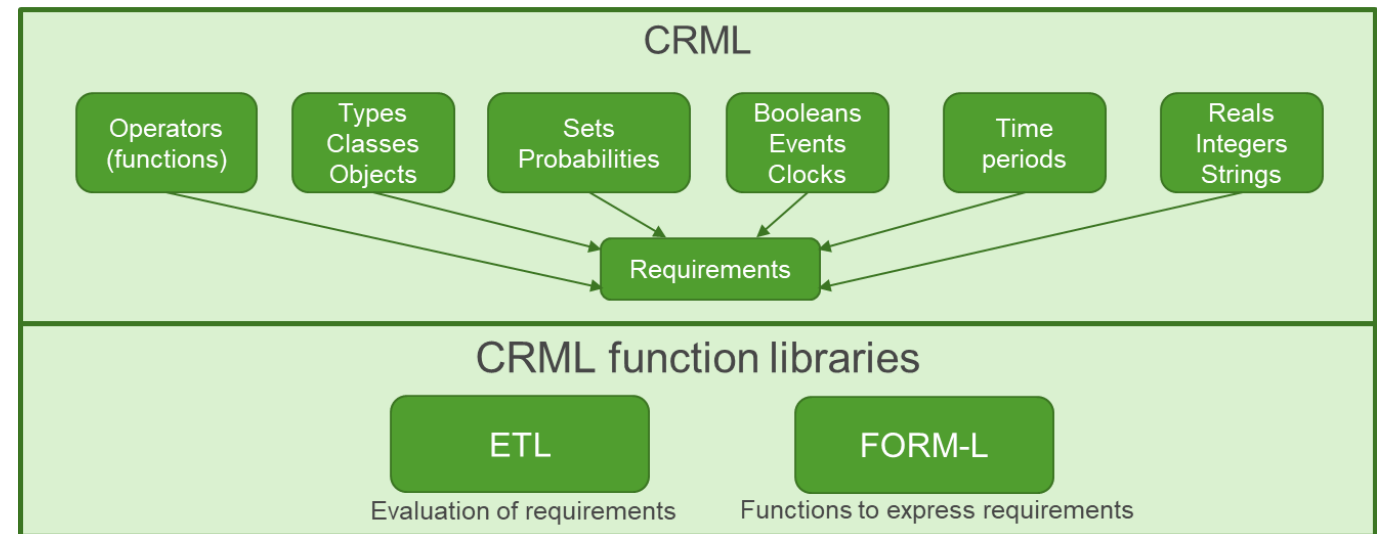
BASEECS 2025 - ITEA5 project, FPP phase
Behavioral Analysis and Simulation for Environmentally and Economically-sustainable
Co-Engineered Systems

How To Express a CRML Requirement?

R = [Where or Which] [When] [What] + (optional) [How well]

```
for all' pump 'in' system.pumps 'during' system.inOperation 'check count' (pump.isStarted 'becomes true') '<=' 3;  
'during' systemOperatingLife 'check at end' (estimator Probability (noStart at inOperation 'becomes false')) '>' 0.99;
```

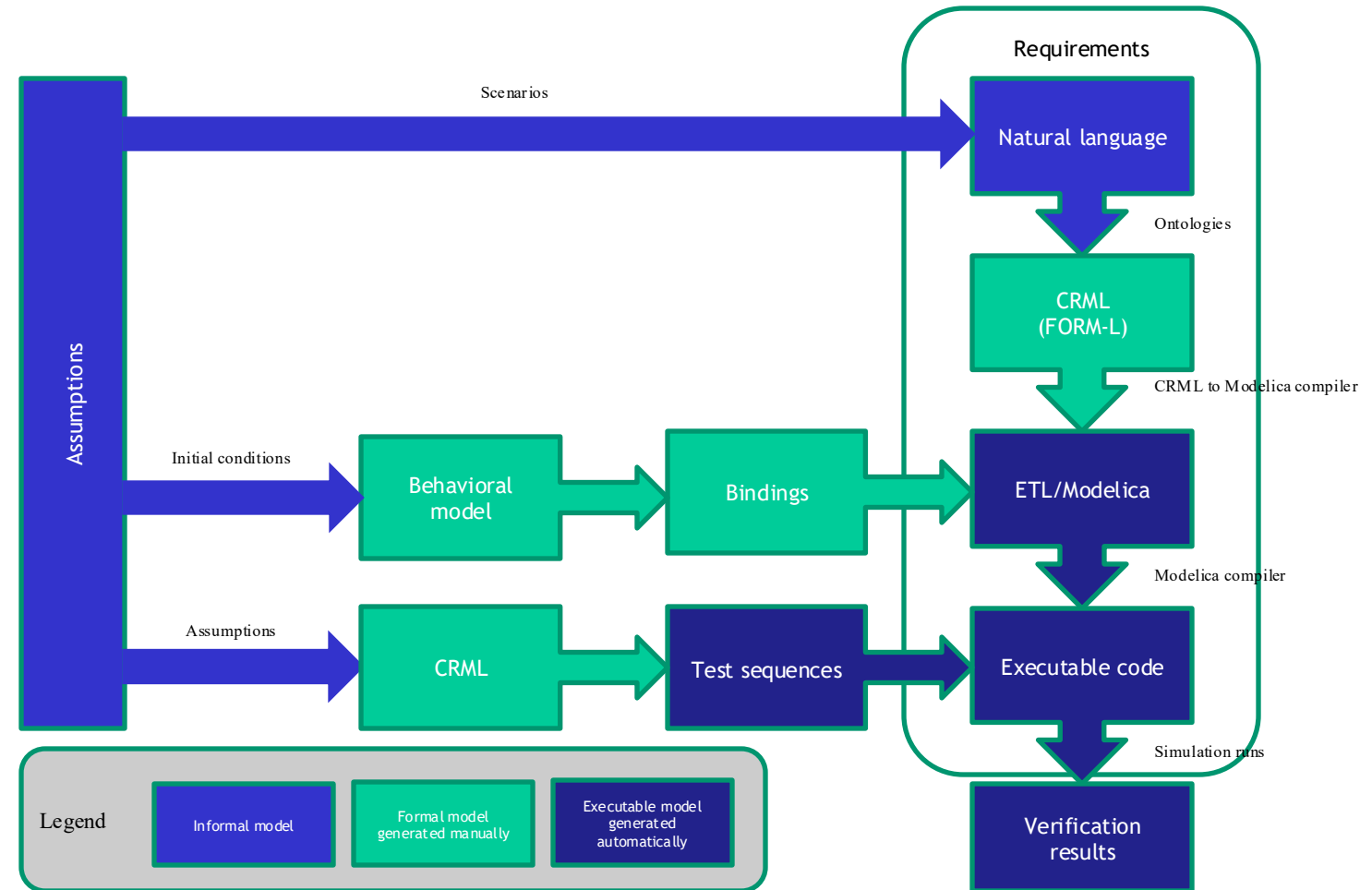
- Combination of 4 items
 - Spatial locators
 - Time locators
 - Condition to be checked
 - (optionally) Performance indicator
- Value at instant t is a Boolean⁴ which can be :
true, false, undefined
or undecided



CRML Specification v1.1 (EMBrACE D2.1, Daniel Bouskela)

How to Use CRML for Verifications?

- **Requirement models** to capture all constraints on the system and define envelopes of acceptable behaviors
- **Behavioral models** to capture the behavior of design solutions
- **Verification models** to automate tests by using requirement models as observers to check whether design solutions meet requirements or not.



How To Evaluate a CRML Requirement?

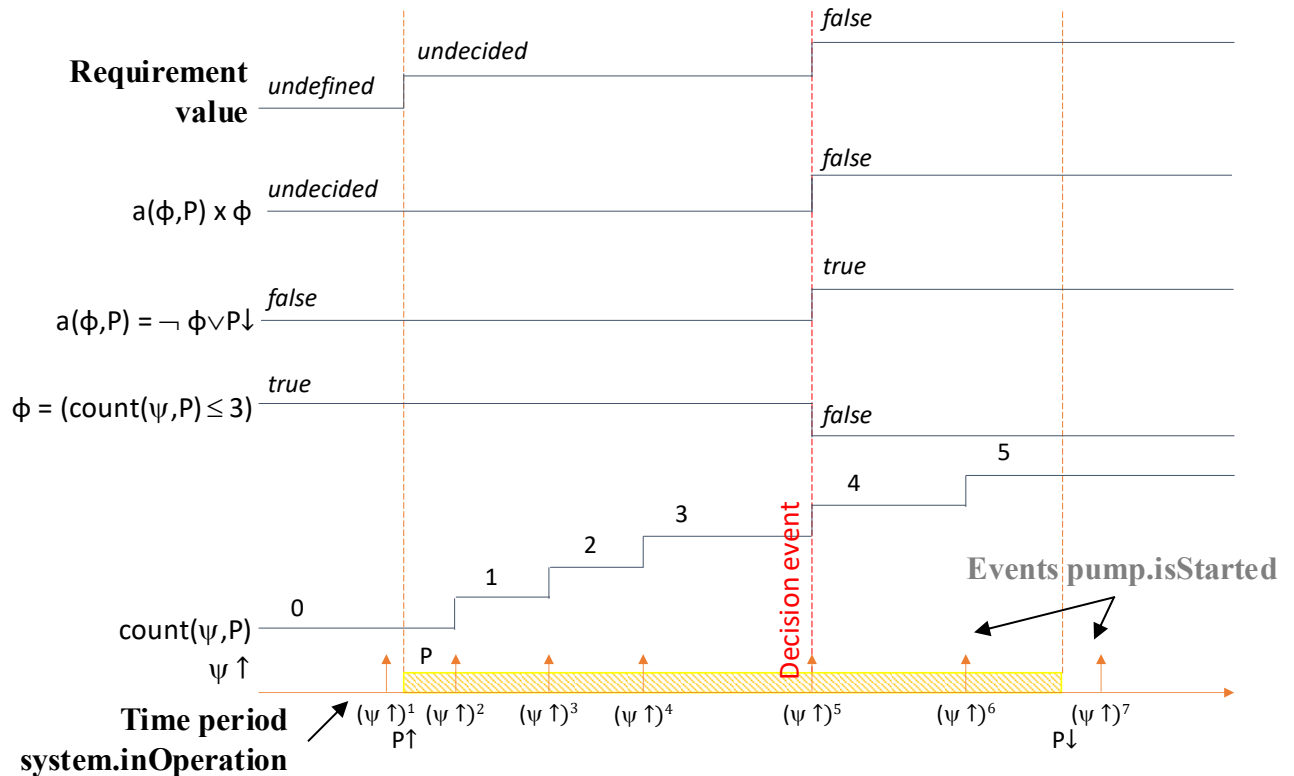
Case 1: Requirement R3 is declared as « violated » as soon as condition φ becomes false

Requirement capture in CRML

```
Class Pump is {
    Boolean isStarted is external};
Class System is {
    Pump{} pumps is external;
    Boolean inOperation is external};
System system;

Requirement R3 is {
    'for all' pump 'in' system.pumps
    'during' system.inOperation
    'check count' (pump.isStarted 'becomes true')
    '<=' 3;
};
```

external keyword is used to retrieve values in solution models
Operators in “ ” are defined by user to improve readability



Requirement evaluation
via observation of system behavioral dynamics

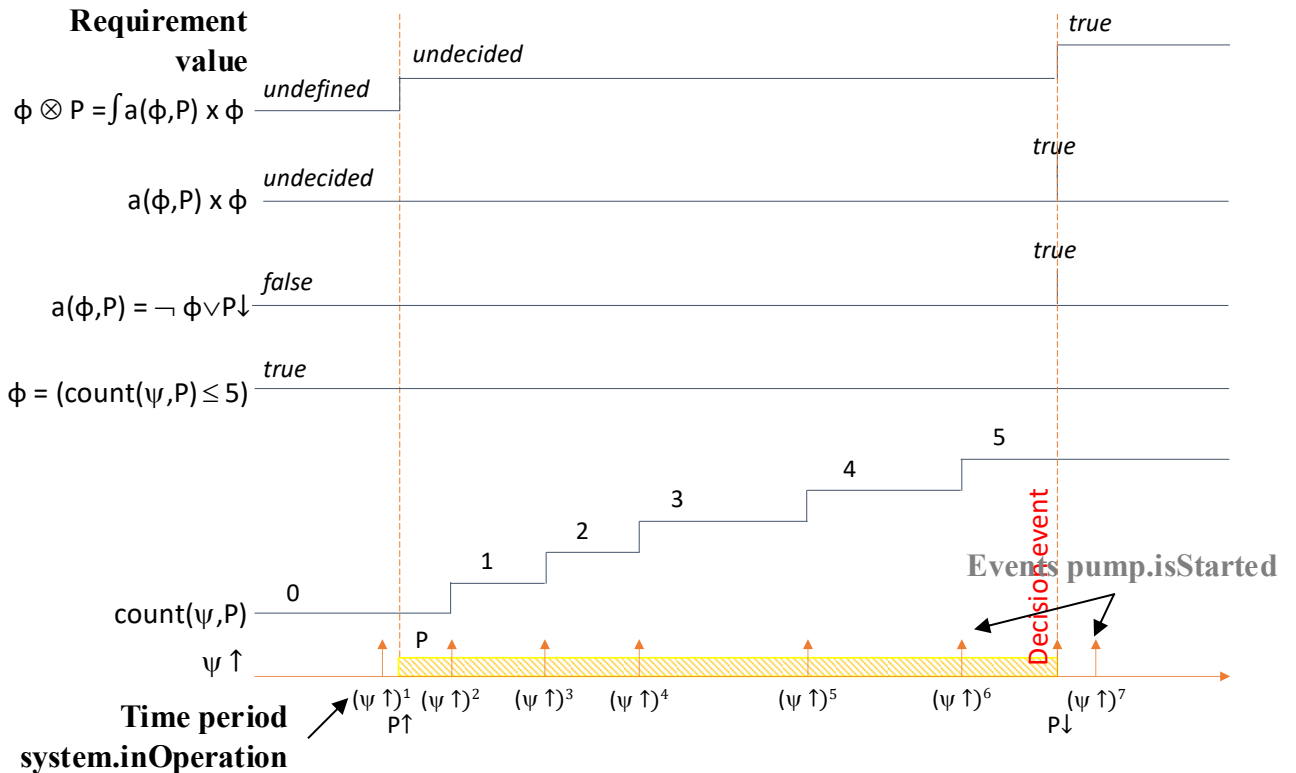
How To Evaluate a CRML Requirement?

Case 2: Requirement R5 is declared as
« undecided » until time period is completed

Requirement capture in CRML

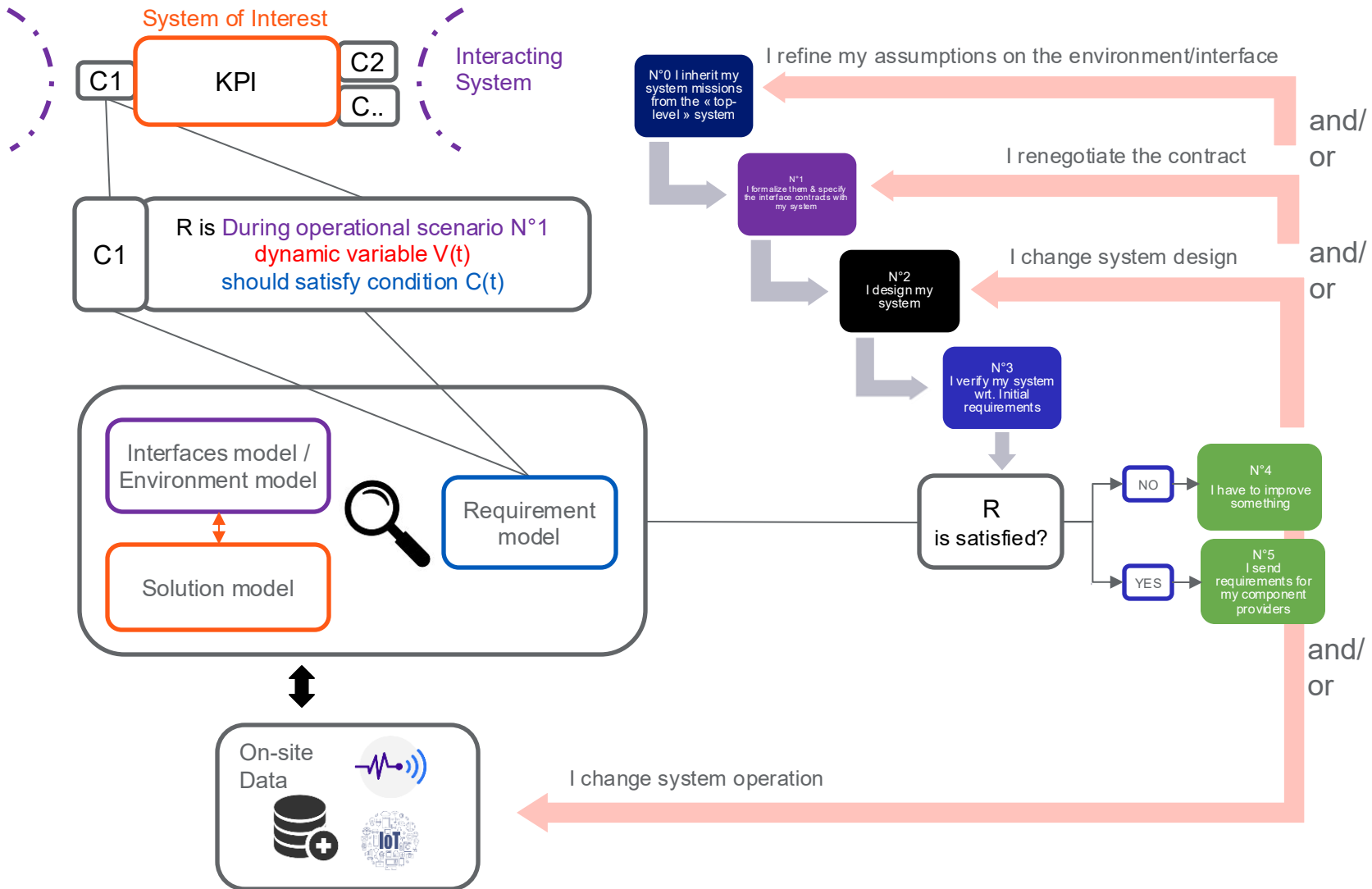
```
Class Pump is {
    Boolean isStarted is external};
Class System is {
    Pump{} pumps is external;
    Boolean inOperation is external};
System system;

Requirement R5 is {
    'for all' pump 'in' system.pumps
    'during' system.inOperation
    'check count' (pump.isStarted 'becomes true')
    '<=' 5;
};
```



Requirement evaluation
via observation of system behavioral dynamics

How to Use CRML As a Decision Tool?



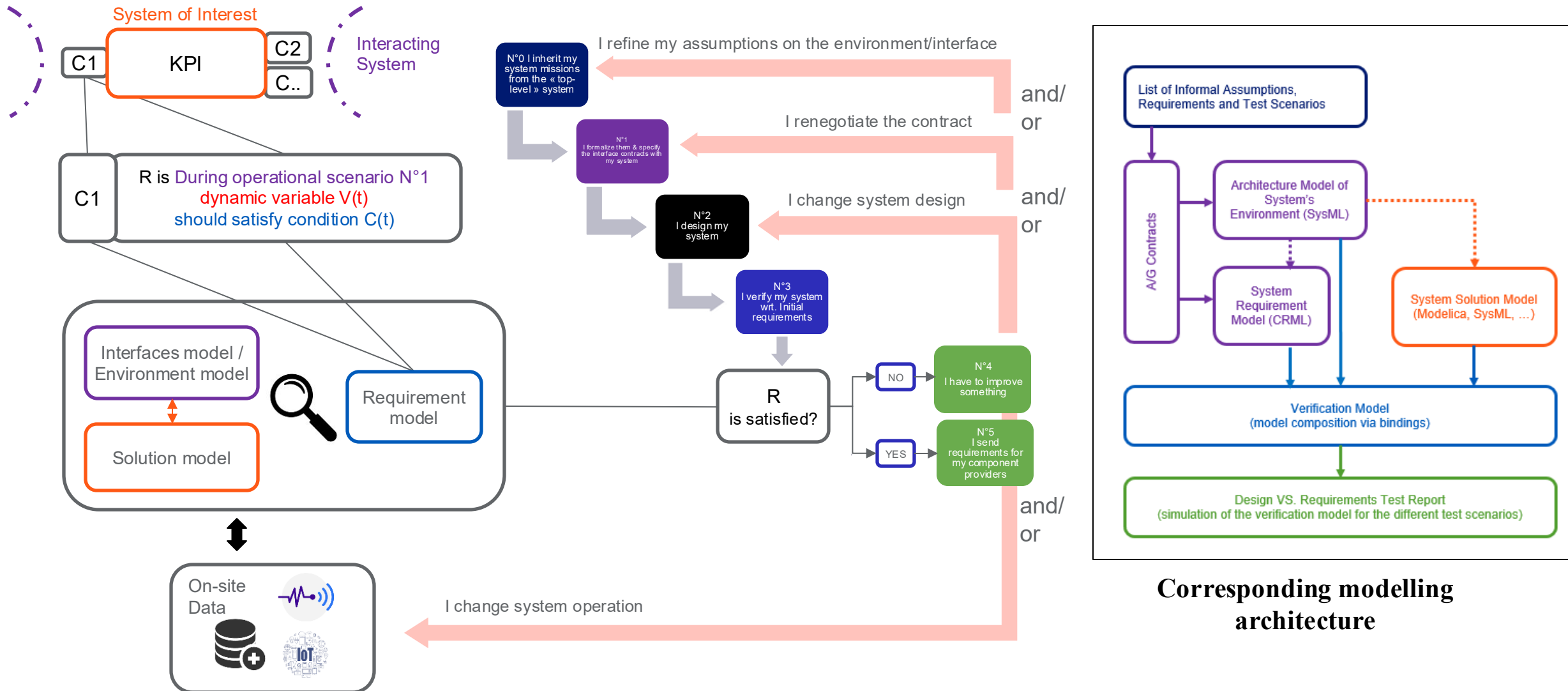
Model to support complexity

- Scope of responsibility of stakeholders
- Multiplicity of constraints and operating scenarios
- Dynamics of interactions between systems, human and environment

Center development on the requirements

- Evaluate the impact of each solution on your overall ambition
- Design only for the « right » need
- Adapt the studies to « what is just needed »
- All along the project
- And according to the data available at instant T

How to Use CRML As a Decision Tool?



- What is CRML
 - The Common Requirement Modelling Language
- CRML Tooling
 - The CRML Compiler
- CRML Integration
 - OMEdit & VSCode & Online
 - Status
- Future work



- The CRML compiler

- <https://github.com/lenaRB/crml-compiler/>
- Implemented in Java
- Translates CRML to Modelica
- Integrates with Unit testing and Reporting

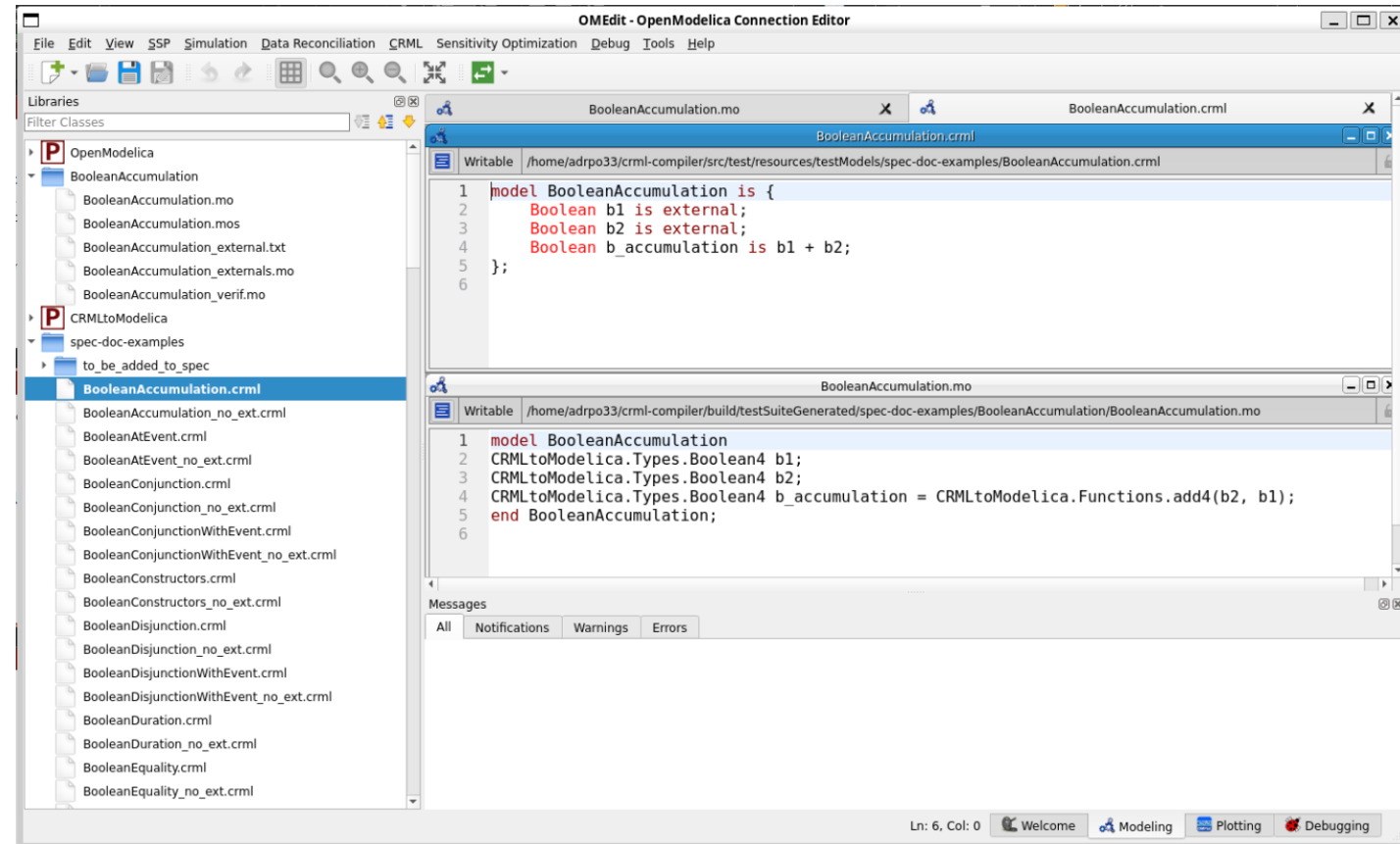
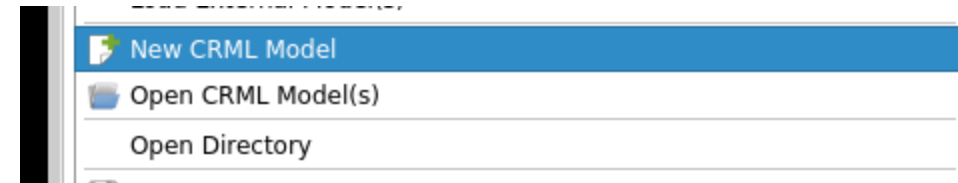
- Ongoing work

- Support the full CRML specification
- Graphical notation for CRML and support in OpenModelica GUI
- Continue to improve the integration with OpenModelica GUI

- CRML menus in OMEdit activated by Tools->Options->General->Enable CRML support
 - Generate and load Modelica code (also via the Library/File browser, right click)
 - Call the CRML compiler on the opened CRML file, generate Modelica code, load it into OMEdit, give errors if the code cannot be loaded
 - Dialog for CRML configuration before compilation
 - Set the name of the generated Modelica file, the package name, etc
 - Future
 - annotation in the CRML file where one can provide a configuration
 - Modelica annotation in the generated Modelica file

- Run test suite
 - Select a directory with CRML files
 - Call the CRML tool to generate the html report
 - Load and display the html test report
 - A CRML test will go through these phases
 - Parsing
 - Translation
 - Verification model generation
 - Execution
 - Result Verification

- New / Open CRML models
- Load directories containing CRML models
- Syntax Highlighting



Generate and Simulate Modelica code

OMEdit - OpenModelica Connection Editor

File Edit View SSP Simulation Data Reconciliation CRML Sensitivity Optimization Debug Tools Help

Libraries

Filter Classes

- OpenModelica
 - BooleanAccumulation
 - BooleanAccumulation.mo
 - BooleanAccumulation.mos
 - BooleanAccumulation_external.txt
 - BooleanAccumulation externals.mo
 - BooleanAccumulation_verif.mo
 - CRMLtoModelica
 - spec-doc-examples
 - CRML
 - UsersGuide
 - TimeLocators
 - Requirements
 - Blocks
 - ETL
 - Examples
 - Tests
 - Modelica
 - Complex
 - ModelicaServices
 - BooleanAccumulation
 - CRML_test
 - CRML_test.Spec_doc.Boole...eanAccumulation externals
 - CRML_test.Spec_doc.Bo...eanAccumulation_verif**

Plot : 1

Grid Log X Log Y

time (s)

Variables

Filter Variables

Simulation Time Unit s

Time: 0 Speed: 1

Variables	Value	Display Uni	Description
(Activ..._verif)			
<input checked="" type="checkbox"/> b1	4		
<input checked="" type="checkbox"/> b2	2		
<input checked="" type="checkbox"/> b_accumulation	4		
externals			

Messages

All Notifications Warnings Errors BooleanAccumulation_verif X

100% Cancel Simulation Open Output File

Compilation Output

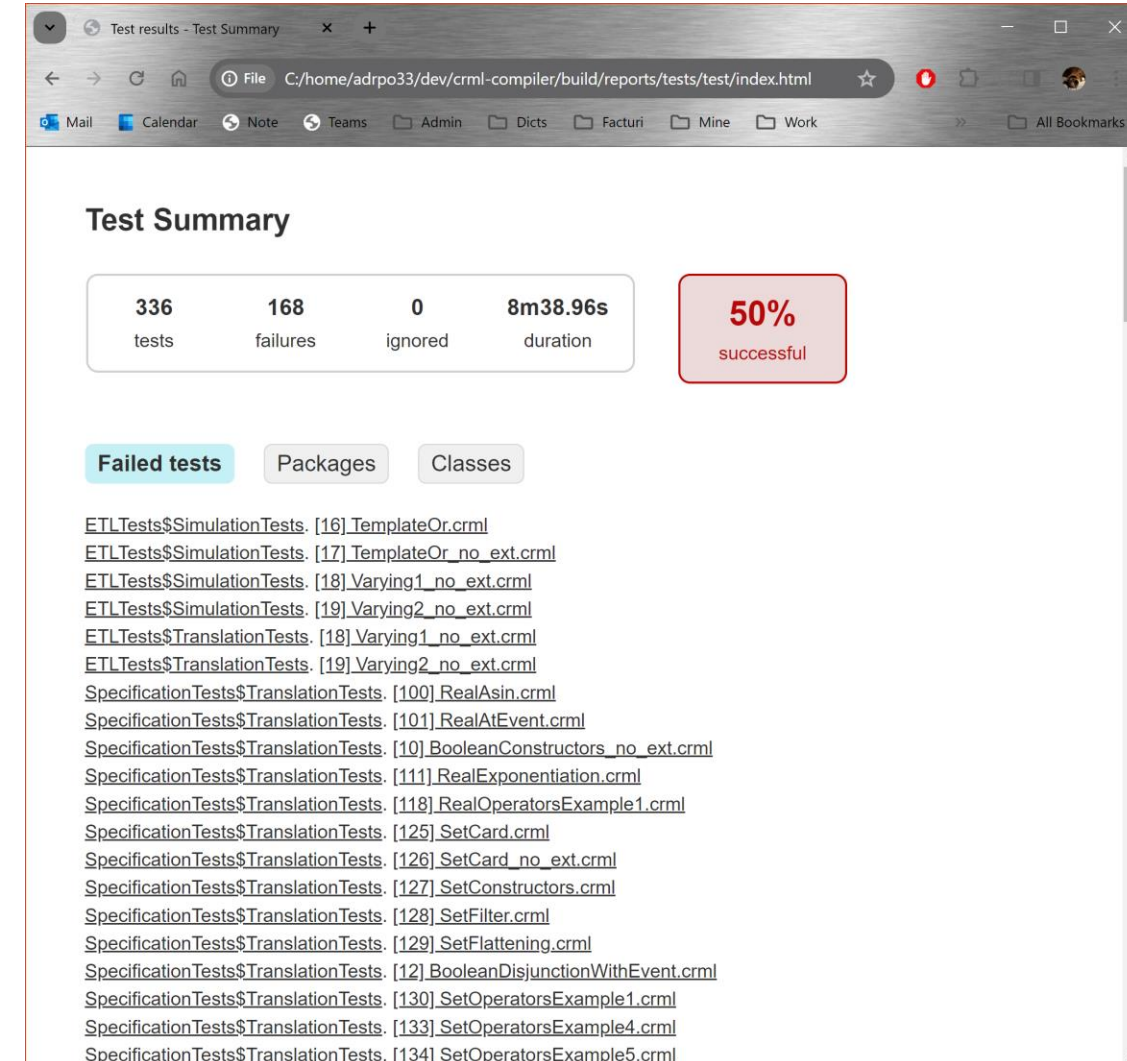
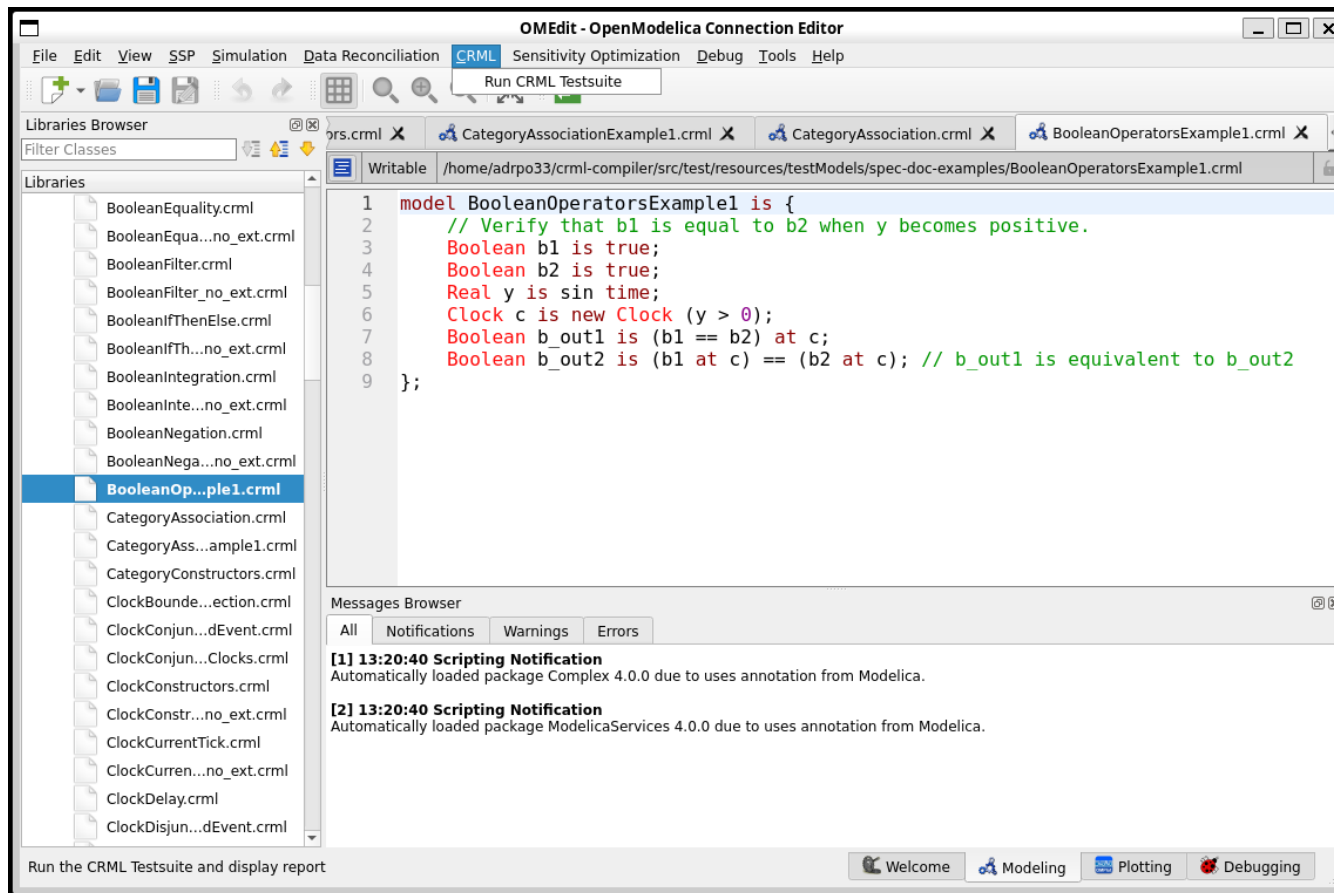
The initialization finished successfully without homotopy method.

STATISTICS

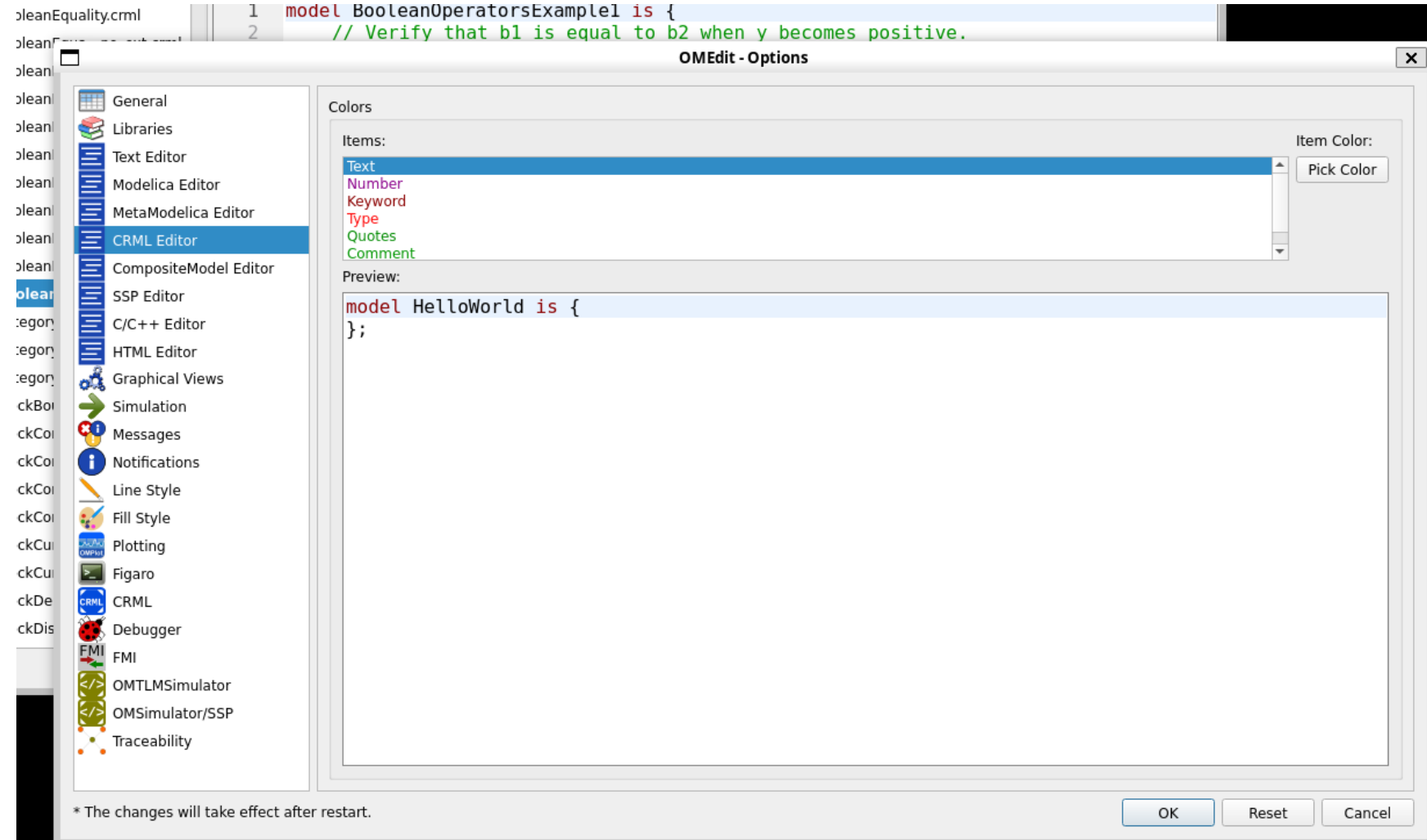
The simulation finished successfully.

Ln: 3, Col: 11 Welcome Modeling Plotting Debugging

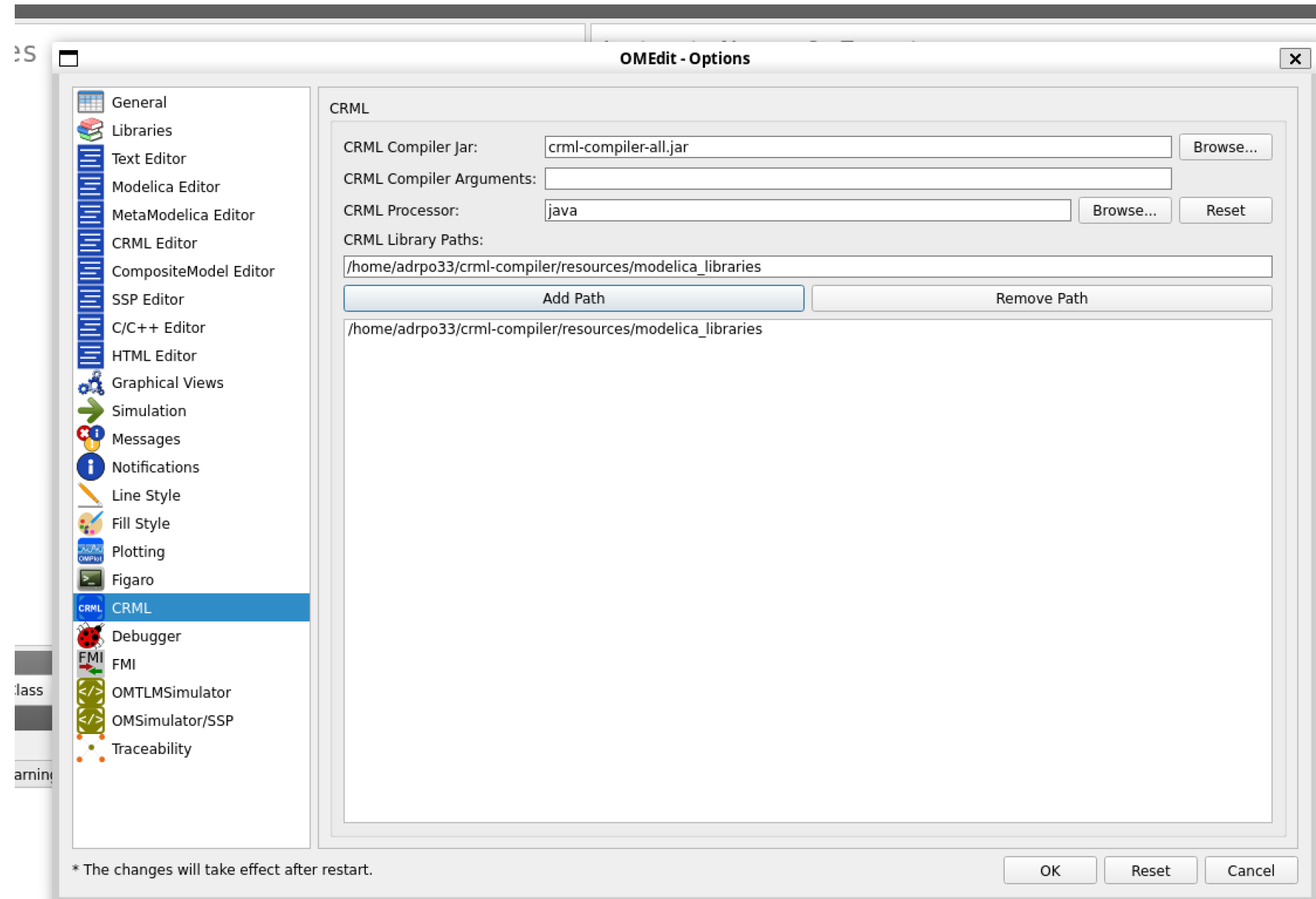
■ Run CRML Testsuite



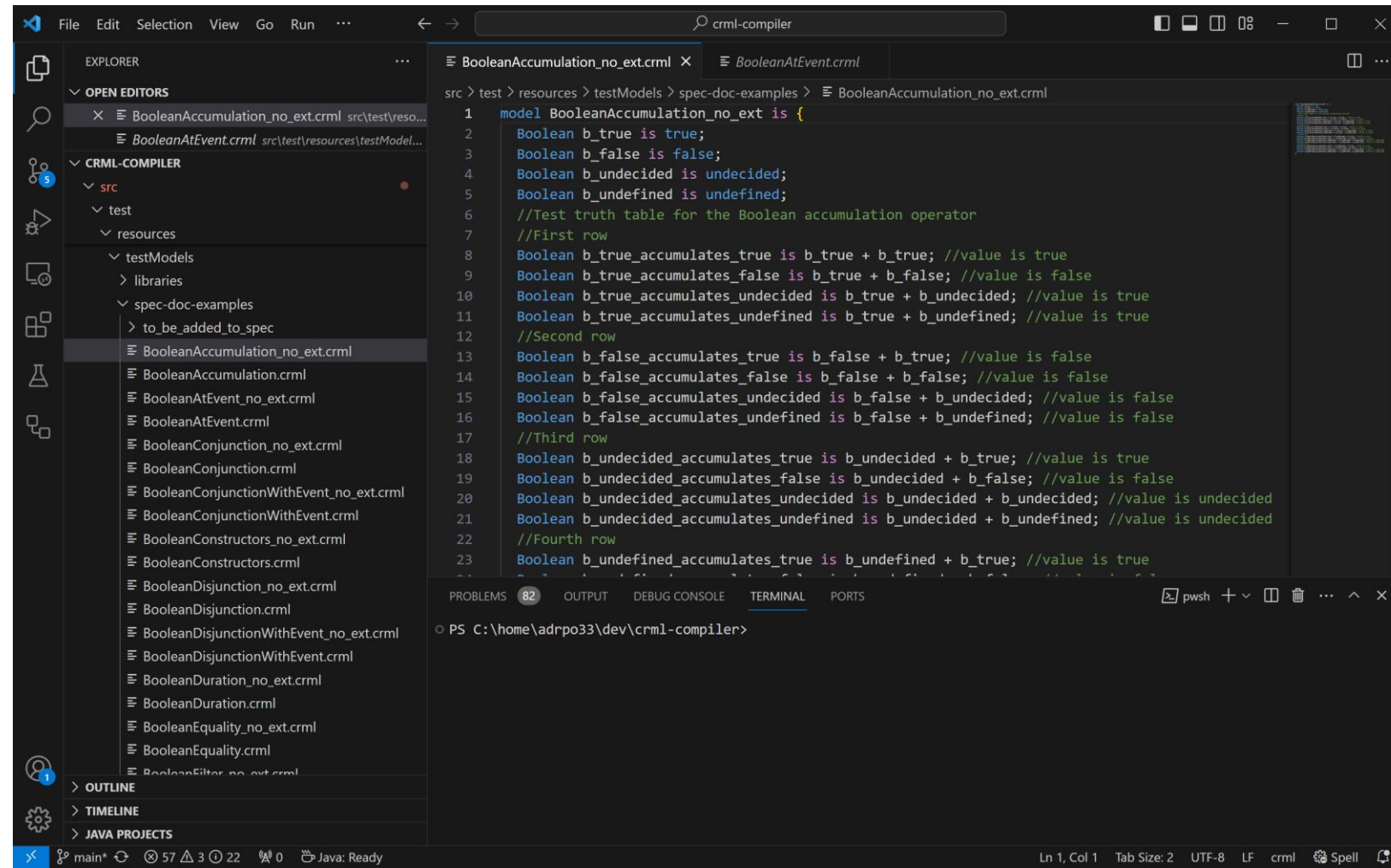
■ Editor Settings



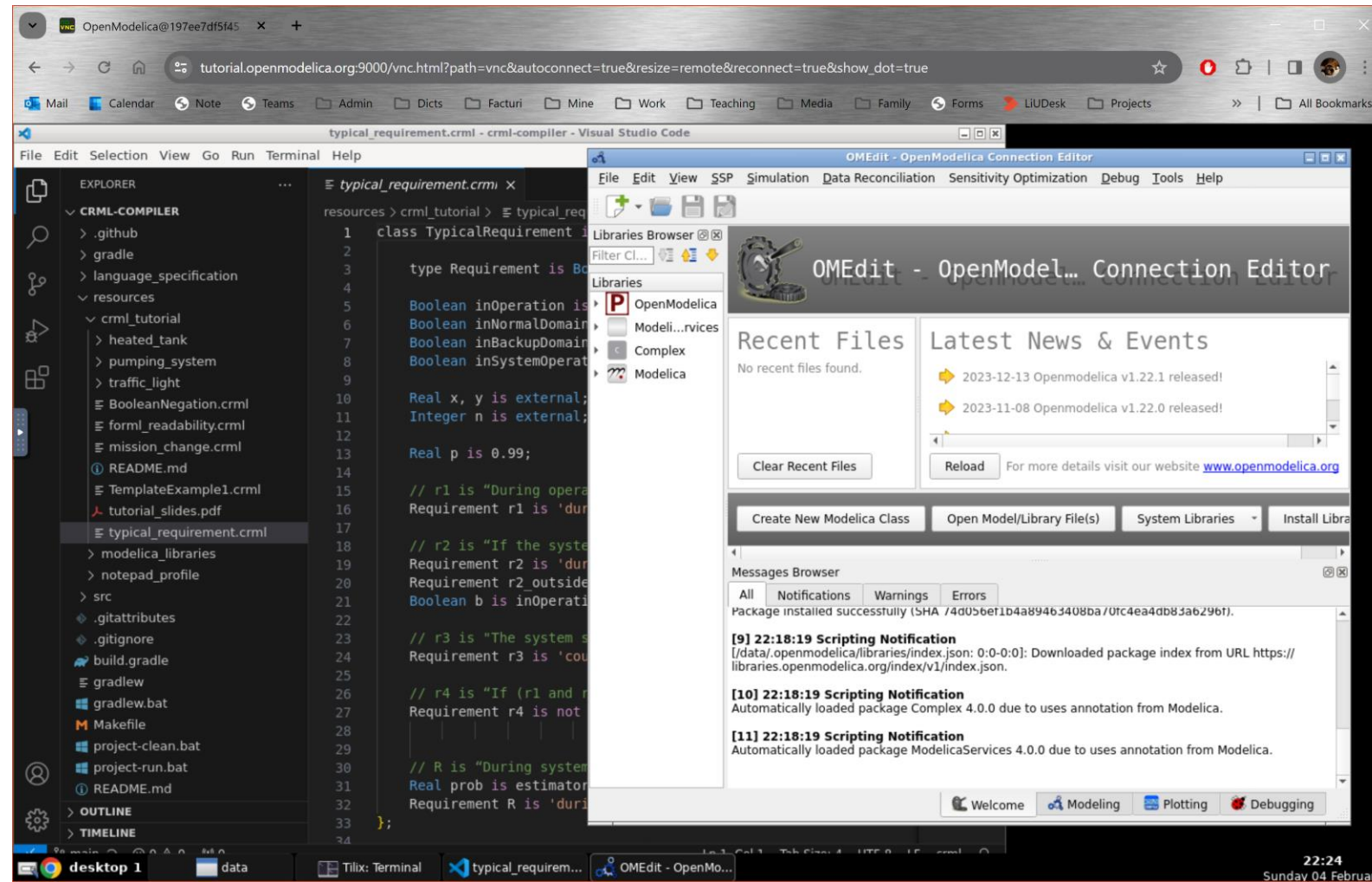
■ Tool Settings



- Basic VSCode extension for CRML
 - <https://github.com/lenaRB/crml-vscode>
 - syntax highlighting

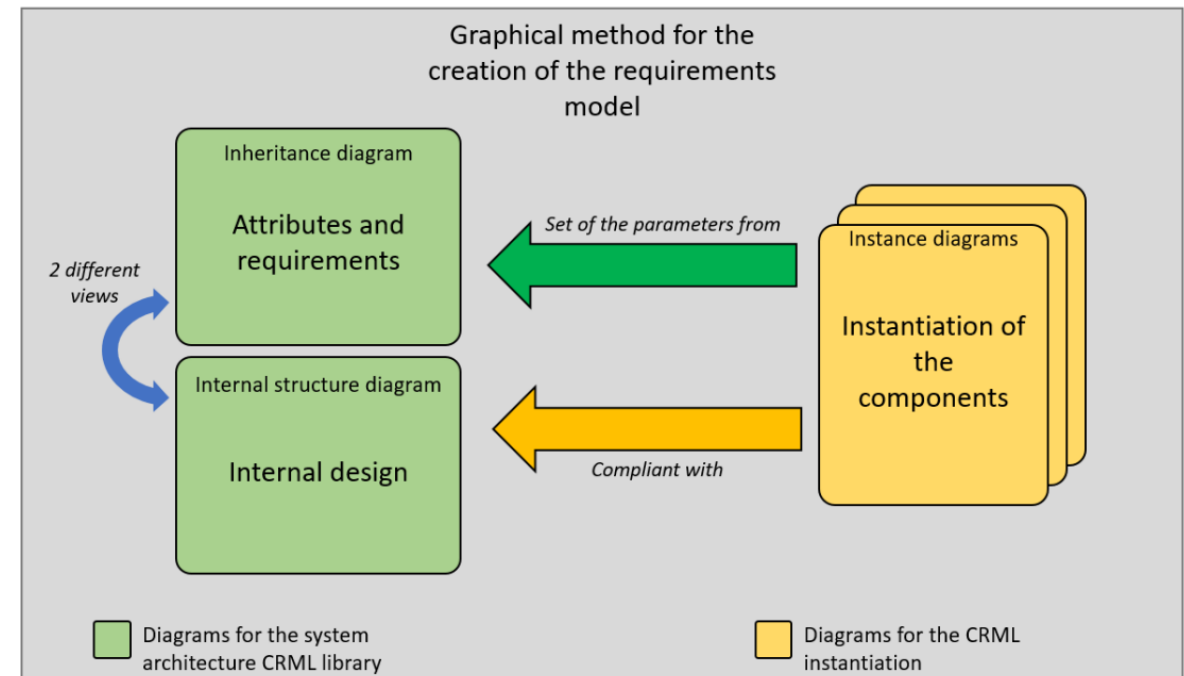


- CRML and OpenModelica tutorial available online
 - <https://tutorial.openmodelica.org/>
 - No install needed
 - Contact us for user access



- CRML integration is now part of OpenModelica v1.25
 - Available as nightly-build
 - Will be released as final version soon
- We continue the CRML compiler development
 - To support as much as possible from the CRML specification
 - Work ongoing on support of ETL and FORML libraries, Probabilistic aspects, hierarchical modeling

- Visual Graphical support for CRML
 - Starting point is the KTH Master Thesis from Baptiste Mazurié
 - Support for UML-style (inheritance) class diagrams for CRML classes, attributes and inheritance
 - Support for UML-style internal structure diagrams for CRML class composition and behavior
 - Support for instance diagrams



- What is CRML
 - The Common Requirement Modelling Language
- CRML Tooling
 - The CRML Compiler
- CRML Integration
 - OMEdit & VSCode & Online
 - Status
- **Future work**

■ Near Future

- Design and implement the CRML graphical support
- Present CRML to Modelica Association as a new standard

■ Future

- How to group together several requirements into a project
- How to handle debugging (CRML <- Modelica <- C code)
- Evaluate traceability from CRML to simulation results
- Integration with dashboards to support dynamic requirement monitoring

Thank You!

Questions?

The CRML Project
<https://crml-standard.org/>

The OpenModelica Project
<https://www.OpenModelica.org>