|  |  |
| --- | --- |
| D2.2 | CRML tooling architecture |
| Access[[1]](#footnote-1): | **PU** |
| Type[[2]](#footnote-2): | **Report** |
| Version: | **1.3** |
| Due Dates[[3]](#footnote-3): | **M12, M24, M36** |
| Environment for model-based rigorous adaptive co-design and operation of CPS | |
| **Executive summary[[4]](#footnote-4):** | |
| This deliverable presents an overall architecture for a Common Requirements Modeling Language (CRML) toolchain that will integrate different partner contributions into the overall workflow and elaborates on how different partners plan to integrate their contributions. | |

**Deliverable Contributors:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Name | Organisation | Primary role in project | Main Author(s)[[5]](#footnote-5) |
| Deliverable Leader[[6]](#footnote-6) | Daniel Bouskela | EDF | WP2 Leader |  |
| Contributing Author(s)[[7]](#footnote-7) | Vince Molnár | BME | T2.2 member | X |
| Marin Priala | KCS-REUSE | T2.2 leader |  |
| Lena Buffoni | LiU | Coordinator |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Internal Reviewer(s)[[8]](#footnote-8) | Jonathan Menu | SISW | T2.2. member |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Document History:**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Reason for Change | Status[[9]](#footnote-9) |
| 0.1 | 03/10/2016 | First Draft Version | Draft |
| 1.0 | 07/10/2016 | First Issue | Released |
| 1.1 | 06/01/2021 | KCS-REUSE contribution in 1.1.1 Contracts in SysML | Draft |
| 1.2 | 07/01/2021 | KCS-REUSE contribution in 1.1.1 Contracts in SysML Modified | Draft |
| 1.3 | 11/01/2021 | LIU contribution | Draft |
| 1.3.1 | 14/01/2021 | Review by SISW | In Review |

**Contents**

[Abbreviations 4](#_Toc61513037)

[1 Tool architecture overview 5](#_Toc61513038)

[1.1 Graphical Contract Modeling in SysML 5](#_Toc61513039)

[1.1.1 Contracts in SysML 5](#_Toc61513040)

[1.1.2 Contract modeling in Gamma 6](#_Toc61513041)

[1.2 Binding to Behavioral Models 7](#_Toc61513042)

[1.2.1 Generating FMUs from SysML models with Gamma 7](#_Toc61513043)

[1.2.2 Bindings to Modelica models 8](#_Toc61513044)

[1.3 CRML to Modelica translation 8](#_Toc61513045)

[References 10](#_Toc61513046)

# Abbreviations

List of abbreviations/acronyms used in document:

**Abbreviation Definition**

CRML Common Requirements Modeling Language

FMI Functional Mock-up Interface

FMU Functional Mock-up Unit

M&S Modelling and Simulation

N/A Not Applicable

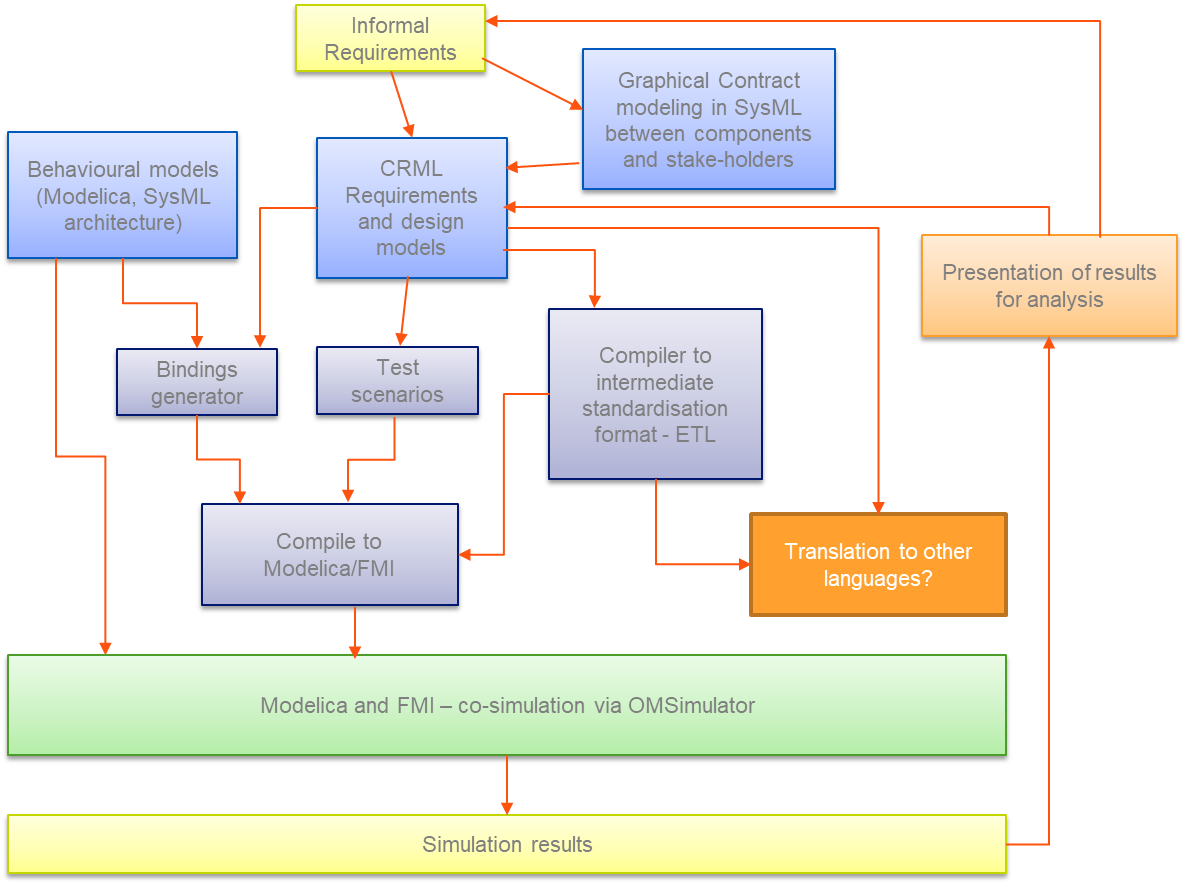
NLP Natural Language Processing

SotA State of the Art

TBD To Be Defined

# Tool architecture overview

The figure below presents the general view of the CRML architecture, different contributions are detailed in the following sections.



## Graphical Contract Modeling in SysML

CRML is not meant to be separate from existing modelling languages. In fact, we plan several connection points based on bindings (see section 1.2). In addition to the theoretical compatibility (described in D2.1) and technical realization of bindings, we also propose a contract-based methodology (also see in D2.1) that we will realize as an extension of the SysML metamodel with a profile. Other experimental work will be performed in the Gamma Statechart Composition Framework [2, 3], where we develop a scenario-based contract language. This language is tightly integrated with the formal verification capabilities of the Gamma framework, so it will be a suitable target for transformations from both SysML, and a relevant subset of CRML. In addition, the theoretical experience that we gather can be used to shape/extend CRML.

### Contracts in SysML

Technically, a *Contract* in SysML will be a stereotyped *Requirement* element, referring to one or more *Assumptions* and *Guarantees* (also implemented as stereotyped *Requirements*). In its simplest use case, assumptions and guarantees can be specified in natural language. Contracts do not use the textual description as a contract by definition requires the compatibility of its assumptions and guarantees.

In addition to the textual description, assumptions and guarantees can be formalized with a constraint too. A constraint may be defined in SysML or any language supported by a given editor (e.g., JavaScript). We will extend this selection with CRML. Technically, this means that the constraint text can contain a CRML specification, tagged with CRML as the constraint language.

With this simple solution, we can enable external tools to find and interpret contracts in CRML in the context of a SysML model. With bindings defined for such a context, evaluation can follow the same path as in a Modelica-based workflow: compile a simulation model or a set of FMUs (functional mockup units) and execute the test scenarios, evaluating CRML requirements based on the expressed behavior.

The result of such an analysis can be recorded in the contract/guarantee element (assumptions are not verified separately, as by definition they are guaranteed to hold if the guarantee and contract are fulfilled). In addition to the result of the automatic analysis, a manual verdict can also be stored in the model. On the one hand, this is the preferred way to go when either the guarantee or the assumption is given in natural language, but it also enables engineers to override the automatic verdict if it is deemed incorrect (using the same field for this is not a good idea, because the automatically computed verdict may be overwritten any time by e.g. a scheduled automatic analysis).

As it was previously mentioned, the *Contracts*, *Assumptions* and *Guarantees* can be specified in natural language. When this happens, the Natural Language Processing (NLP) technology can be applied to enhance and assure the quality, coherence, correctness, consistency and completeness of the written text. In the framework, authoring of the different statements forming the contract itself, the assumptions and the guarantees becomes possible using the dedicated software like the RAT Authoring tool[[10]](#footnote-10).

Using tools like RAT Authoring it will be possible to define and verify the structure of the *Contracts*, *Assumptions* and *Guaranties* in natural language, allowing automatic transformation and connection with simulation environments (as mentioned above).

The correct structure of the texts allows the evaluation of its quality as well, by using textual metrics, all of them calculated using natural language technology.

### Contract modeling in Gamma

The Gamma Statechart Composition Framework is a tool to model, formally verify and generate code from component-based reactive systems. Gamma has been used as a verification back-end for SysML models [1]. Recently, we have started the extension of the tool with scenario-based contracts, which we continue to investigate in the framework of the EMBRACE project.

In Gamma, components may interact only through ports with well-defined interfaces, defining the types of messages that can be sent/received through them. Ports of components can be connected in a composite component with channels, as well as bindings can be defined to assign a port of a component as the realization of a (proxy) port of a composite component. Therefore, it seems natural that scenarios should be defined on ports.

A scenario can be visualized as a sequence diagram, with extensions like message modality. There will be a lifeline for the component owning the port, and another one for the component on the other end of the channel. A message may be hot, meaning that it is mandatory to send/receive it (otherwise the contract is violated), or cold (which means the contract did not specify the current case, i.e., the result is inconclusive). Scenarios consist of two parts: the pre-chart (with only cold messages), and the main chart (with cold and/or hot messages). The former is responsible for triggering the evaluation of the contract, after which we expect the main chart to be realized by the communication through the port.

With the evolution of CRML, we will investigate if and how the two approaches overlap, and if we can transfer elements and experience from one to the other. Contracts in Gamma can be used as a specification in model checking, as well as the basis for black-box test generation, covering the behaviors described in the contract. Therefore, mapping (a subset) of CRML to Gamma contracts can facilitate the formal verification of discrete-state systems against contracts specified in CRML.

## Binding to Behavioral Models

When specifying behavioral models in CRML, we have to refer to different variables, events and states. These concepts are present in CRML, although typically we will not strive for a complete transformation of the behavioral model to CRML. Instead, we have to declare in CRML the necessary elements and *bind* them to the corresponding elements of the behavioral model – then we can co-simulate the two models with the facilities developed in the framework of the project.

The theoretical foundations for binding to SysML models can be found in D2.1.

### Generating FMUs from SysML models with Gamma

The basis of co-simulation will be FMUs. A nontrivial task is to create an FMU from SysML behavioral models. SysML has known semantical gaps, and most tools support even simple simulation in a very limited and semantically inaccurate way. We plan to solve this issue for discrete-state systems (modeled as a network of components implementing state machines and/or activities) by translating the models to the well-defined Gamma statechart and composition languages, where semantic integrity can be preserved by various methods, including precisely defined transformations, formally specified semantics, and validation of the transformation result with automatically generated tests. Our ongoing work on translating SysML to Gamma for formal verification has been published in [1]. We have started to apply the same methodology in the EMBRACE project as well, but this time for the generation of FMUs.

Once we have a Gamma model, the code generator can generate a Java program that precisely simulates the behavior of the original SysML model. In the framework of the EMBRACE project, we will extend the generator to also generate a wrapper implementing the FMI (functional mockup interface), i.e., to generate an FMU for the original SysML models. Furthermore, the code generator is flexible enough to make new features easily realizable – for example, a fine-grained control of nondeterministic behavior or a more orchestrated cooperation between FMUs may become necessary as the project evolves.

### Bindings to Modelica models

The work on bindings between CRML and Modelica has currently started and will be based on the algorithm developed in the ITEA2 project MODRIO[[11]](#footnote-11).

## CRML to Modelica translation

**Diagram

Description automatically generated**

The figure shows the CRML integration in the model design cycle.

CRML allows the user to define their own language constructs on top of the built-in basic syntax. The CRML specification is described more in detail in deliverable D2.1.

The first step of the translation process is to add the operator and template definitions in the user model to the pre-defined list of constructs (1).

In order to translate the model to Modelica, it might be necessary to have the definition of the associated physical model, for instance to identify all the instances of a certain type that might belong in a set. The connection between the requirements model and the physical model is done via the bindings mechanism developed in the MODRIO project (2).

The information from the physical model will be used to populate the sets of instances defined in the requirement model (3).

The CRML model is then translated into standard Modelica code which can be compiled with the OpenModelica compiler (4). Since during the simulation a new event can trigger the creation of new periods with associated equations, the work from WP3 on multi-mode models will be used to provide support for the creation of new objects (dynamic data-structures). This support is currently being implemented in the prototype Julia-based Modelica compiler. Later on, work on dynamic recompilation will be integrated for improved handling of sets (5).

The requirement model can then be co-simulated with other models in Modelica directly (6) or exported as an FMI unit and used for co-simulation via the OMSimulator (7) developed in the ITEA3 OPENCPS project[[12]](#footnote-12).

# References

1. Benedek Horváth, Bence Graics, Ákos Hajdu, Zoltán Micskei, Vince Molnár, István Ráth, Luigi Andolfato, Ivan Gomes, and Robert Karban. 2020. Model checking as a service: Towards pragmatic hidden formal methods. In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS '20). Association for Computing Machinery, New York, NY, USA, Article 37, 1–5.
2. Vince Molnár, Bence Graics, András Vörös, István Majzik, Dániel Varró. The Gamma statechart composition framework: Design, verification and code generation for component-based reactive systems. ICSE (Companion Volume) 2018: 113-116
3. Bence Graics, Vince Molnár, András Vörös, István Majzik, Dániel Varró. Mixed-semantics composition of statecharts for the component-based design of reactive systems. Softw. Syst. Model. 19(6): 1483-1517 (2020)

1. Access classification as per definitions in PCA; PU = Public, CO = Confidential. Access classification per deliverable stated in FPP. [↑](#footnote-ref-1)
2. Deliverable type according to FPP, note that all non-report deliverables must be accompanied by a deliverable report. [↑](#footnote-ref-2)
3. Due month(s) according to FPP. [↑](#footnote-ref-3)
4. It is mandatory to provide an executive summary for each deliverable. [↑](#footnote-ref-4)
5. Indicate Main Author(s) with an “X” in this column. [↑](#footnote-ref-5)
6. Deliverable leader according to FPP, role definition in PCA. [↑](#footnote-ref-6)
7. Person(s) from contributing partners for the deliverable, expected contributing partners stated in FPP. [↑](#footnote-ref-7)
8. Typically person(s) with appropriate expertise to assess deliverable structure and quality. [↑](#footnote-ref-8)
9. Status = “Draft”, “In Review”, “Released”. [↑](#footnote-ref-9)
10. https://www.reusecompany.com/rat-authoring-tools [↑](#footnote-ref-10)
11. https://itea3.org/project/modrio.html [↑](#footnote-ref-11)
12. https://itea3.org/project/opencps.html [↑](#footnote-ref-12)