CHESS: a Tool to Support the Design of Safety-Critical Systems using Formal Methods

Outline

- Technological context of CHESS
- Functionalities currently implemented in CHESS
- Details: CHESS Profile & Formal Specification Languages

Technological Context

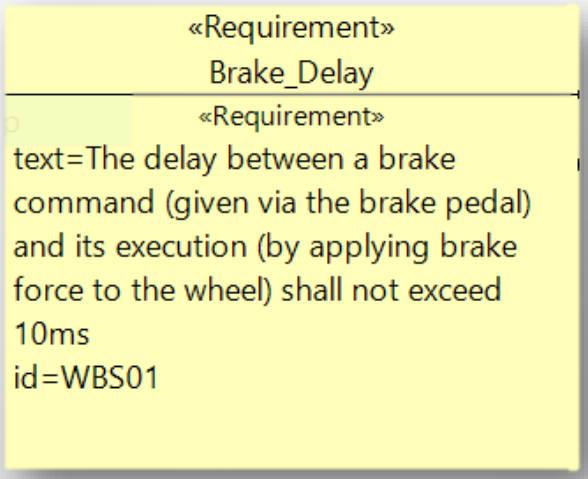
- **Papyrus** (https://eclipse.org/papyrus/) is an open-source project to provide an integrated environment for editing UML and SysML models.
- **CHESS** (www.chess-project.org) is an open-source project to provide component-based engineering methodology and tool support for the development of high-integrity embedded systems.

Functionalities currently implemented

- **System Design**
  - Requirements Specification
  - System definition
  - Requirement Formalization
  - Functional Refinement
    - Architectural Refinement
    - Contract Refinement
    - Parametrized Architecture
  - Nominal and Faulty Behavior Definition
- **Functional Early V&V**
  - Validation of contracts
  - Check the contract refinement
  - Contract-Based V&V of State Machines
  - V&V of state machines
- **Safety Analysis**
  - Contract-Based Safety Analysis
  - Model-based Safety Analysis
- **Trade-off Analysis**
- **Document generation**

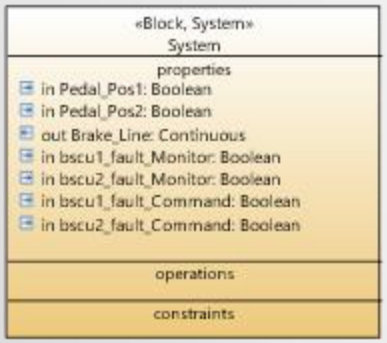
System Design: Requirements Specification

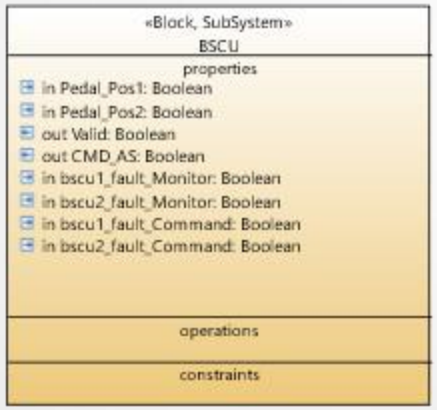
- Informal requirements can be:
  - represented using SysML Requirements Diagram.
  - imported from excel files, csv files, or by using the ReqIF (Requirement Interchange format)

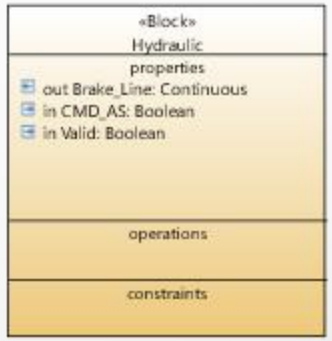


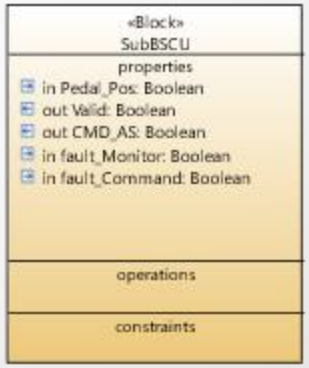
System Design: Architecture definition

- Define components and their interface using the SysML Block Definition Diagrams



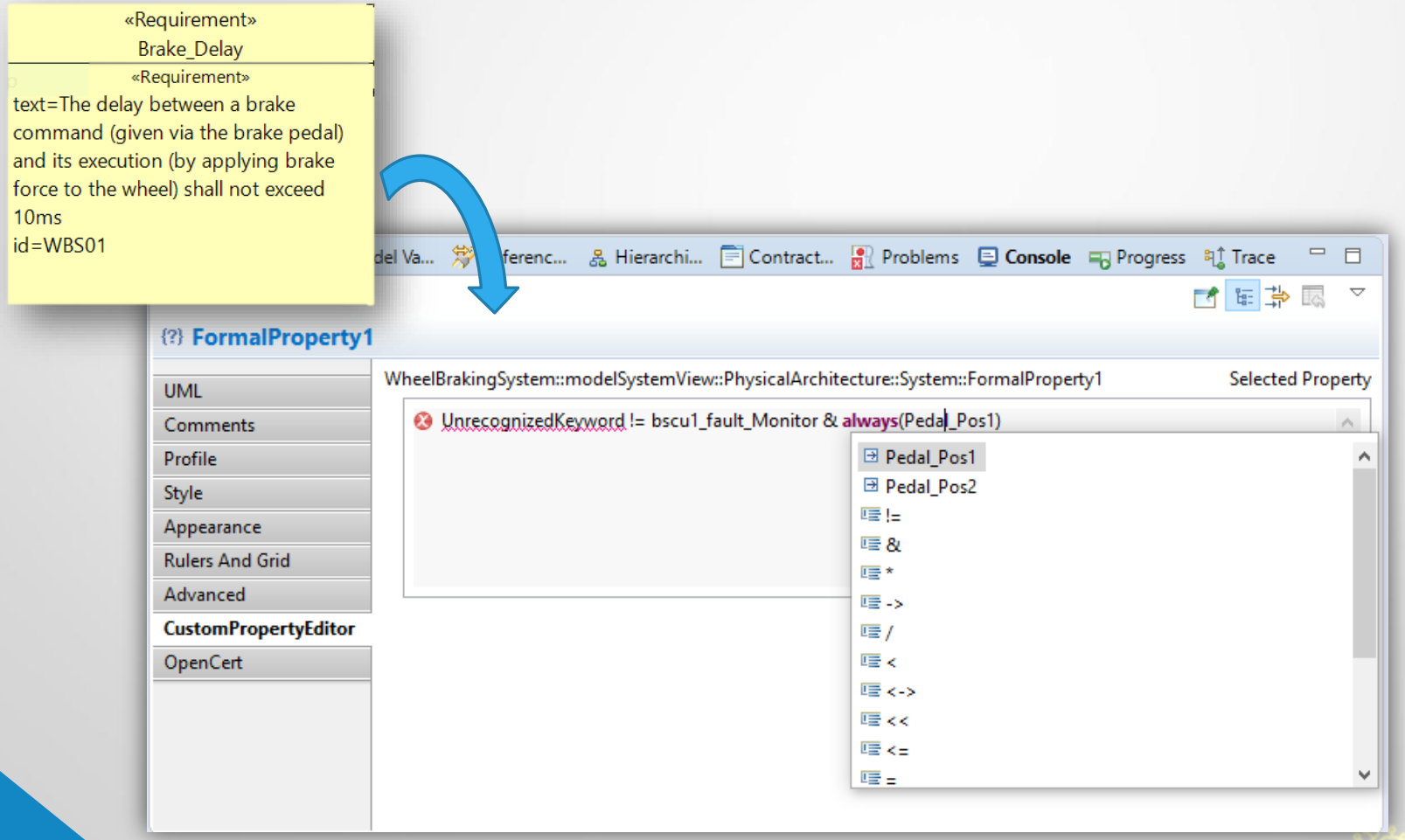






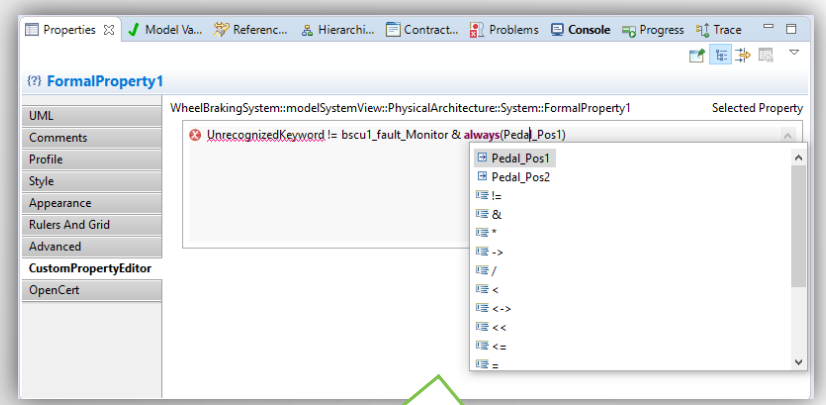
System Design: Requirement Formalization

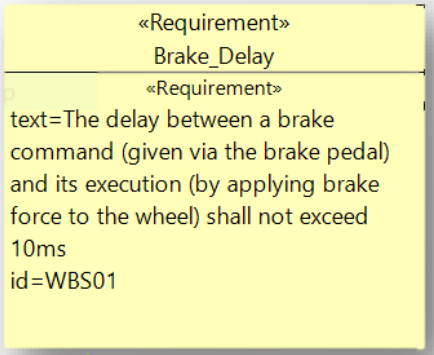
- Translate the informal requirements into formal properties using the textual editor with content assistant for LTL (linear temporal logic)

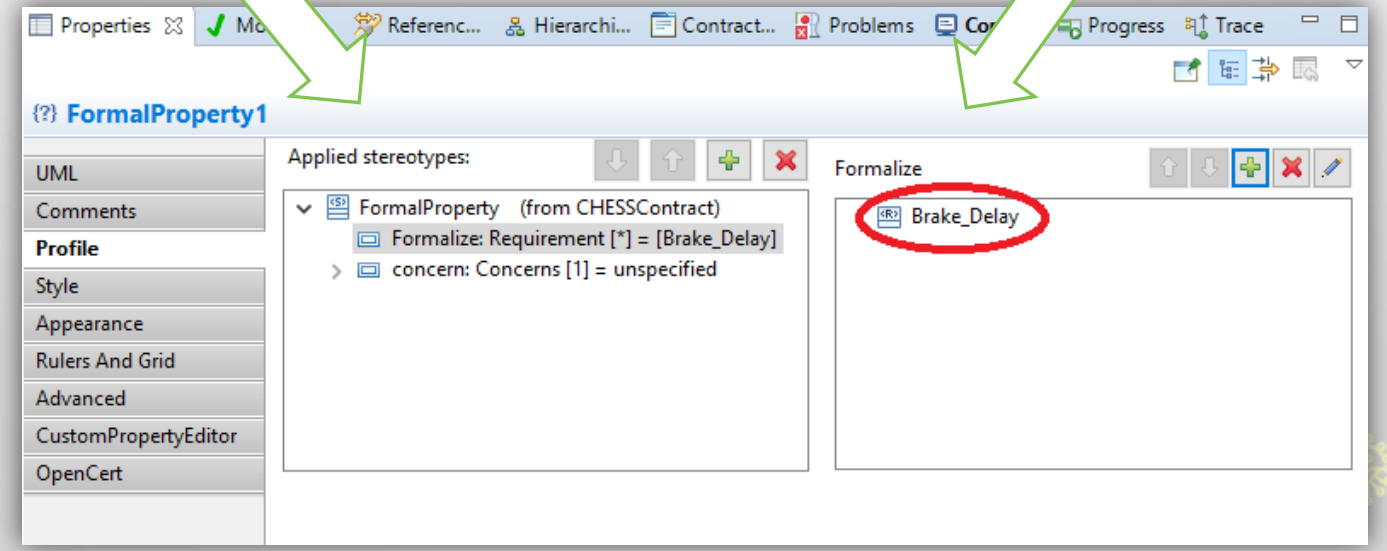


System Design: Requirement Formalization

- Link the informal requirements to the formal properties

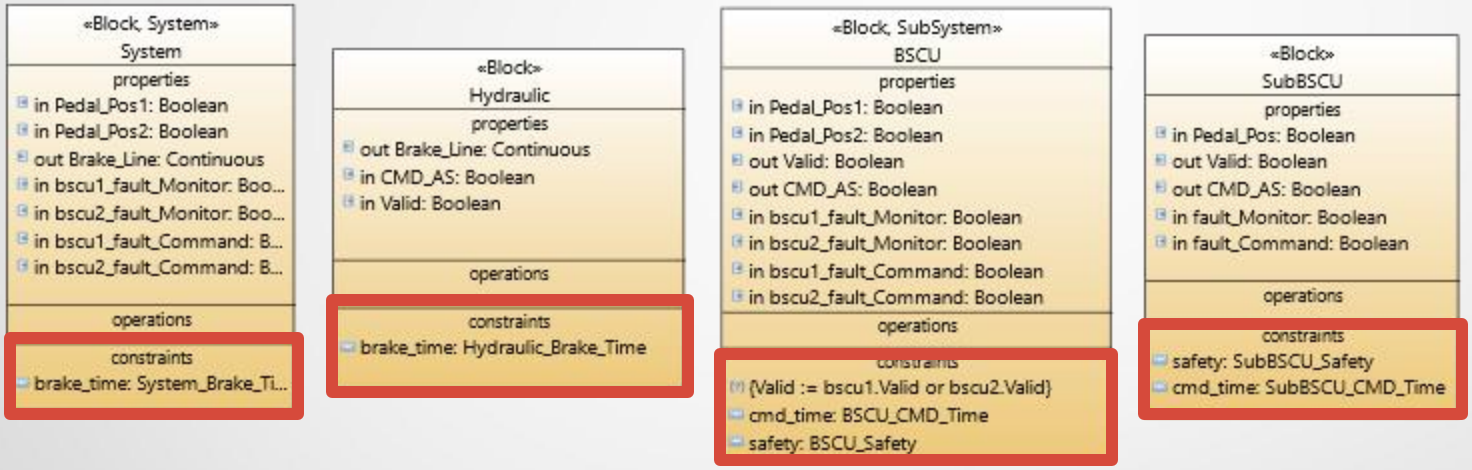






System Design: Contract Definition

- Formal Properties can be further structured into *contracts.*
- Contract describes the expected behavior of the component

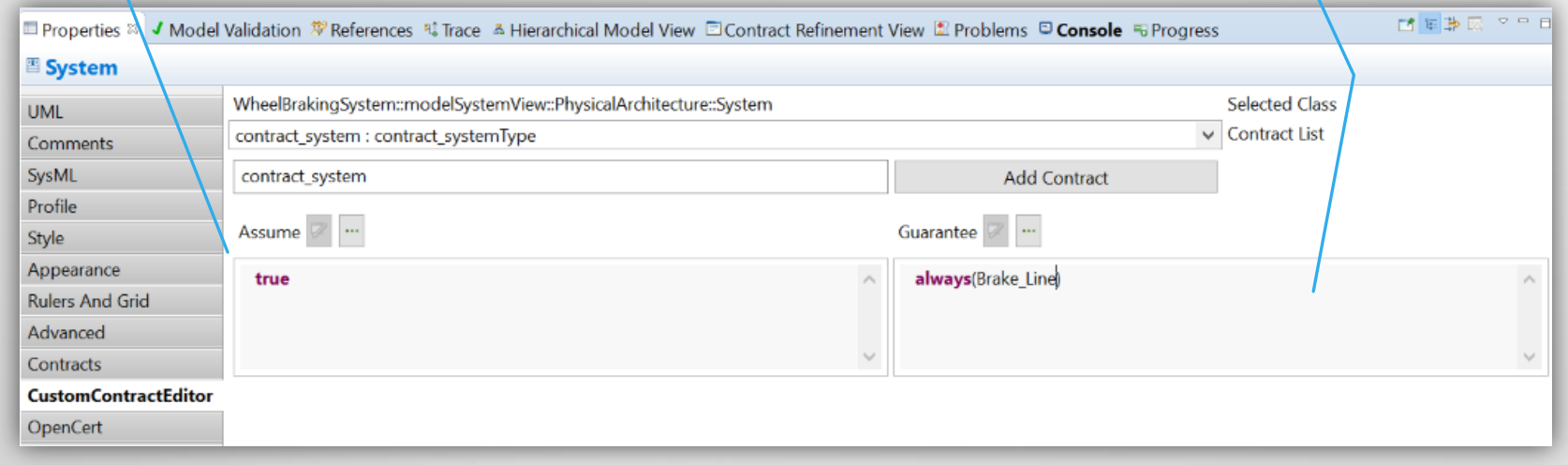


System Design: Contract Definition

- The contract is a pair of properties representing an assumption and a guarantee

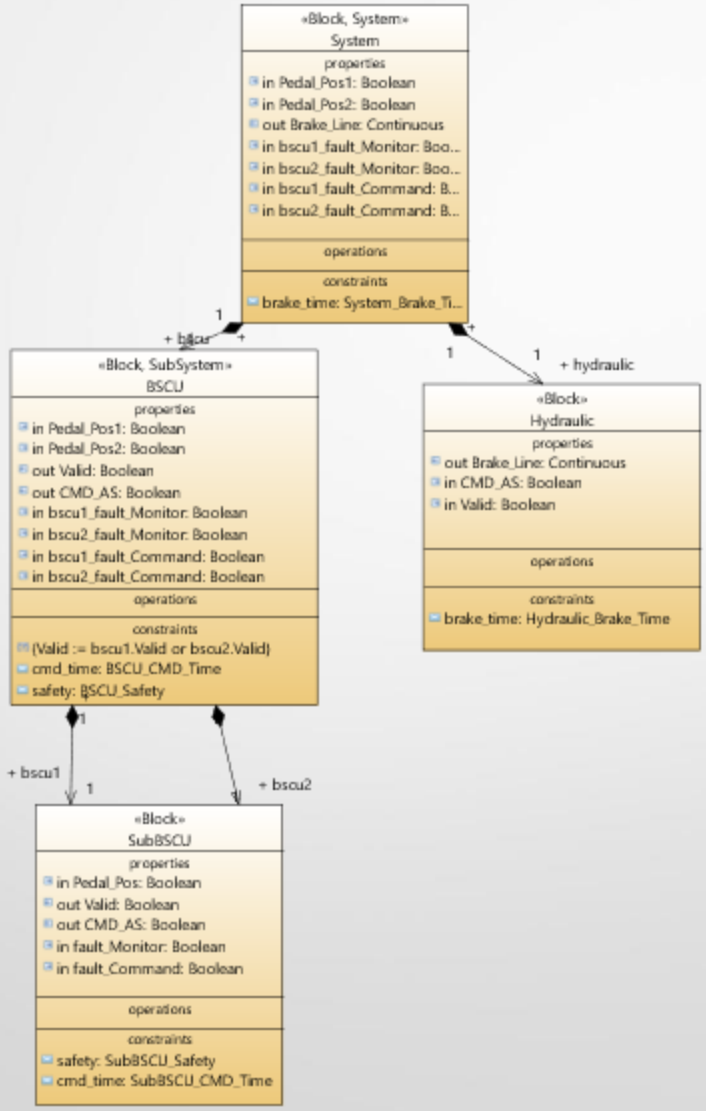
Assumption: Property to be satisfied by the component environment

Guarantee: Property expected to be satisfied by the component provided that the assumption holds



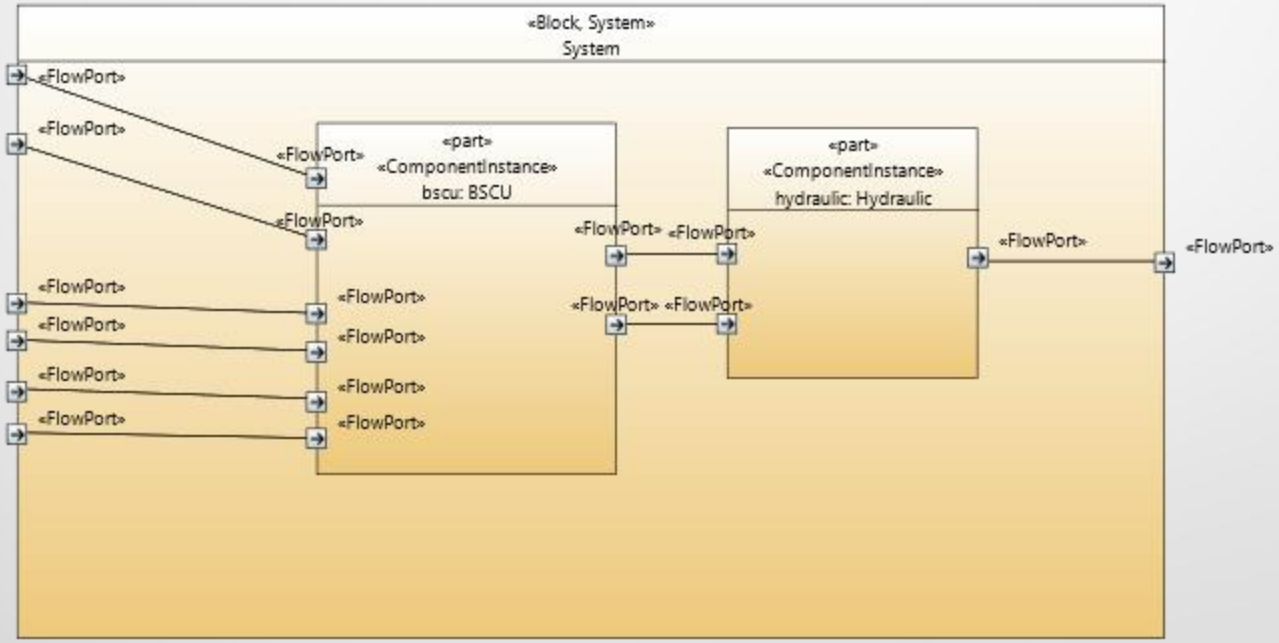
System Design: Architectural Refinement

- Decompose a component via SysML Block Definition Diagrams



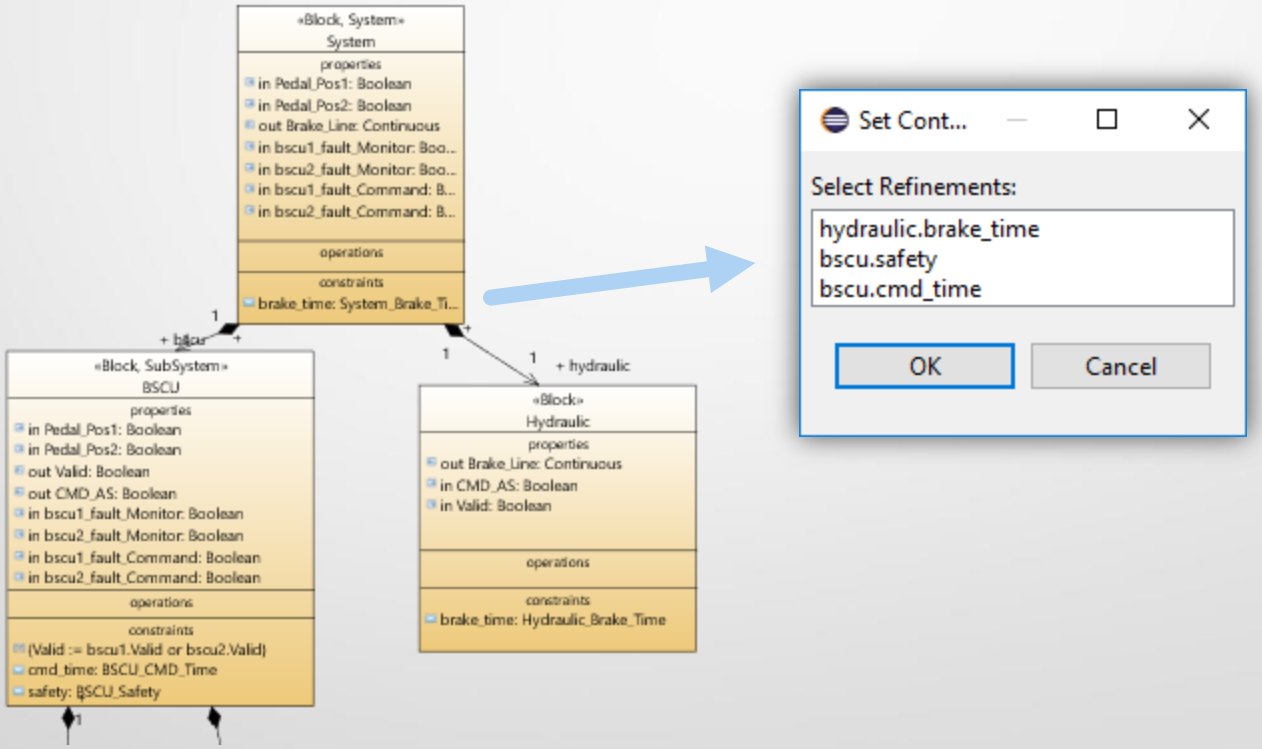
System Design: Architectural Refinement

- Decompose a component via SysML Block Definition Diagrams
- Connect input/output ports of the components using SysML Internal Block Diagrams



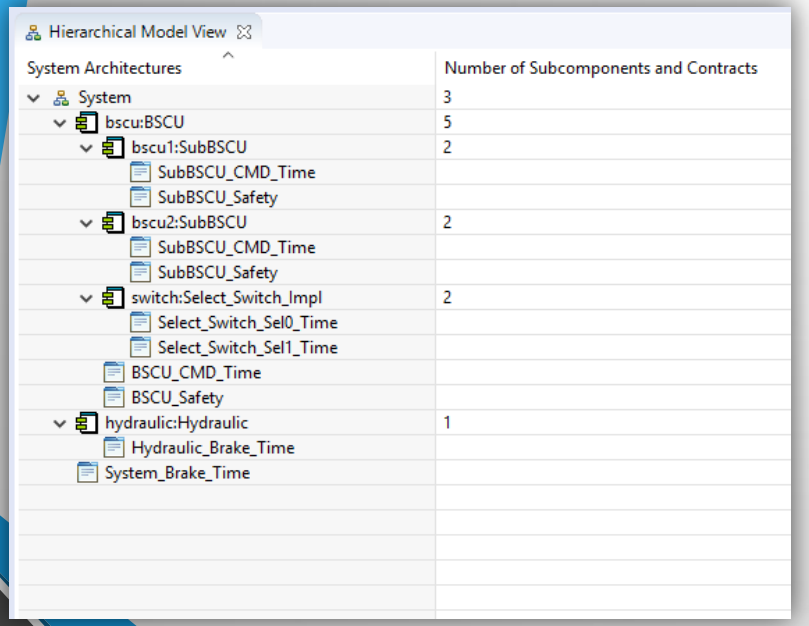
System Design: Contract Refinement

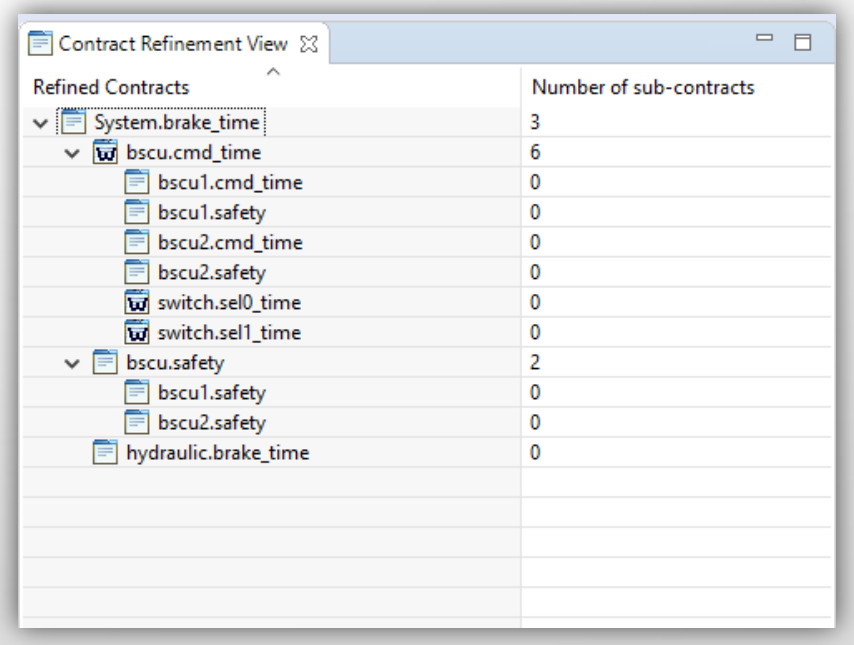
- Refine contracts of composite components linking them to the contracts of subcomponents



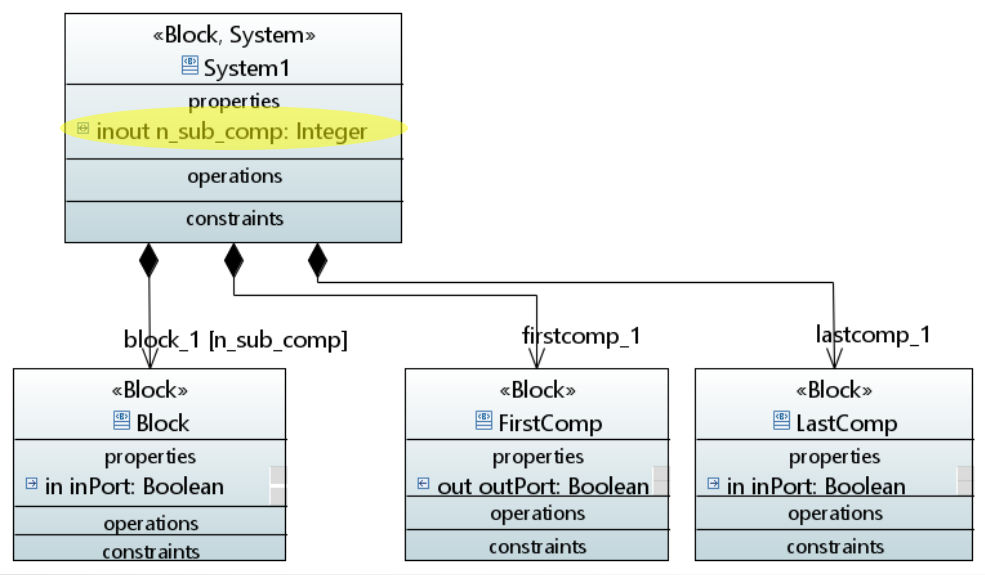
System Design: Contract Refinement

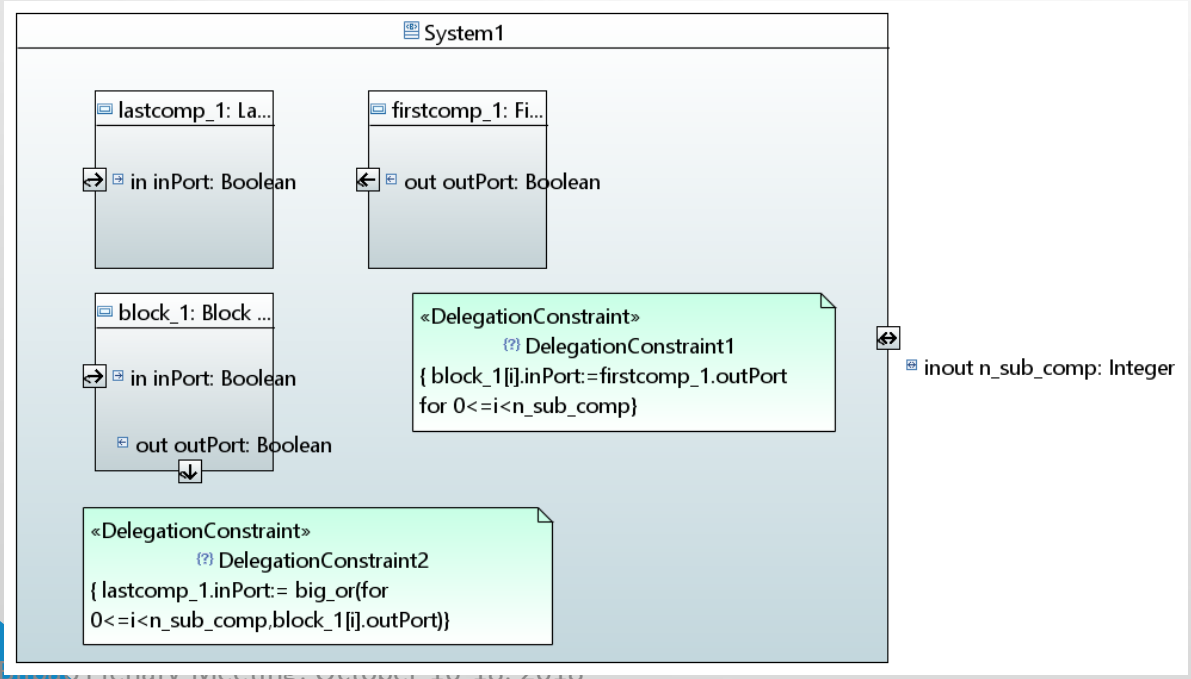
- Refine contracts
- Inspect through overview over:
  - System decomposition
  - Contract refinements



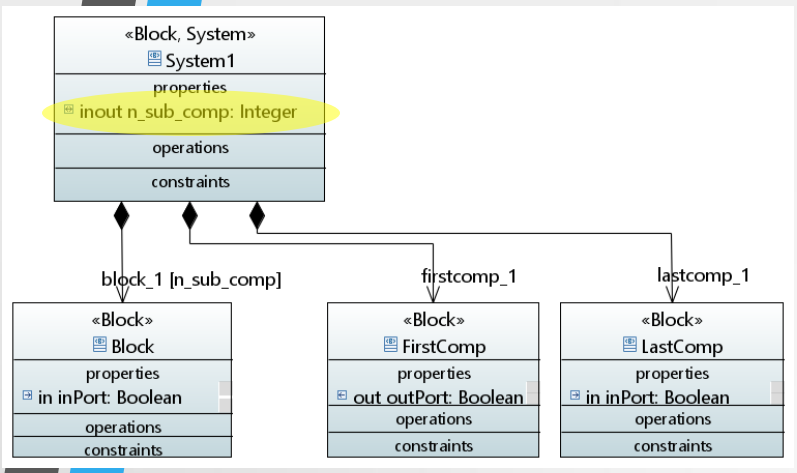


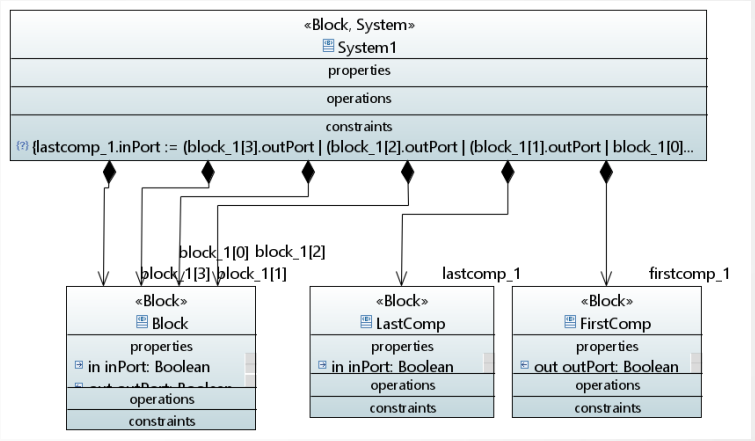
System Design: Parametrized Architecture

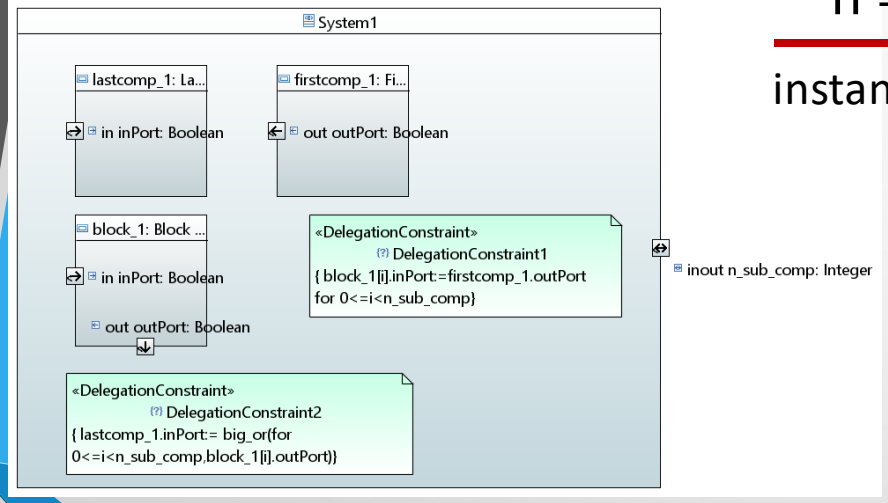




System Design: Parameters instantiation

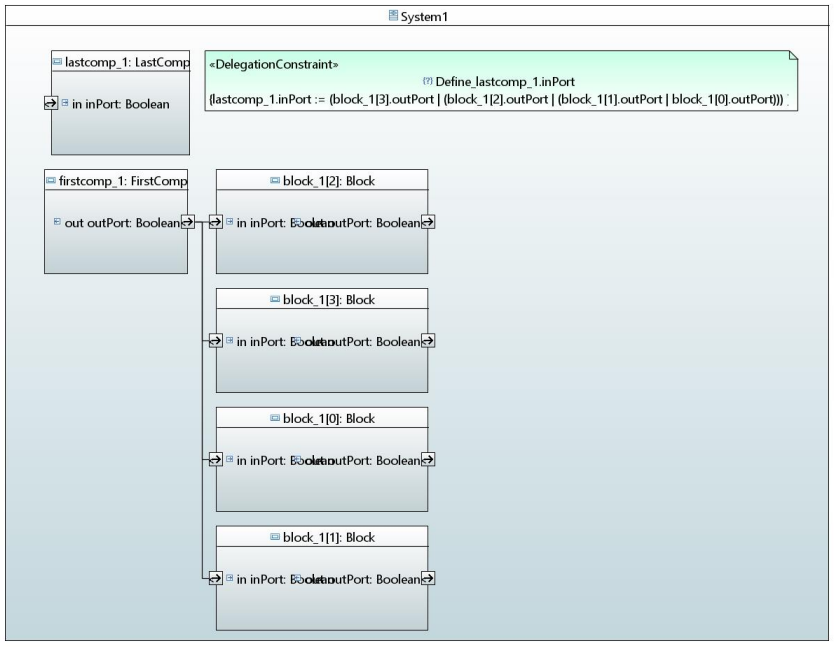






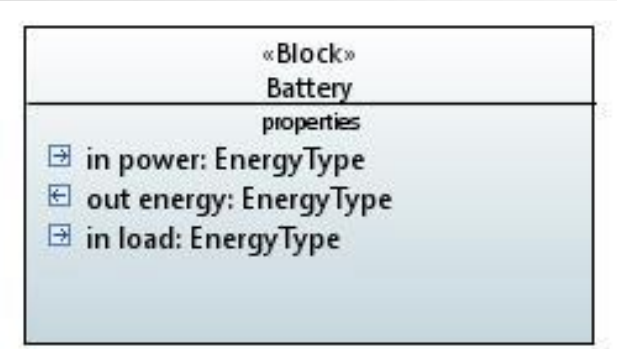
n = 4

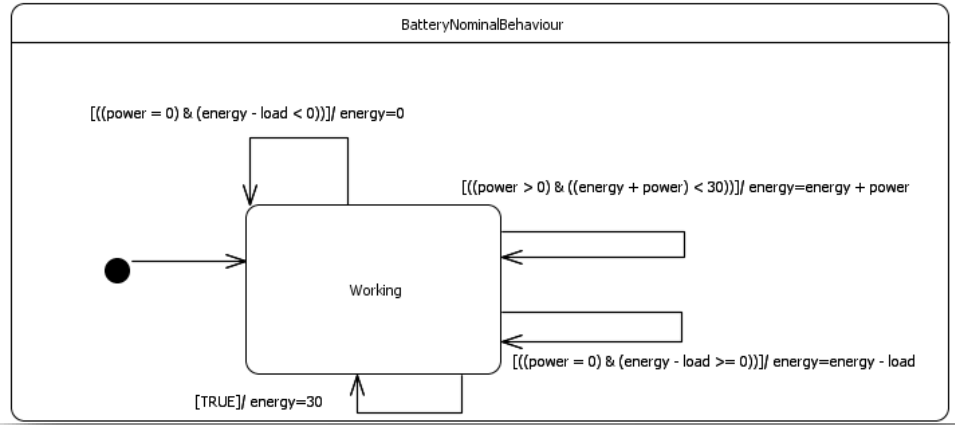
instantiation



System Design: Nominal Behavior Definition

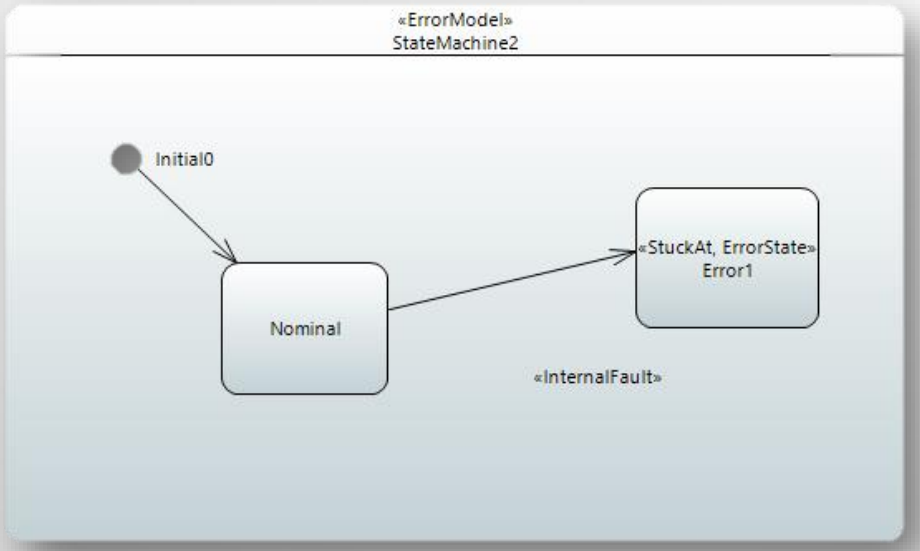
- SysML State Machine Diagrams are used to model the nominal behavior definition of the component.
- A transition comes with a guard and an effect. The guard is a boolean condition upon the values of components properties.





System Design: Faulty Behavior Definition

- Faults are introduced into the system (Fault injection)
- SysML State Machine Diagrams are used to model the faulty behavior definition of the component.

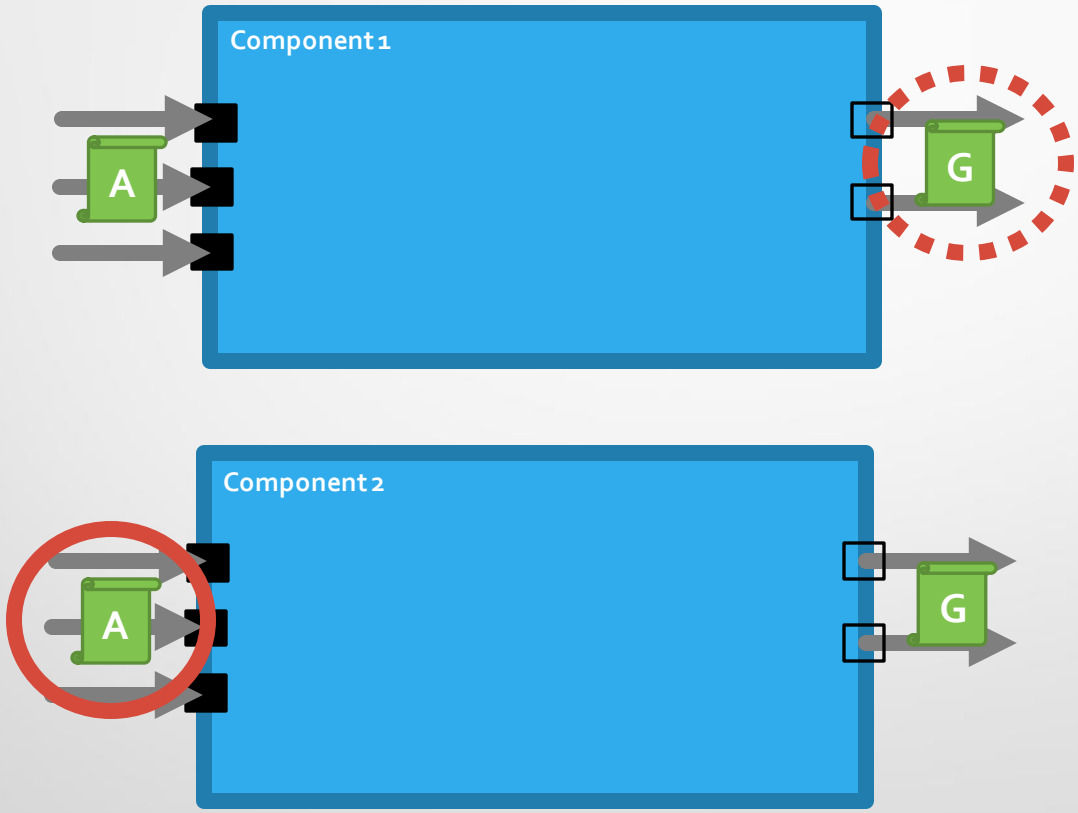


Functionalities currently implemented

- System Design
  - Requirements Specification
  - System definition
  - Requirement Formalization
  - Functional Refinement
    - Architectural Refinement
    - Contract Refinement
    - Parametrized Architecture
  - Nominal and Faulty Definition
- **Functional Early V&V**
  - Validation of contracts
  - Check the contract refinement
  - Contract-Based V&V of State Machines
  - V&V of state machines
- **Safety Analysis**
  - Contract-Based Safety Analysis
  - Model-based Safety Analysis
- **Trade-off Analysis**
- **Document generation**

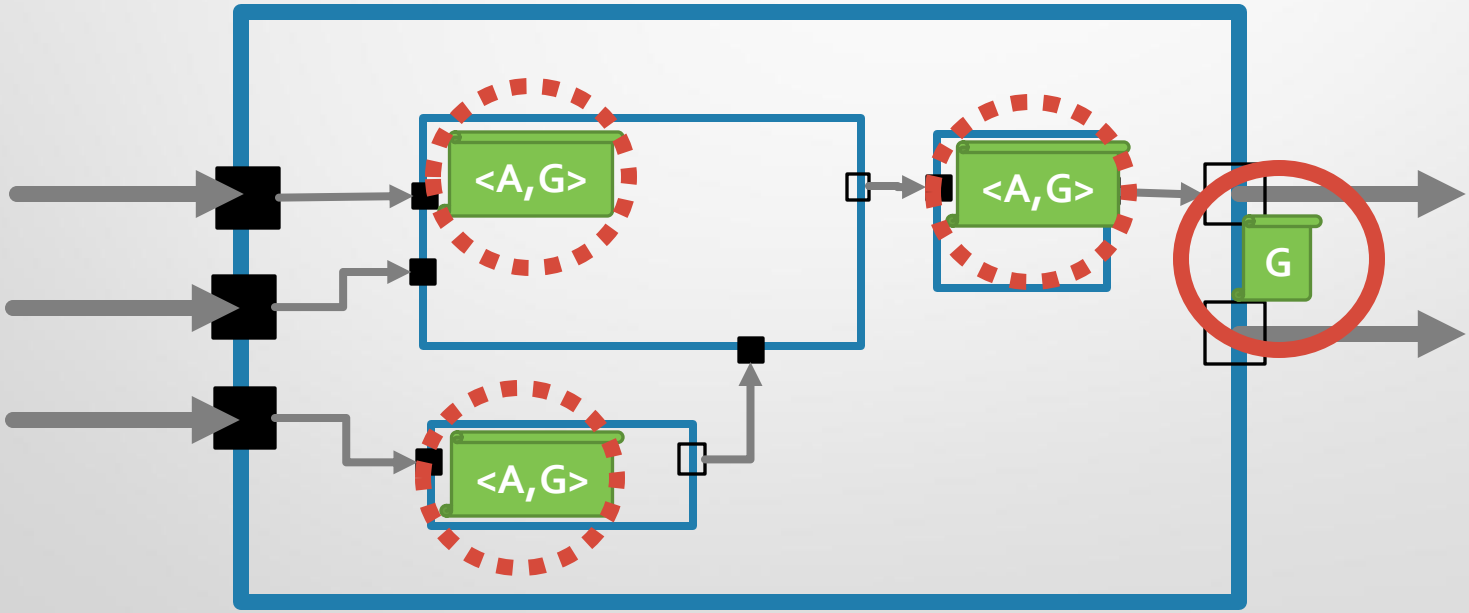
Functional V&V: Validation of contracts

- Check if a specific guarantee of the contract satisfies the assumption of another contract



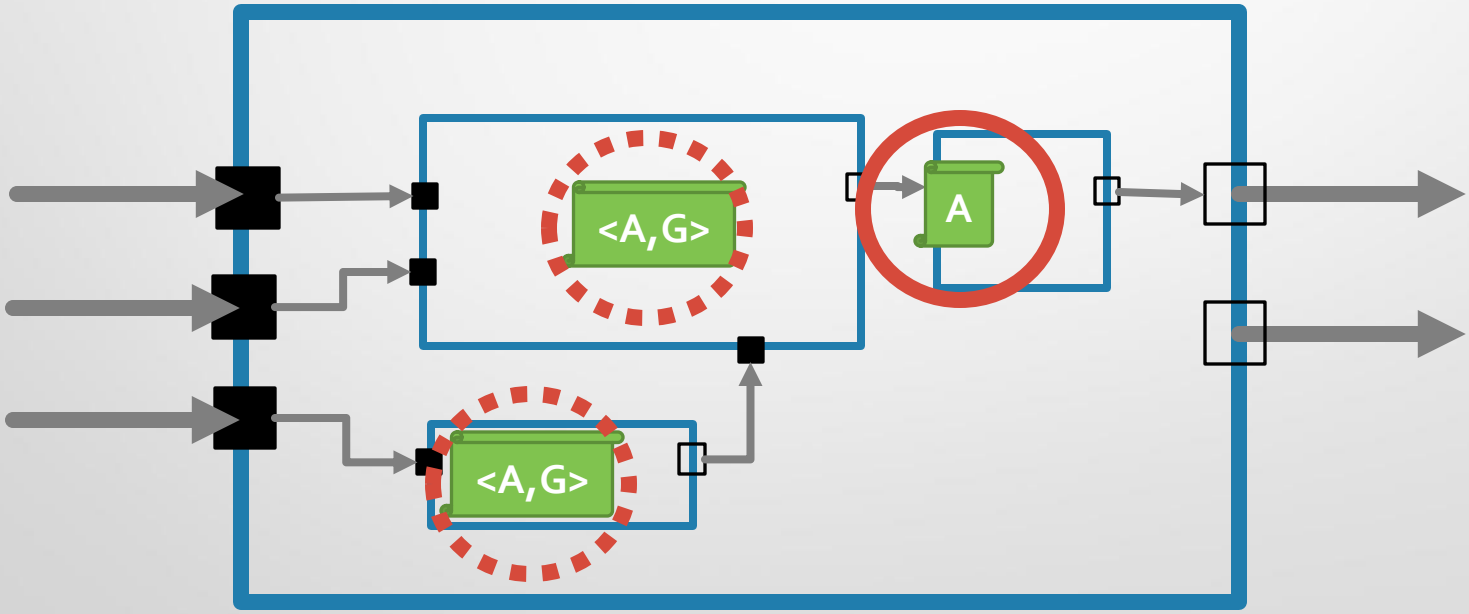
Functional V&V: Check the contract refinement

- Check the contract refinement
  - Guarantees of composite component ensured by contracts of subcomponents



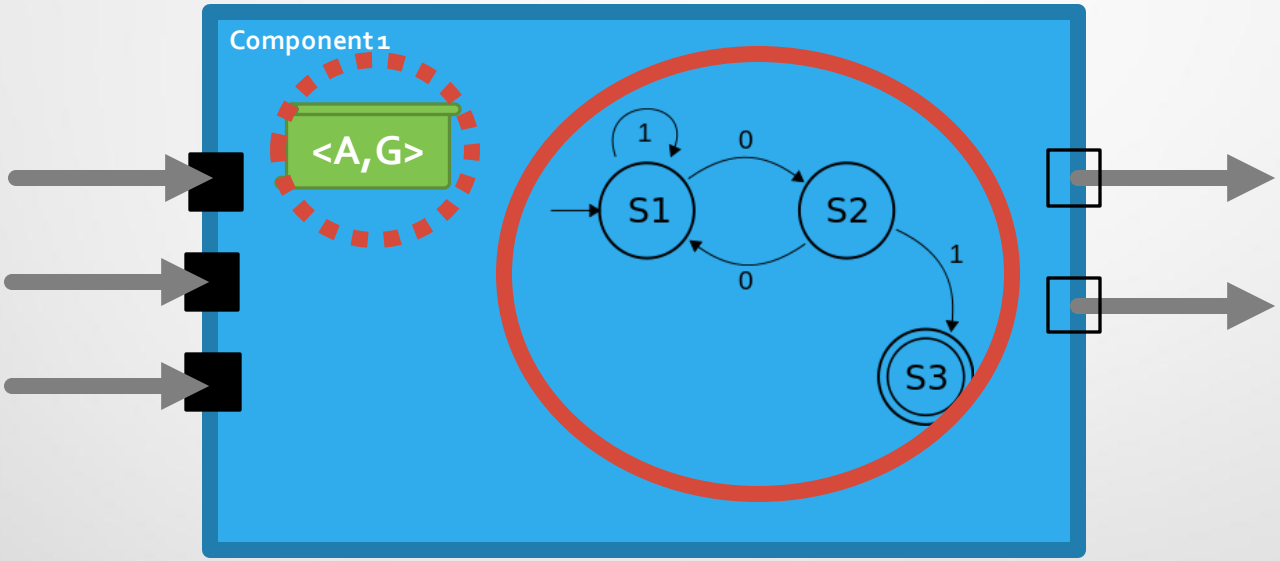
Functional V&V: Check the contract refinement

- Check the contract refinement
  - Guarantees of composite component ensured by contracts of subcomponents
  - Assumptions of subcomponents ensured by contracts of other components



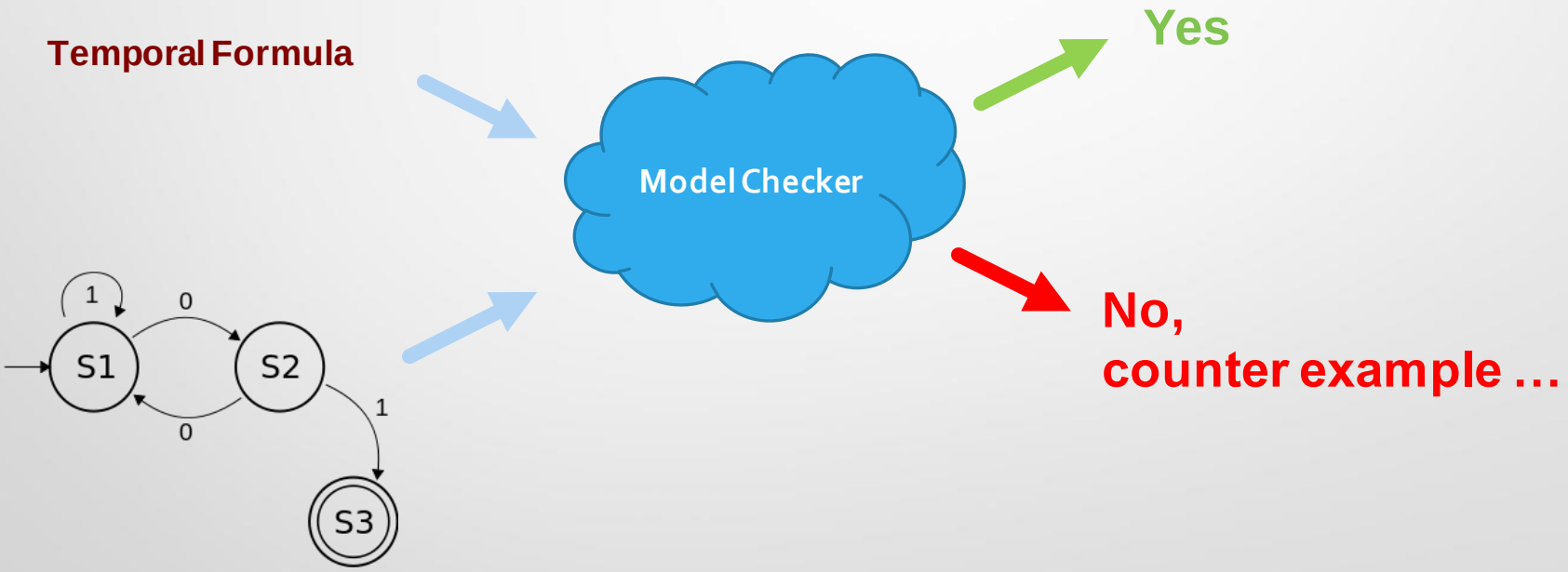
Functional Early Verification & Validation

- Contract-Based Verification of State Machines
  - Verify if every state machine locally satisfies the contracts of the associated component



Functional V&V: V&V of State Machines

- Perform the verification of properties on state machines (model checking, independently on contracts).
- Model Validation (e.g. reachability of all states)

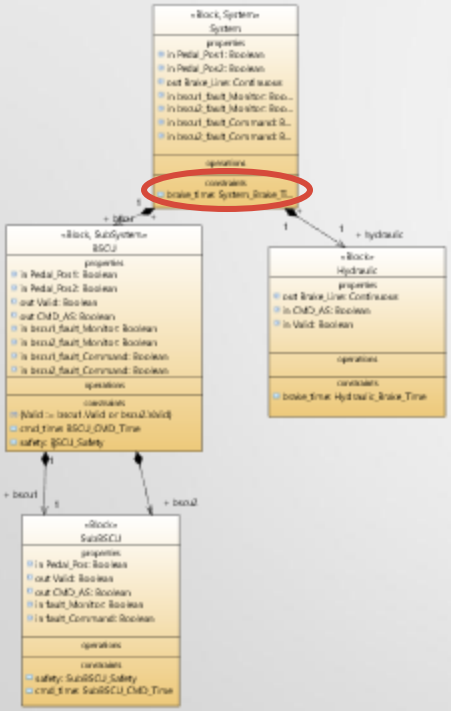


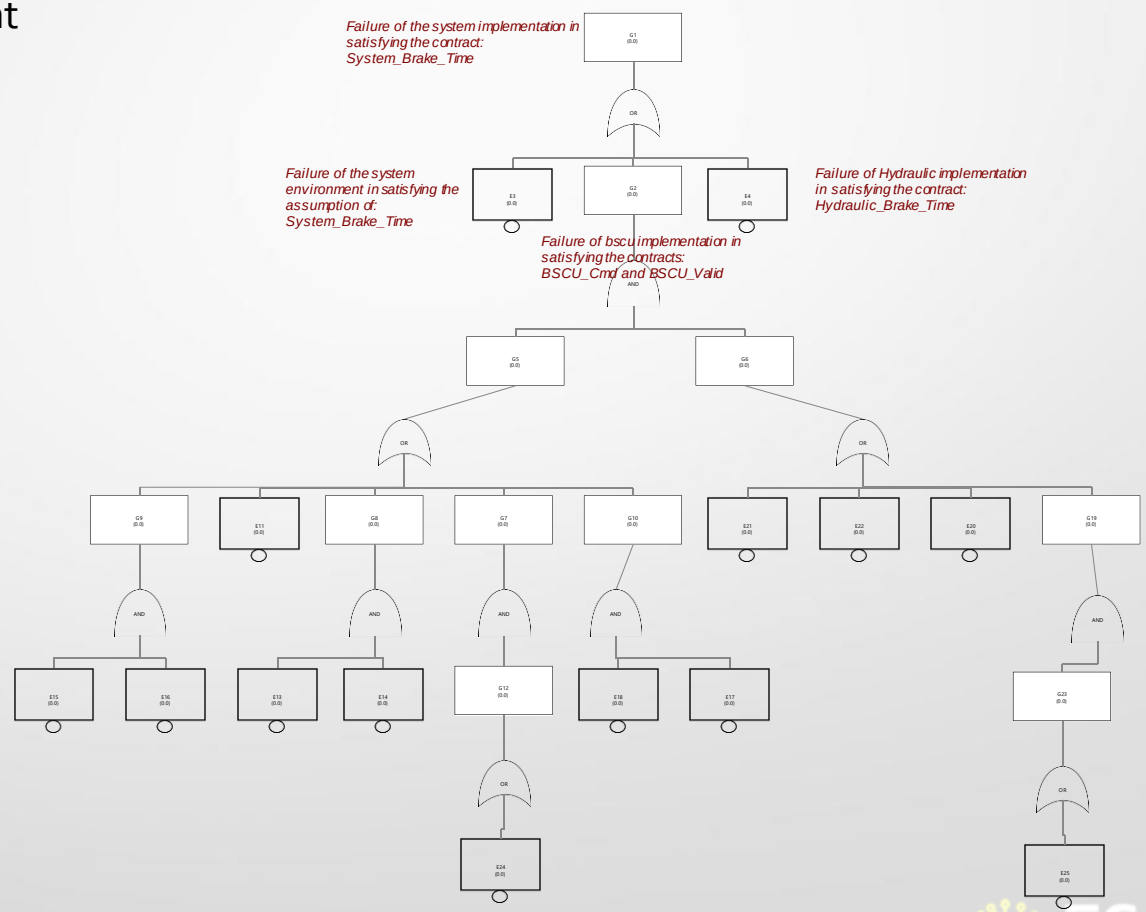
Functionalities currently implemented

- System Design
  - Architecture definition
  - Requirement Formalization
  - Functional Refinement
    - Architectural Refinement
    - Contract Refinement
- Functional Early V&V
  - Validation of contracts
  - Check the contract refinement
  - Contract-Based V&V of State Machines
  - V&V of state machines
- **Safety Analysis**
  - Contract-Based Safety Analysis
  - Model-based Safety Analysis
- Trade-off Analysis
- Document generation

Contract-based Safety Analysis (CBSA)

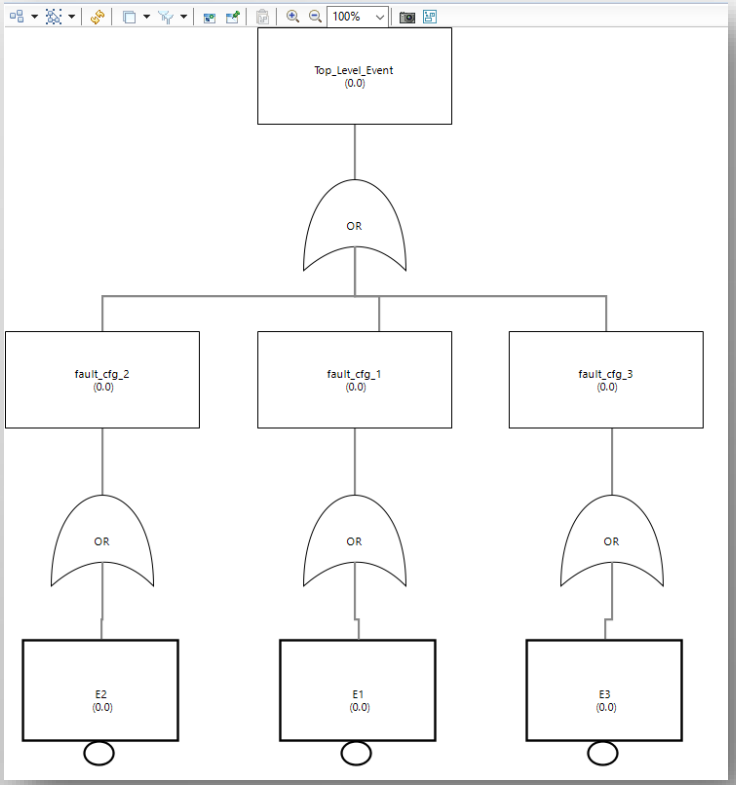
- Generate a hierarchical fault tree, given a contract refinement
  - The top-level event is the failure of the system component
  - The basic events are the failures of the leaf components and the failure of the system environment





Model-based Safety Analysis (MBSA)

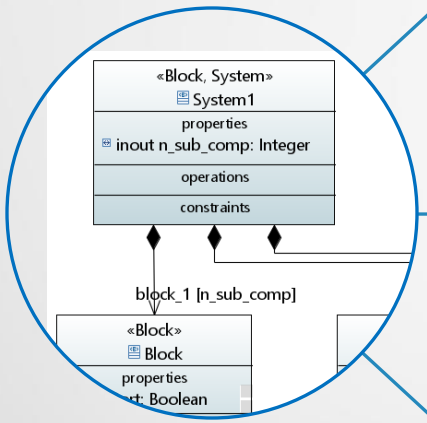
- MBSA analyses the model extended with faulty behavior and searches for the possible combinations that may lead to a system failure.
- Generate fault tree from the extended state machine



Functionalities currently implemented

- Nominal System Definition
  - Architecture definition
  - Requirement Formalization
  - Functional Refinement
    - Architectural Refinement
    - Contract Refinement
    - Parametrized Architecture
- Functional Early V&V
  - Contract-Based V&V of Refinement
  - Contract-Based V&V of State Machines
  - V&V of state machines
- Safety Analysis
  - Contract-Based Safety Analysis
  - Fault Injection and model-based safety analysis
- **Trade-off Analysis**
- Document generation

Trade-off analysis



Conf1

- Prop1
- Prop2

Conf2

- Prop1
- Prop2

Conf3

- Prop1
- Prop2

Functionalities currently implemented

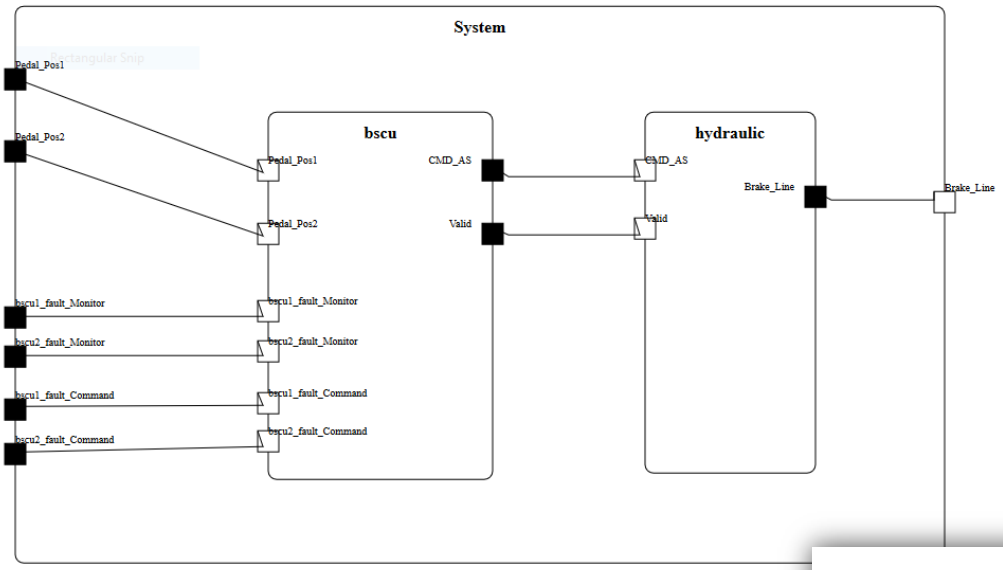
- **Nominal System Definition**
  - Architecture definition
  - Requirement Formalization
  - Functional Refinement
    - Architectural Refinement
    - Contract Refinement
    - Parametrized Architecture
- **Functional Early V&V**
  - Contract-Based V&V of Refinement
  - Contract-Based V&V of State Machines
  - V&V of state machines
- **Safety Analysis**
  - Contract-Based Safety Analysis
  - Fault Injection and model-based safety analysis
- **Trade-off Analysis**
- **Document generation**

Document Generation

- Generate a document in html format, including:
  - Textual information about:
    - Components
    - Ports
    - Attributes
    - Formal Properties
    - Contracts
  - Diagrams exported from CHESS as vector images format.
  - Results of the verification, validation, and safety analysis (work in progress)

Document Generation

System



Component description:

Name	Type	Notes
System		

Input ports:

Name	Type	Range
Pedal_Pos1	*Boolean*	*[PortRange]*
Pedal_Pos2	*Boolean*	*[PortRange]*
bscu1_fault_Monitor	*Boolean*	*[PortRange]*
bscu2_fault_Monitor	*Boolean*	*[PortRange]*
bscu1_fault_Command	*Boolean*	*[PortRange]*
bscu2_fault_Command	*Boolean*	*[PortRange]*

Output ports:

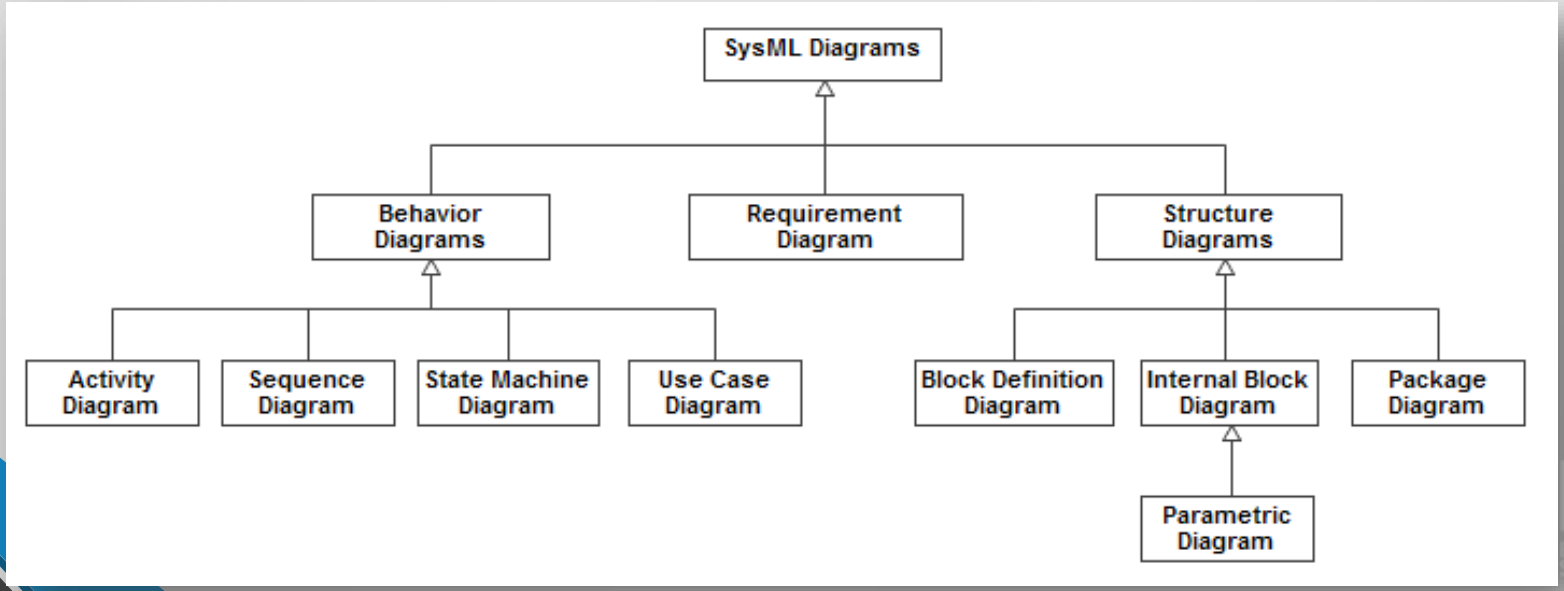
Name	Type	Range
Brake_Line	*Continuous*	*[PortRange]*

Details: CHESS Project structure

- CHESS Project is structured in 5 main views (packages):
  - Requirement View: informal requirements definition
  - System View: logical components and contracts definition
  - Component View: software components definition
  - Deployment View: hardware components definition
  - Analysis View: analysis configurations definition

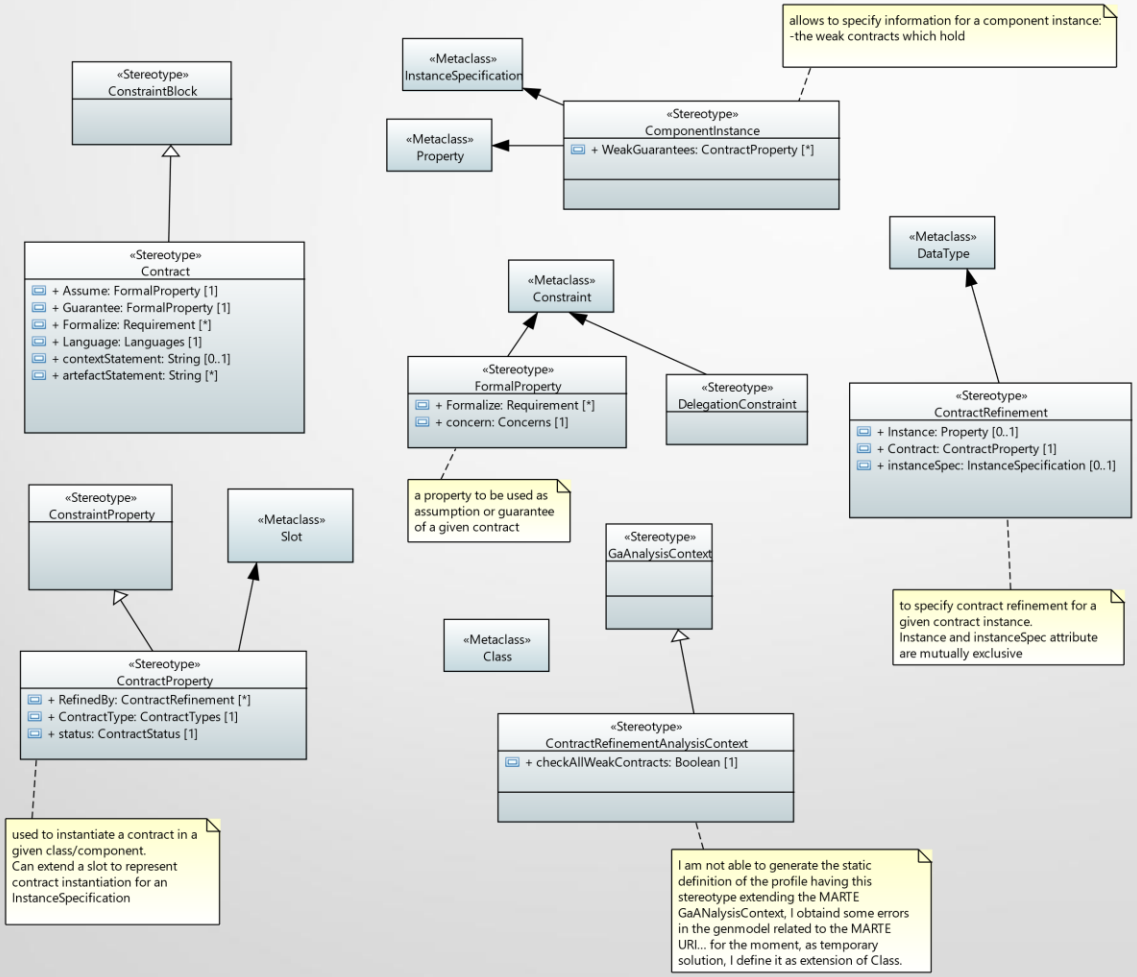
Details: CHESS Profile

- A profile in the UML provides a generic extension mechanism for customizing UML models for particular domains and platforms.
- The Systems Modeling Language (SysML) is a general-purpose modeling language for systems engineering applications.
- SysML allows the user to model:
  - Informal Requirements
  - System Architecture
  - System behavior



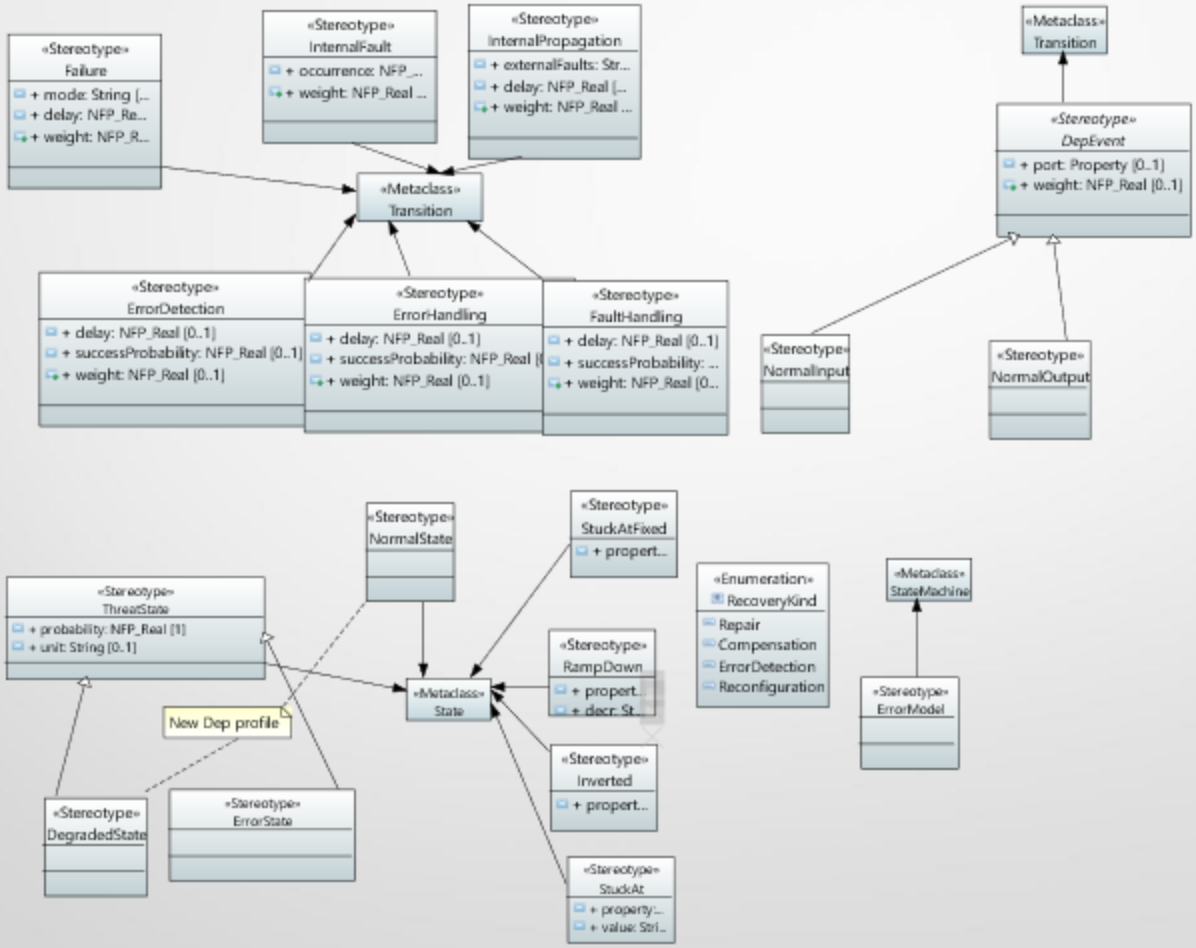
Details: CHESS Profile

- SysML model extended with Formal Methods concepts



Details: CHESS Profile

- SysML model extended with faulty behavior/fault injection



Details: Formal Specification Languages

- OSS (OCRA System Specification): it is used to specify the components, their ports and contracts, and their decomposition.
- For the contract definition, it involves a Linear-time Temporal Logic Language (LTL with future and past operators)

```
COMPONENT example1 system
        INTERFACE
                INPUT PORT in_data: boolean;
                OUTPUT PORT out_data: boolean;
                CONTRACT reaction
                        assume: in the future in_data;
                        guarantee: always (in_data implies in the future out_data);
        REFINEMENT
                SUB a: A;
                SUB b: B;
                CONNECTION a.in_data := in_data;
                CONNECTION b.in_data := a.out_data;
                CONNECTION out_data:= b.out_data;
                CONTRACT reaction REFINEDBY a.reaction, b.pass;
COMPONENT A
        INTERFACE
                INPUT PORT in_data: boolean;
                OUTPUT PORT out_data: boolean;
                CONTRACT reaction
                        assume: in the future in_data;
                        guarantee: always (in_data implies in the future out_data);
COMPONENT B
        INTERFACE
                INPUT PORT in_data: boolean;
                OUTPUT PORT out_data: boolean;
                CONTRACT pass
                        assume: true;
                        guarantee: always (in_data implies out_data);
```

Details: Formal Specification Languages

- NUSMV (a new Symbolic Model Verifier) language: it is designed to describe the system behavior.
- It allows description of completely synchronous to asynchronous systems, detailed to abstract systems.

```
MODULE main
VAR
        request: boolean;
        state: {ready,busy};
ASSIGN
        init(state) := ready;
        next(state) :=
                case
                        state=ready & request: busy;
                        TRUE: {ready,busy};
                esac;
SPEC AG(request --> AF (state = busy))
```