Modern Formal Methods for the Design and Verification of Complex Systems

6th Italian Workshop on Embedded Systems
Rome, 9-10 November 2021

Alessandro Cimatti
cimatti@fbk.eu

Fondazione Bruno Kessler
Center for Digital Industry
Director

The ES Unit at FBK

Fondazione Bruno Kessler (FBK) is a research non-profit public interest entity, located in Trento, Italy.

The **Digital Industry** center focuses on digital technologies for various application domains (aerospace, railway, automotive, energy, agriculture, manufacturing, etc.)

- Research themes: Artificial Intelligence, Edge computing, Formal methods
- Resarch staff > 120, Director: Alessandro Cimatti

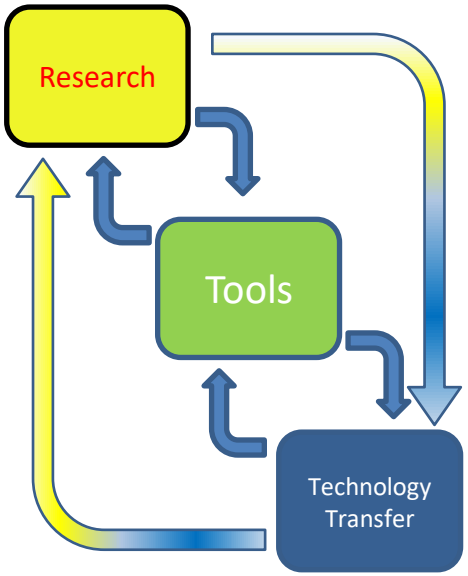
Strategy: strong synergy between basic Research, Tool development and Technology Transfer

Research Units

- Technologies for Vision, 3D Optical Metrology, Open IoT
- Data Science, Software Engineering, Machine Translation
- Embedded Systems
- People: >40, research staff, programmers, PhD students, technologists
- European projects: VALU3S, HUBCAP, AIPLAN4EU, AMASS, CITADEL, D-MILS, …
- Technology transfer: RFI, Bosch, Boeing, NASA, Ansaldo, Intel, others under NDA
- Research focused on formal methods and automated reasoning

Topics:

- Model checking
- Requirements analysis
- Contract-based design
- Fault injection and safety analysis
- Fault detection, identification and recovery (FDIR)
- Planning
- Condition Monitoring
- Runtime verification



Goal and outline of the presentation

Goal:

- How can *modern* formal analysis techniques and tools support model-based design of complex systems

Outline of the talk:

- Formal verification
- Model-based safety assessment
- Contract-based design
- Comprehensive process/tool integration

Part 1:
Functional Verification

Model Checking

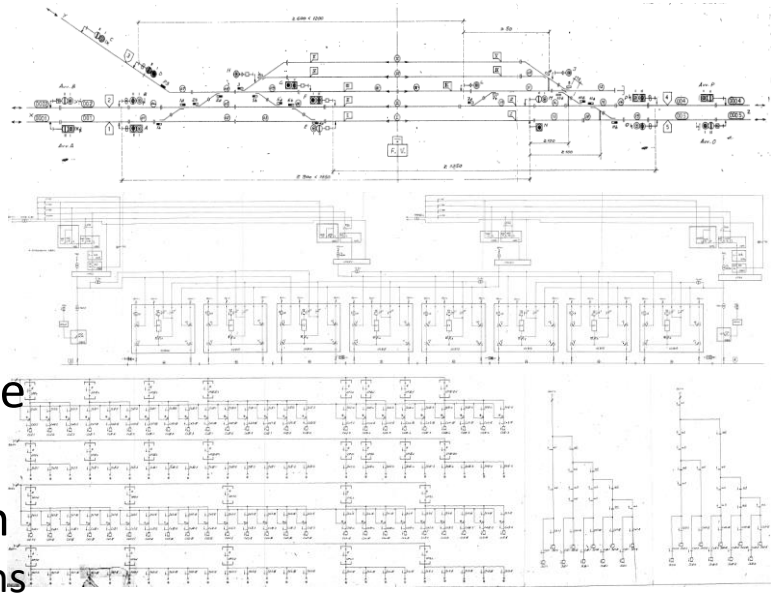
- Does system satisfy Requirements?
  - Systems modeled with a formal semantics
  - Requirements as properties in temporal logic
- System correctness reduced to formal reasoning in mathematical logic
- Model checking proves the property by means of state space exploration and deduction techniques
  - Fully automated
  - Explores all behaviors
  - Feedback as counterexamples or proofs
- Models? From netlists, RTL circuits, software, protocols, high-level languages

Modern Model Checking

- Increased automation: SAT-based verification
  - Based on Boolean reasoning, large capacity
  - Techniques: BMC, Induction, Interpolation, IC3
- Increased expressiveness: from SAT to SMT
  - Satisfiability Modulo Theories
  - Combining Boolean and mathematical reasoning
  - Timed and hybrid systems, RTL, microcode, software
- Fundamental role of abstraction-refinement
  - Implicit predicate abstraction
  - Incremental linearization
- Additional features
  - Proof production for certification
  - Parameter synthesis for design space exploration

A flagship application in the Railways domain

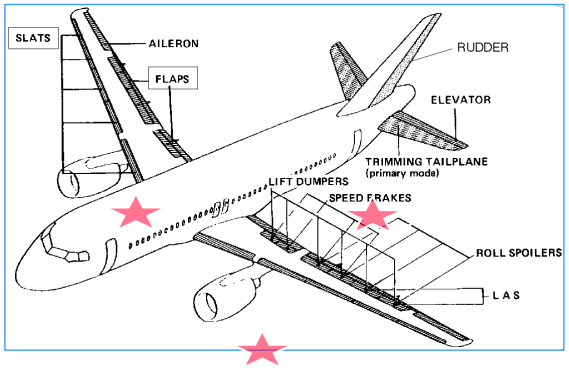
- New generation Station interlocking
- Two key objectives:
  - designing the new solution
  - dealing with the legacy
- Model-based design flow for new computer-based solution
  - Requirements in Structured Natural Language
  - Model checking of FSM and Software
  - Key challenge: dealing with parameterization
    - VV model: first V domain, second V's applications
- Legacy: relay interlocking systems
  - Complex modeling from circuit schematics
  - Compilation into hybrid automata
    - real time, clocks, electrical quantities
- Ongoing challenges
  - Relay-to-software traceability
  - Digitalization of legacy printouts

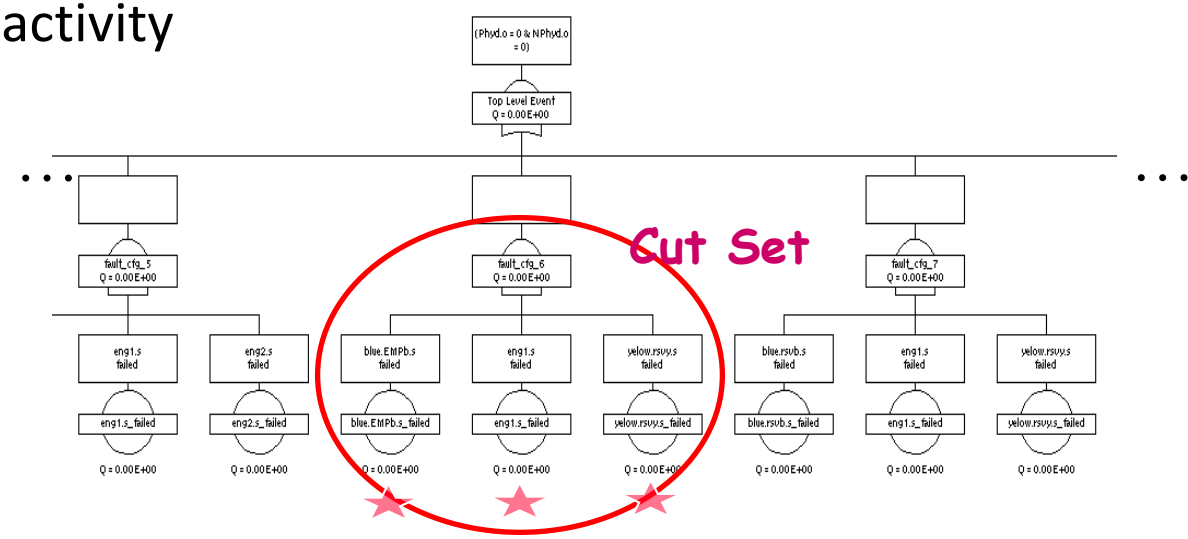


Part 2:
Safety Assessment

Safety assessment

- How does a system respond to faults?
  - Hazard analysis, PASA/PSA, FTA, FMEA, CCA, ...
- Analysis of system behavior in presence of faults
  - Fault Tree Analysis (FTA)
  - Failure Modes and Effects Analysis (FMEA)
  - Common Cause Analysis (CCA)
- Typical problems
  - PB1: misalignment between design and SA, out of sync artifacts
  - PB2: labor-intensive activity



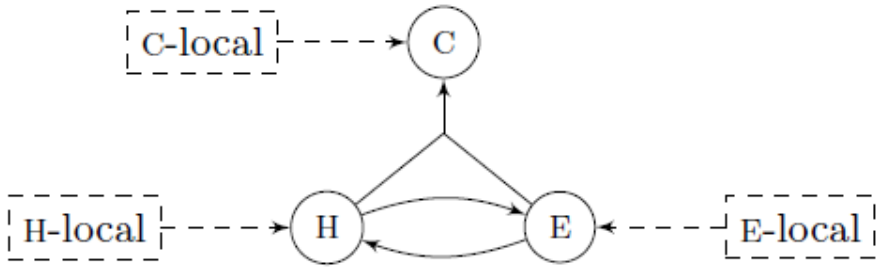


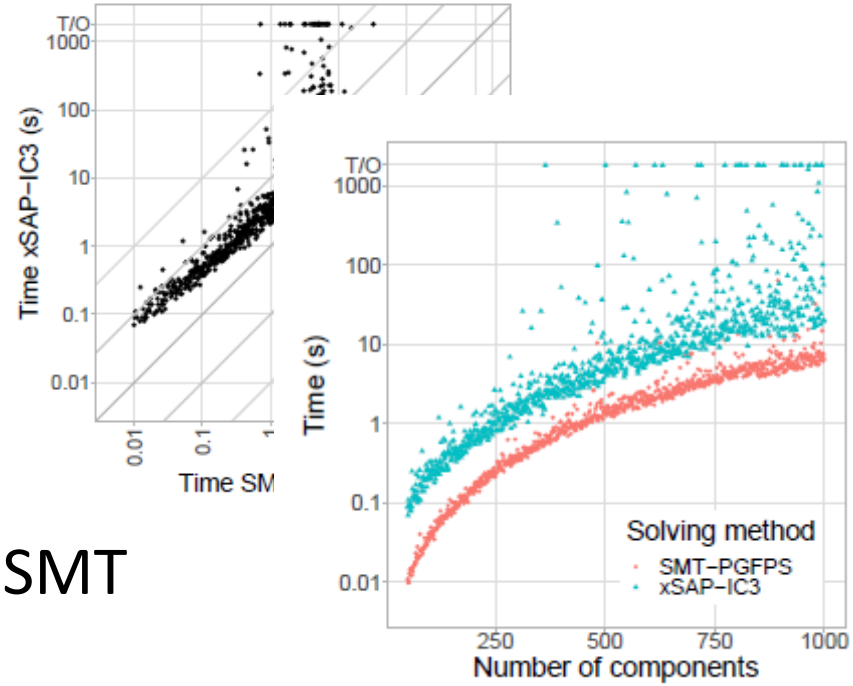
Model-based safety assessment

- Use formal techniques for modeling and analysis of safety-related aspects
- Model includes non-nominal, faulty behaviors
  - Fault variables trigger non-nominal behaviors
  - Use model checking techniques for automation
- Automated techniques for minimal cut set analysis
  - collect configuration of assignments to fault variables causing the feared (top-level) event
    - TLE as violation of the requirement
  - Extract minimal cut set, iterate
  - Results: fault trees, FMEA tables, reliability measures

High-level Propagation Analysis

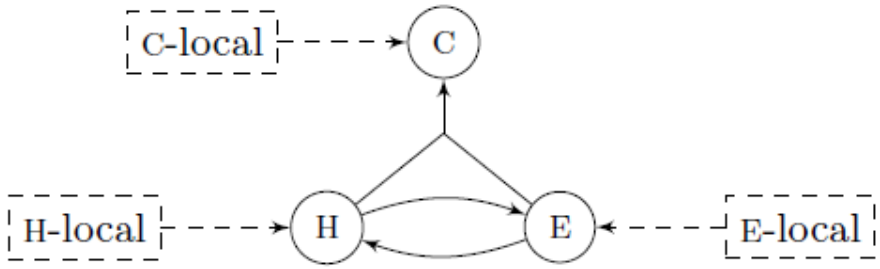
- Abstract view of subsystems
  - endogenous failures
  - propagated failures
  - and-or dependencies
- Challenges
  - Complex fault models
  - Cyclic structures
  - Timing dependencies
  - Mode dependencies
- Scalability?
  - Sequential semantics
  - Reduction to combinational SMT yields great speed-ups

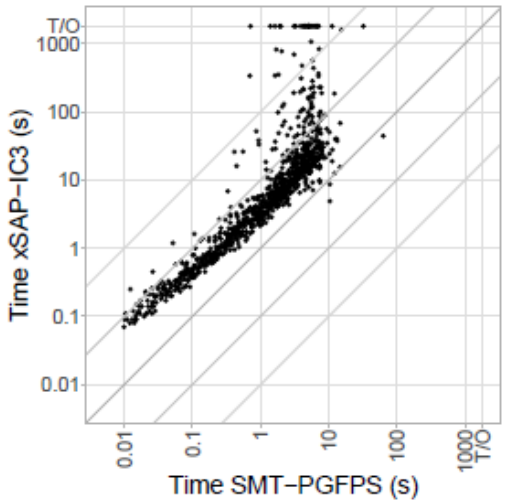




High-level Propagation Analysis

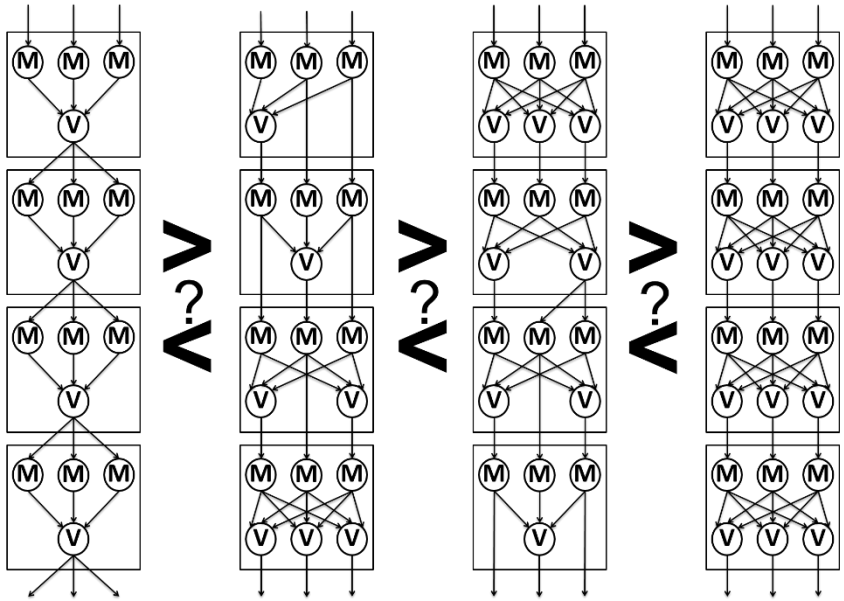
- Abstract view of subsystems
  - endogenous failures
  - propagated failures
  - and-or dependencies
- Challenges
  - Complex fault models
  - Cyclic structures
  - Timing dependencies
  - Mode dependencies
- Scalability?
  - Sequential semantics
  - Reduction to combinational SMT yields great speed-ups

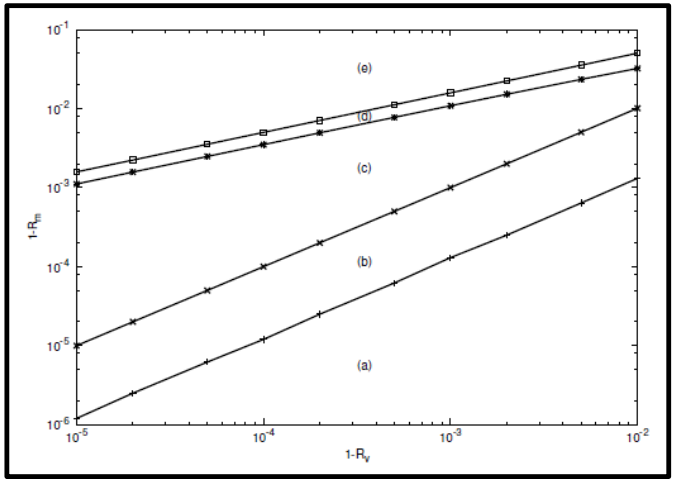




Reliability Analysis of Redundancy Architectures

- Does a solution based on redundancy (e.g. TMR) help?
  - SMT-based techniques for cut-set analysis
  - Symbolic Reliability computation
- Design-Space Exploration
  - assess quality of deployment configurations
  - find best deployment configuration
  - multi-objective optimization
    - cost, weight, power, reliability





Behavioral Model-Based Safety Analysis

- Start from model of nominal behaviors
- Automatically inject faults
  - Fault-extension directives
  - Library-based approach
- Formal Analyses of extended system model
  - Fault Tree Analysis (FTA)
  - Failure Modes and Effects Analysis (FMEA)
  - Common Cause Analysis (CCA)
- PB1? Extended model aligned by construction to nominal model!
- PB2? Model checking engine ensures scalability!

Model-Based Safety Analysis

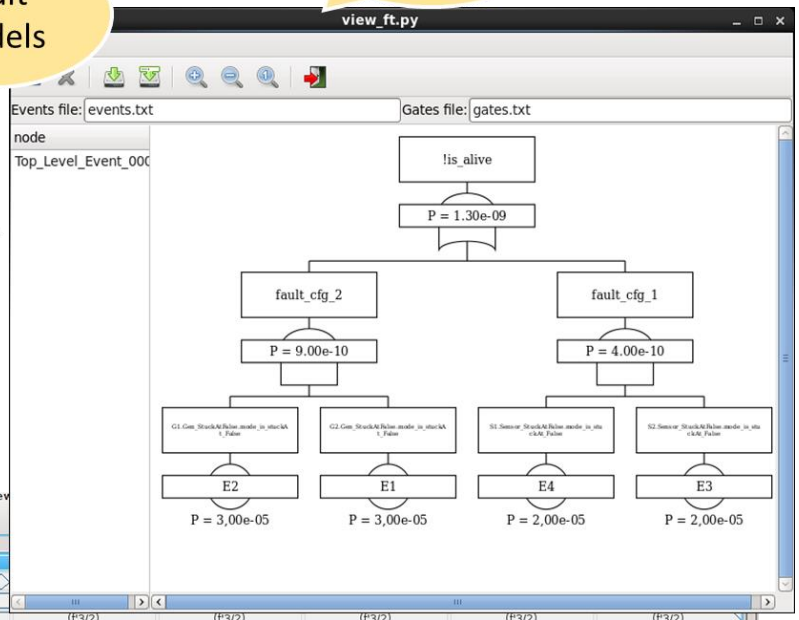
```
MODULE power_system_unit
VAR
  gen1: generator;  gen2: generator;
  voltage_out : real;
ASSIGN
  voltage_out := case
    gen1.voltage_out >=
gen2.voltage_out: gen1.voltage_out;
    TRUE: gen2.voltage_out;
  esac;
```

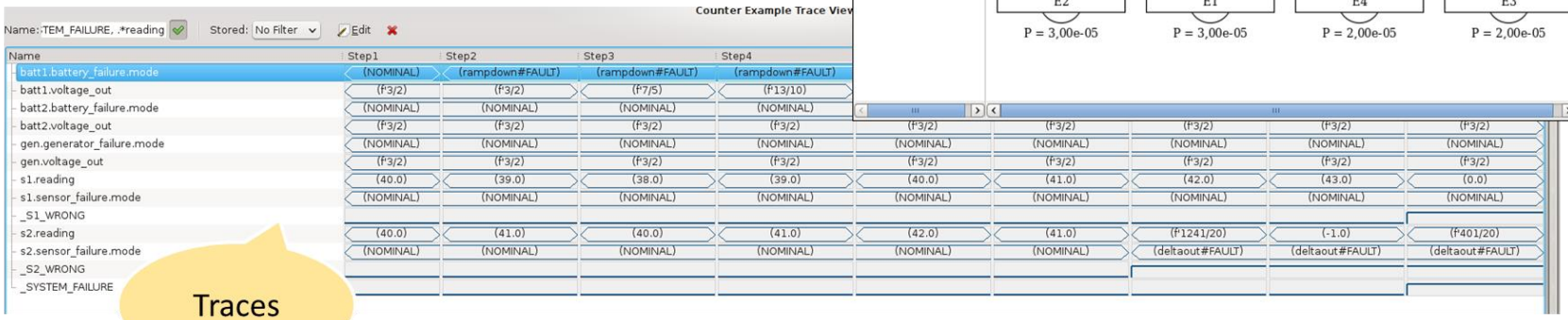
Nominal model

```
EXTENSION OF MODULE generator
    /-- Description of Fault Slice
generator_failure --/
    SLICE generator_failure AFFECTS
voltage_out WITH
MODE rampdown {1.5e-6} : Permanent
.ampDown(
            data decr << 0.1,
            data end_value << 0.0,
            data input << voltage_out,
            data varout >> voltage_out,
            event failure);
```

Fault models

Fault trees

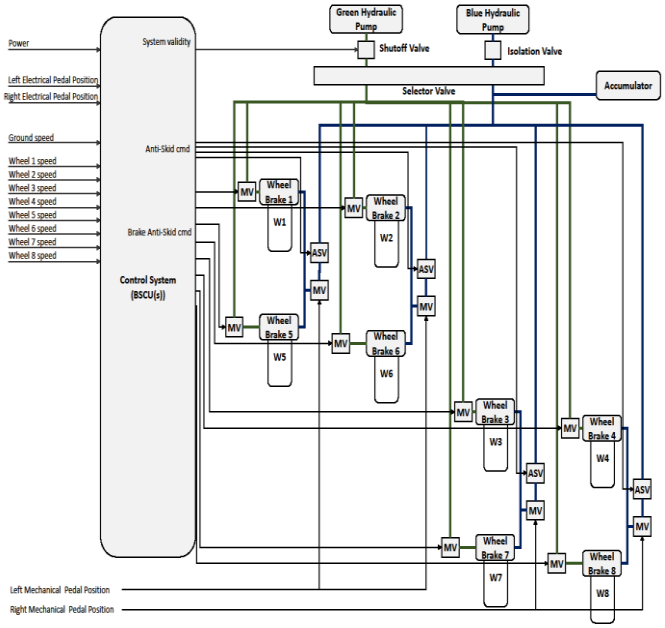




Traces

AIR6110 Wheel Brake System

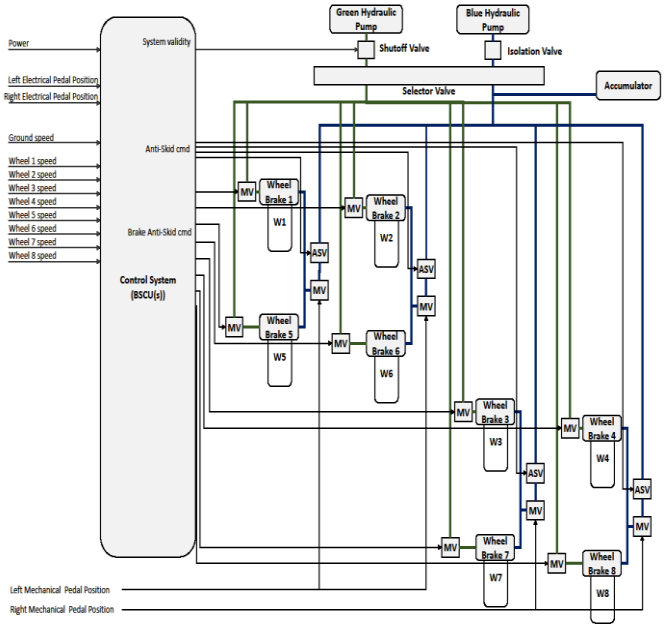
- Aerospace Information Report 6110:
  - Traditional Aircraft/System Development Process Example
  - Describes the development process of a Wheel Brake System for a fictional dual-engine aircraft
  - Analyzes different architectures with standard informal techniques
- Objectives:
  - Analyze the system safety through formal techniques
  - Repeat AIR6110 steps with formal techniques to demonstrate the usefulness and suitability of formal techniques for improving the overall traditional development and supporting aircraft certification
- Main features of the system
  - Control brake for aircraft wheels
  - Redundancy
    - Multiple BCSU
    - Hydraulic plants
  - Functions
    - Asymmetrical braking
    - Antiskid (single wheel/coupled, depending on control mode)
- Review of the AIR6110 with:
  - Formal modeling
  - Formal Verification & Validation
  - Formal Safety Assessment



AIR6110 Wheel Brake System

- Aerospace Information Report 6110:
  - Traditional Aircraft/System Dev… P… le
  - Describes t… ment p… st… fictional dua…
  - Analyze… diff…
- Obje…
  - Analyz…
  - ove… …ertificatio…
- Main…
  - Control bra… t…
  - Redundancy
    - Multipl… CSU
    - Hydraulic plants
  - Functions
    - Asymmetrical braking
    - Antiskid (single wheel/coupled, depending on control mode)
- Review of the AIR6110 with:
  - Formal modeling
  - Formal Verification & Validation
  - Formal Safety Assessment

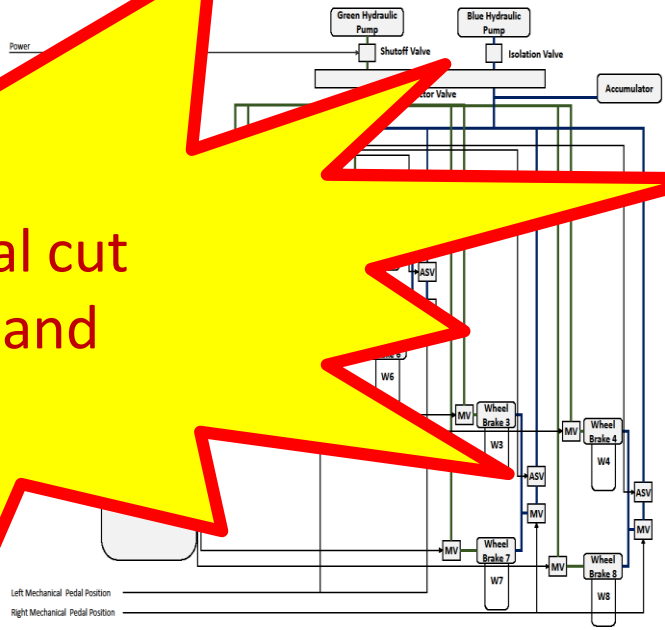
"Automated Fault Extension results in tremendous increase in productivity"



AIR6110 Wheel Brake System

- Aerospace Information Report 6[illegible]10:
  - Traditional [illegible]ircraft/System Dev[illegible]le
  - Describes t[illegible]ment p[illegible]st
    fictional dua[illegible]
  - Analyze[illegible]
- Obje[illegible]
  - Analy[illegible]
- Main[illegible]
  - Redundancy
    - Multiple[illegible]CSU
    - Hydraulic plants
  - Functions
    - Asymmet[illegible]
    - Antiskid (single[illegible]
- Review of the AIR6110 [illegible]
  - Formal [illegible]
  - Formal Verificati[illegible]
  - Formal Safety Assess[illegible]

"Automated Fault Extension results in tremendous increase in productivity"

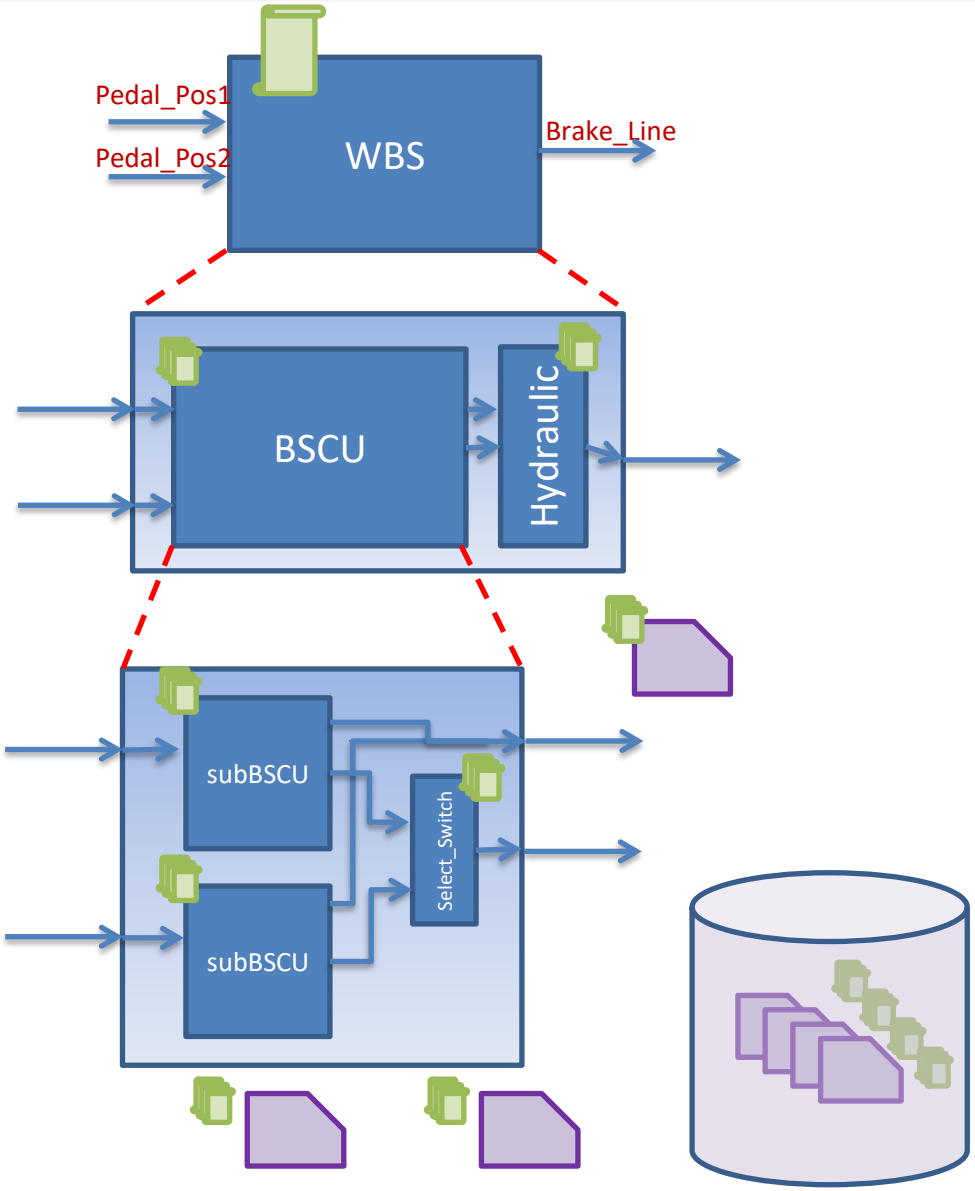


O(100K) minimal cut sets, degree 7 and above

Part 3:
Contract-Based Design

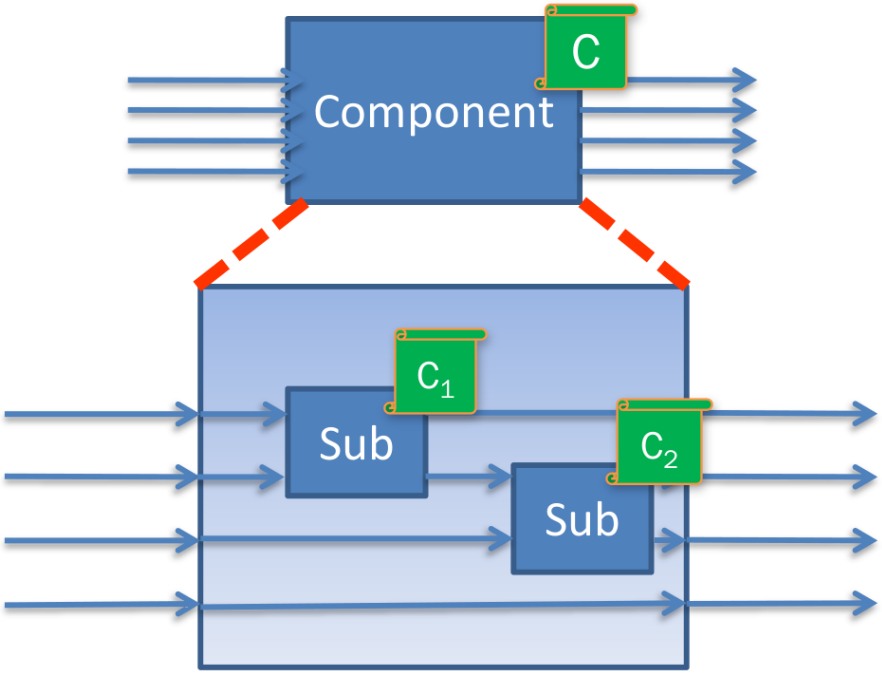
Contract-based design

- Contracts used to specify assumptions and guarantees
  - First conceived for software, now popular also for system architectural design
- Assumptions and guarantees are properties respectively of the environment and of component
- Can be seen as assertions for component interfaces.
- Contracts used for:
  - Early validation of refinement
  - Composition verification
  - Ensuring correct reuse



Contract-refinement Calculus

- Component decomposed into connection of subcomponents
- Contracts <A,G> as temporal logic formulae, written in controlled natural language
- Top level contract follows from contracts of subcomponents
  - C_1 & C_2 -> C



- Assumptions on component must be satisfied
  - A & C_1 & ...& C_{n-1} -> A_n

Contract-Based Verification of Automotive Software

- Ongoing joint project w/ Evidence
- Integration of contract-based design within Huawei AUTOSAR design environment
- Translation of Architecture to OCRA diagrams
  - Contract editing
  - Refinement verification
  - Direct mapping on FBK contract-based design tool OCRA
- Behavioral Analysis of Components
  - Does runnable satisfy contracts?
  - Runnables verification via software model checking
  - Direct mapping on FBK software model checker KRATOS

Part 4:
Comprehensive Integration within the design flow

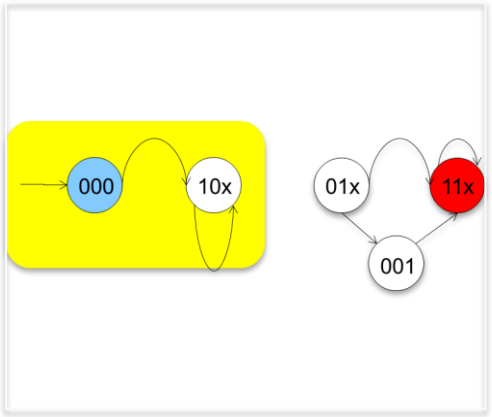
Integration in front-end tools

- FBK MBD tools for model checking, safety assessment and contract-based design have been harmonized and integrated into various AADL- and SysML-based modeling environments:
  - AF3, COMPASS, CHESS, CAMEO
- Integration requires:
  - Extension for needed modeling elements (contracts, fault injection, etc.)
  - Semantic-preserving translation
  - Mapping back of results

FBK Backend Tools

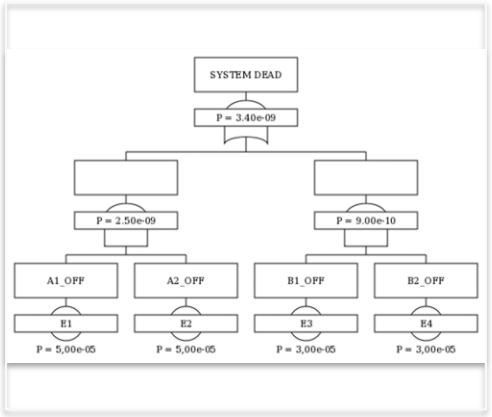
Integrated Design Environment

EST – common integration interface



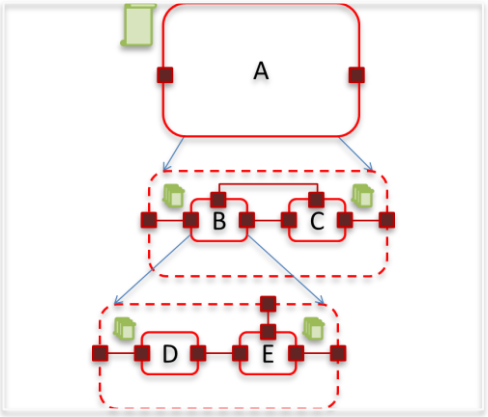
nuXmv, kratos, hycomp, ...

Model checking



xSAP

Model-based safety analysis

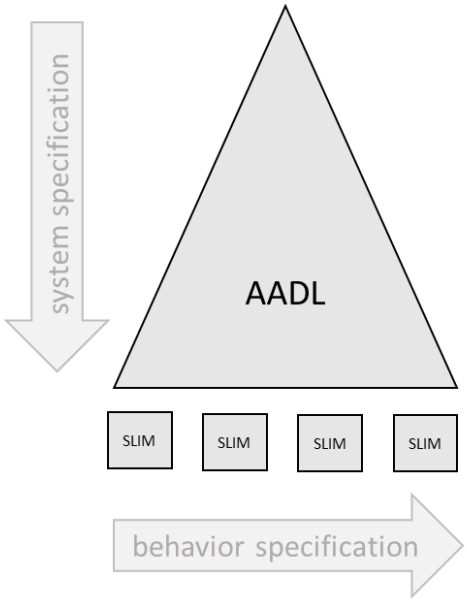


OCRA

Contract-based design

COMPASS: verification + contracts + safety assessment

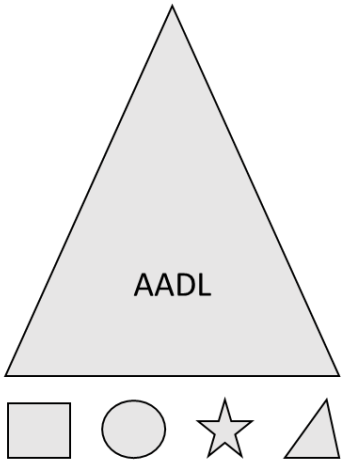
- Toolset for model-based system/SW co-engineering
- Developed in a series of ESA studies (2008-2016)
- Based on a variant of AADL (SLIM)
- Formal design , V&V
  - Requirements specification and analysis
  - Contract-based design
  - Functional verification
  - Fault injection, model extension
  - Safety assessment and dependability, FDIR
- Based on model checking
- Latest release: COMPASS 3.1 (2019)



- Model checking
- Requirements analysis
- Contract-based design
- Fault injection
- FTA/FMEA
- FDIR
- Performability analysis

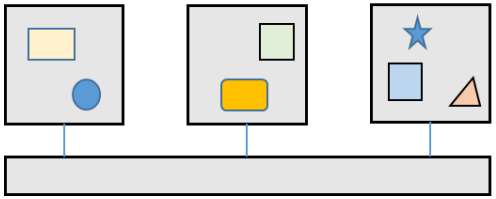
TASTE: model-based design + run-time deployment

- Development environment dedicated to model-based design of embedded, real- time systems
- Created by initiative of ESA in 2008
- Modeling languages
  - AADL for the functional logical architecture (Interface view description)
  - ASN.1 for data abstraction and implementation
  - SDL, Simulink, VDM, … for behavior specification
  - AADL for the physical architecture (Deployment view description)
- Tools: graphical editors, visualizers, code generators and middleware
  - Code generation to C, Ada, supported in the OpenGeode editor and tools such as QGen

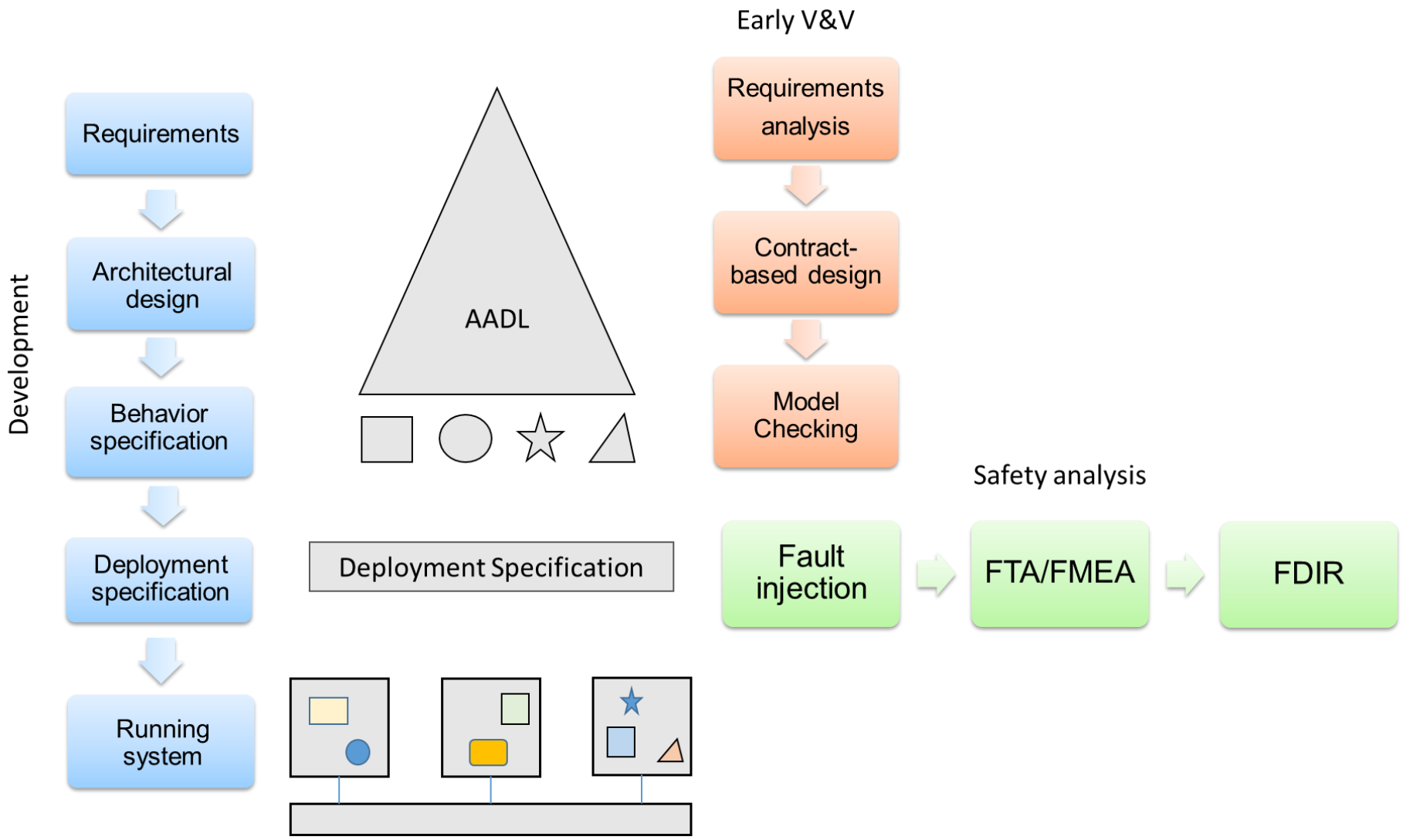


- Many languages (behavior)
- Push-button compilers to deployment
- Graphical editor for AADL
- Graphical editor for SDL
- Integrability via generation of DB, GUI and Python

Deployment Specification



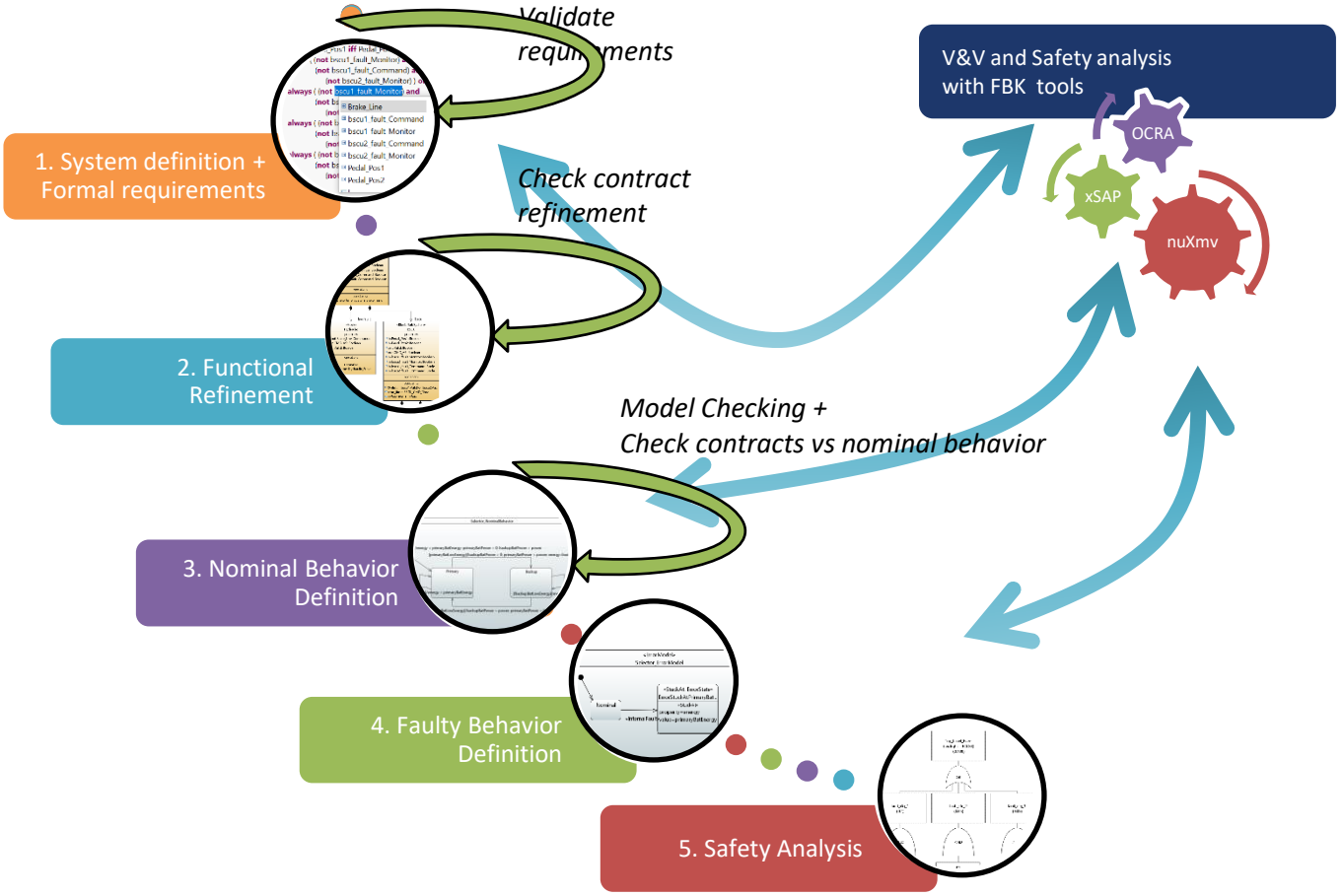
COMPASS+TASTE: integrating the whole flow!



CHESS

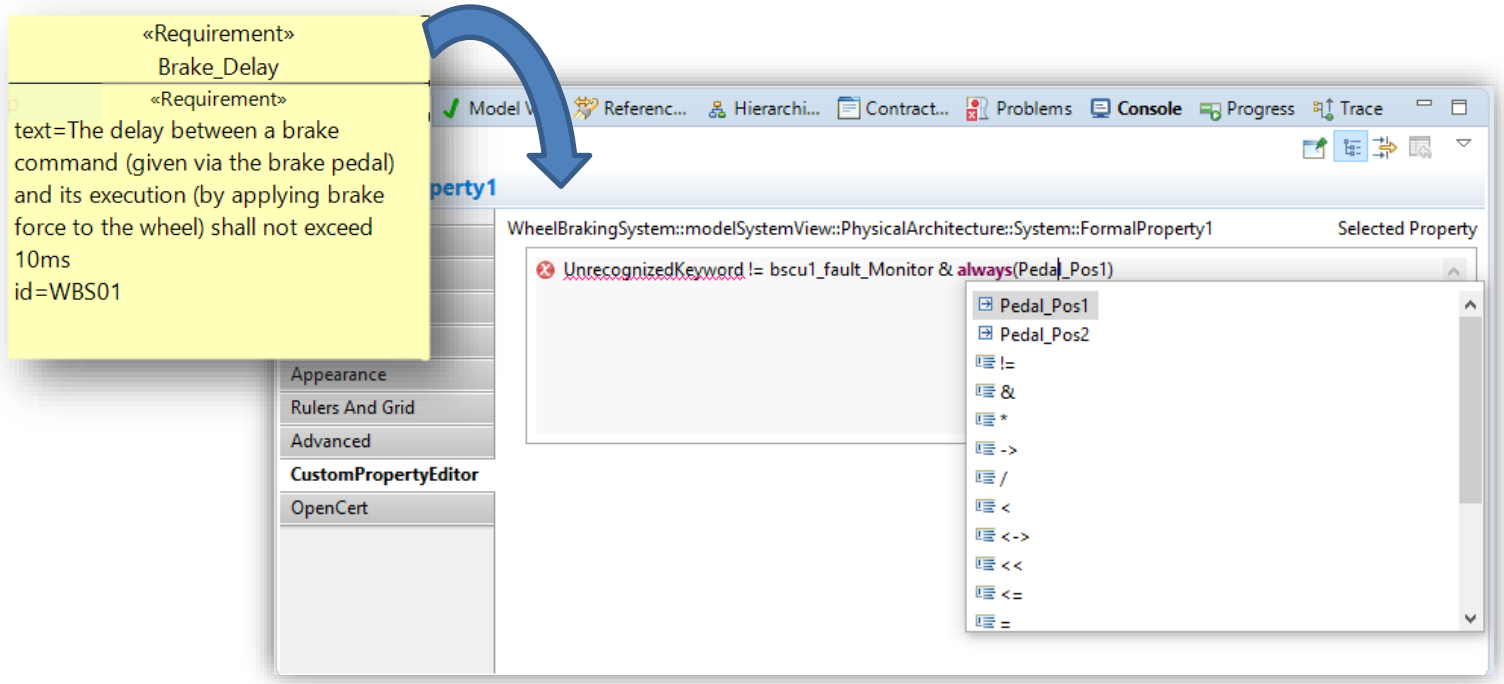
- Model-Based Design of Safety Critical Systems
- UML/SysML diagrams (Papyrus) extended with stereotypes for analysis: formal properties, contracts, error models, etc.
- Analysis supported by backend tools including nuXmv, xSAP, and OCRA
- Implemented in Java and Eclipse
- Open source supported by Eclipse community
- Currently mainly contributed by Intecs and FBK

Architecture Design in CHESS



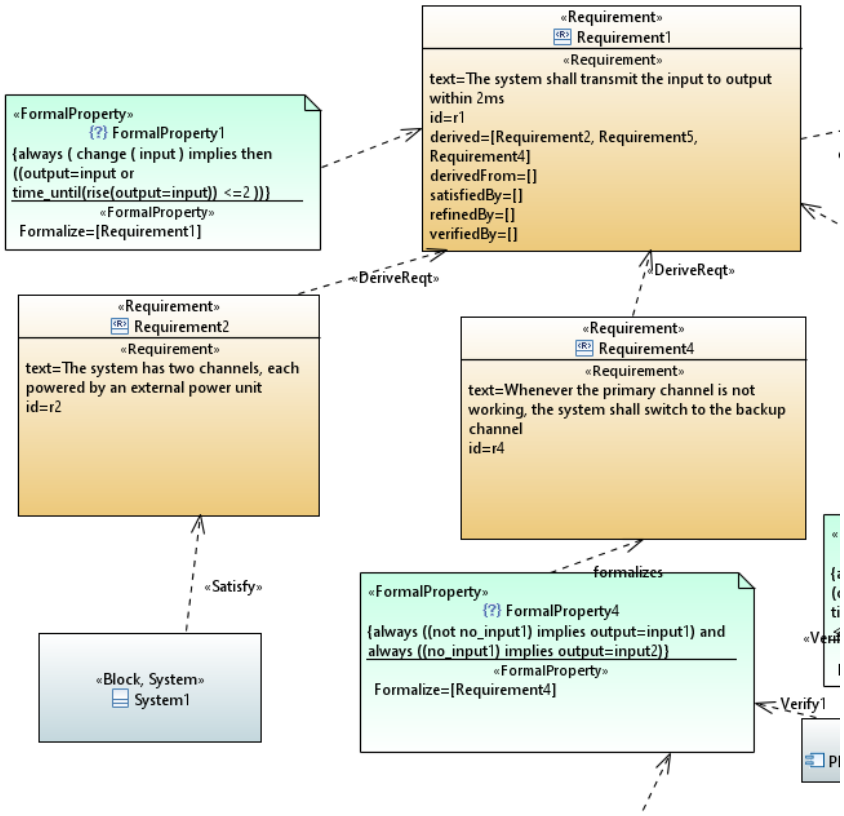
System Design: Requirement Formalization

- Informal requirements represented in SysML Requirements Diagram
  - imported from excel files, csv files, or by using the ReqIF
- Formalized into LTL properties (new stereotype FormalProperty)
  - textual editor with content assistant



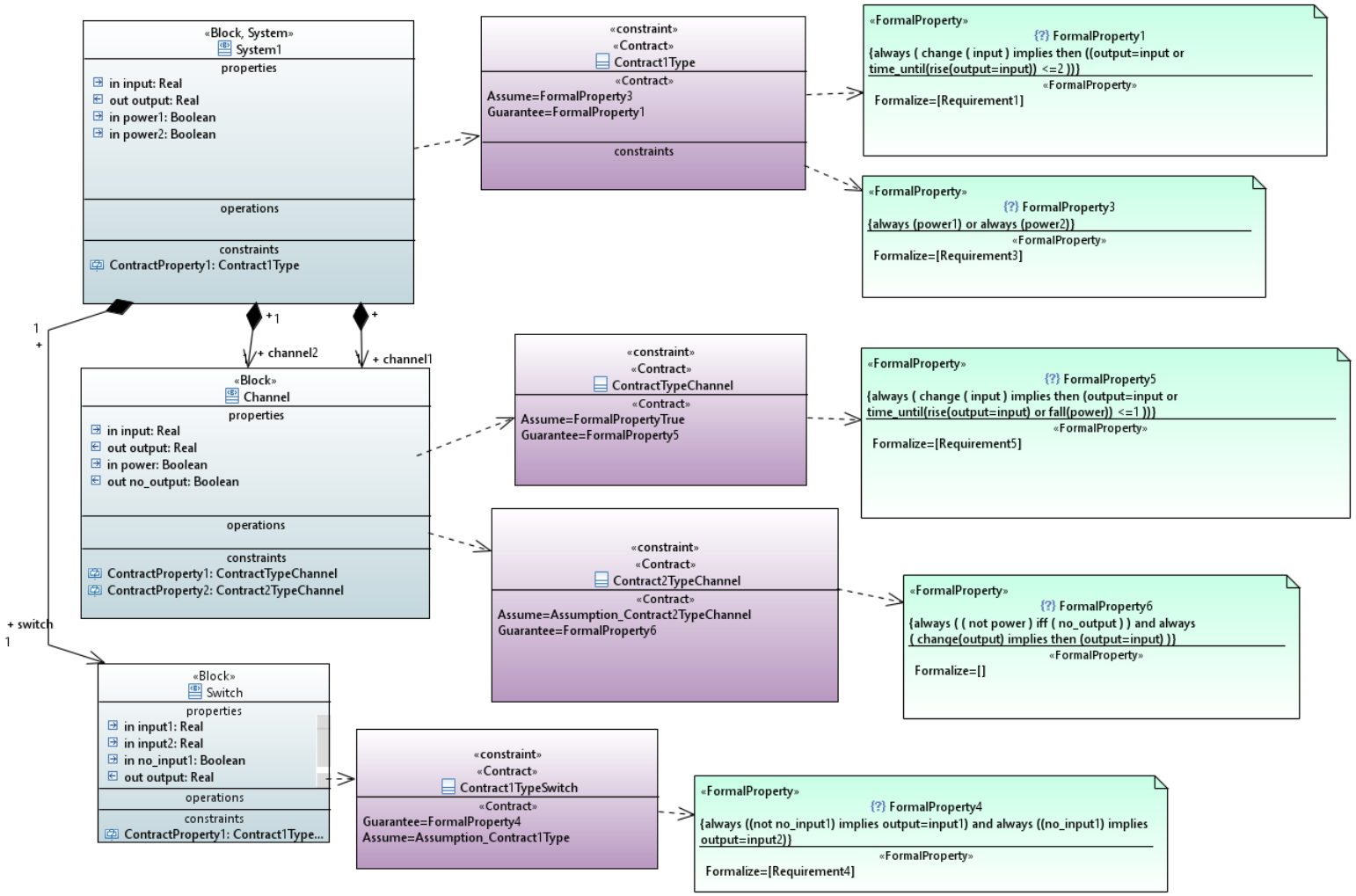
System Design: Requirement Formalization

- Informal requirements represented in SysML Requirements Diagrams
  - imported from excel files, csv files, or by using ReqIF
- Formalized into LTL properties
  - new stereotype FormalProperty
  - textual editor with content assistant
- Requirements diagrams used also to track allocation to components and analysis results



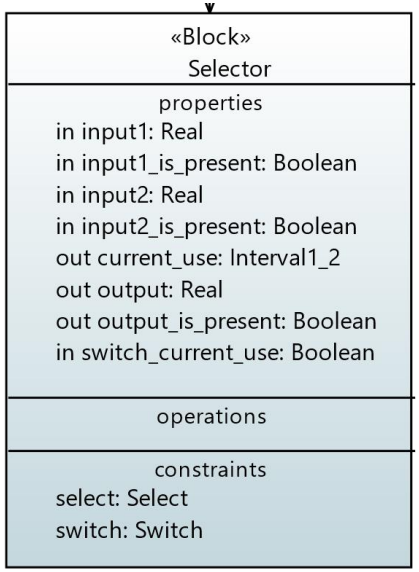
System Design: Architectural Refinement

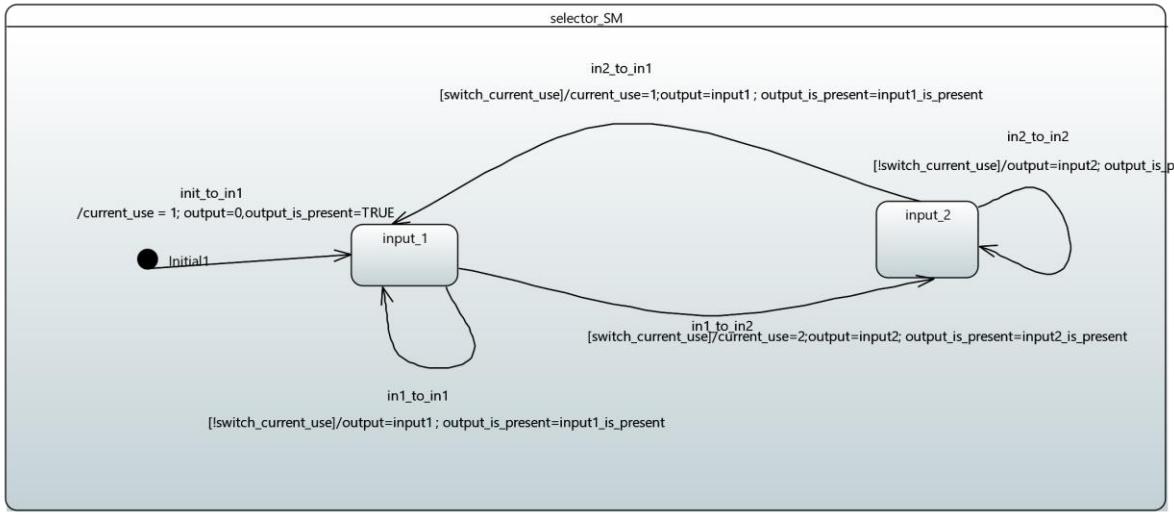
- SysML Block Diagrams extended with contracts



System Design: Nominal Behavior Definition

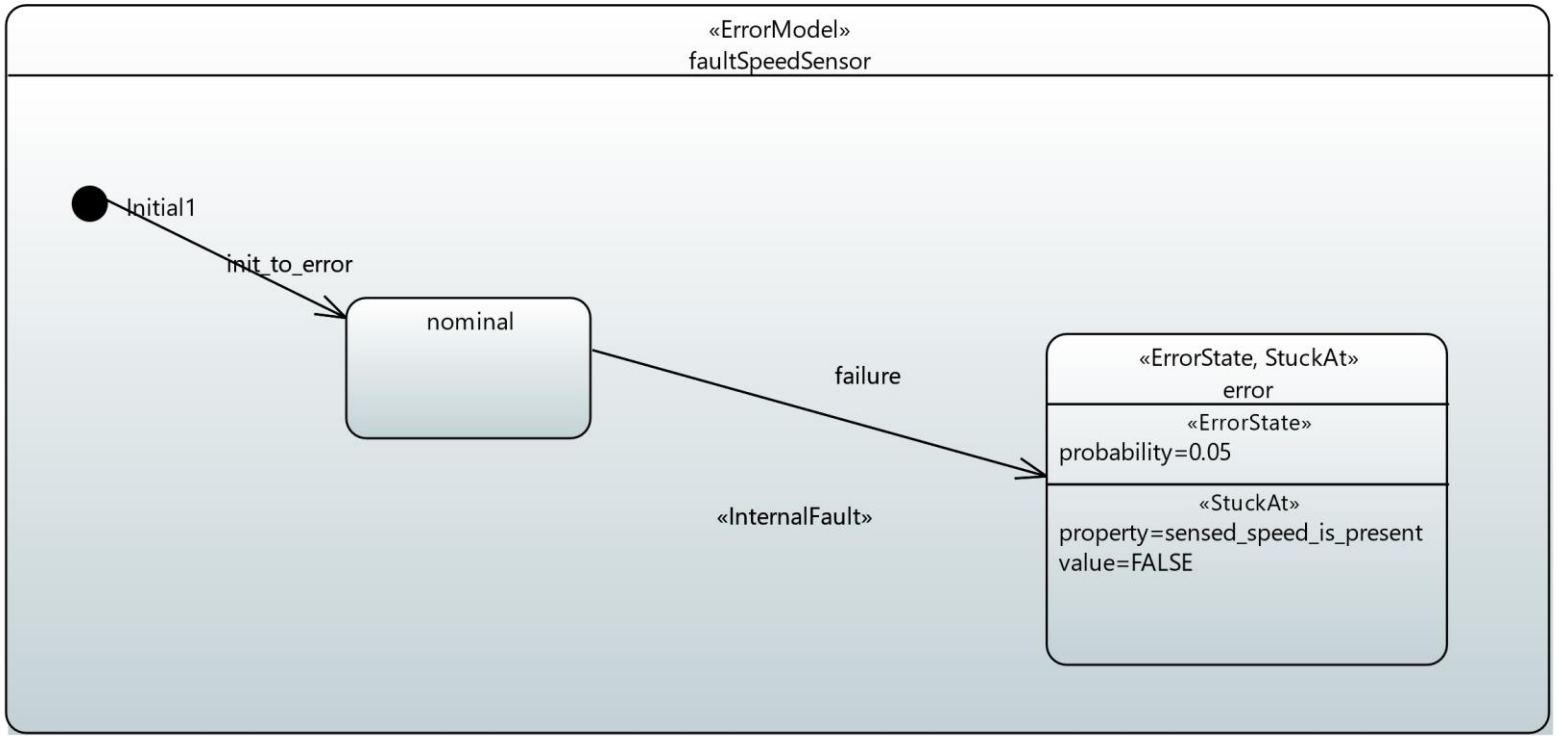
- SysML State Machine Diagrams are used to model the nominal behavior definition of the component.
- A transition comes with a guard and an effect. The guard is a boolean condition upon the values of components properties.





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.



Wrapping up

Conclusions

- Beyond traditional formal methods
  - Does system satisfy requirements
  - Automated abstraction: huge scalability improvements
- Contract-based design
  - formal counterpart of compositional design
  - contracts, contract refinement
  - compositional proofs, parallelization
- Model-based Safety Assessment
  - Does system properly deal with faults
  - Propagation analysis: high-level fault propagation graphs
  - Redundancy analysis
  - Behavioral view: fault injection
- Overarching integration and tool support
- Several practical applications

Future challenges

- Verification of AI-based systems
  - Does system satisfy requirements
  - Automated abstraction: huge scalability improvements
- Tighter integration of Safety Assessment phases
  - from PASA/PSA to behavioral analysis
  - Hierarchical fault trees
- Contract-based design
  - Contract-based Safety Assessment
  - Contract synthesis/discovery
- Hiding formal within traditional design flow

Thanks for your attention

Questions?