CHESS

Contract-based design with the CHESS toolset

Silvia Mazzini, Stefano Puri
Intecs

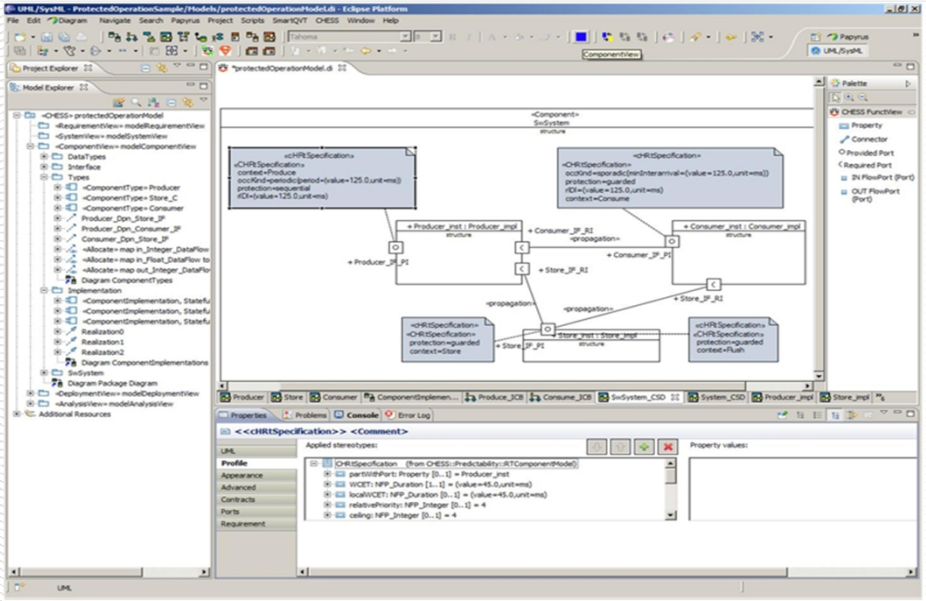
Credits to University of Padua, University of Florence,
Fondazione Bruno Kessler, Mälardalen University Sweden

intecs
the Brainware company

The CHESS Open Source Toolset

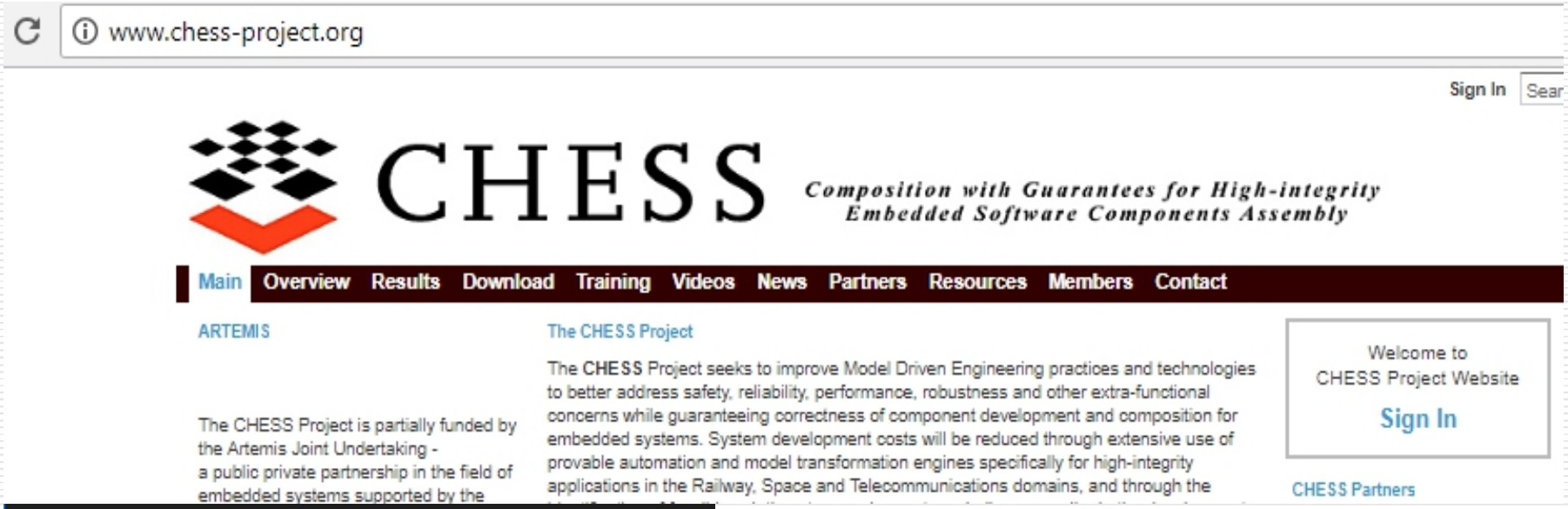
Composition with guarantees for high-integrity embedded software component assembly

- Model based engineering
  - CHESS Modelling Language
  - Based upon Eclipse, UMLPapyrus
- Separation of concerns
  - Functional vs non functional
  - Among design views
- Component based development
  - Specialized to capture the non functional properties of components
    - Real Time
    - Dependability/Safety
- Correctness by construction
  - Extra functional properties are:
    - asserted and verified at design time
    - preserved/guaranteed at run time



CHESS is available as Eclipse Polarsys Project
https://www.polarsys.org/chess/

Main R&D pojects



ARTEMIS

CONCERTO
Guaranteed Component Assembly with Round Trip Analysis for Energy Efficient High-integrity Multi-core Systems

ECSEL Joint Undertaking

European Commission

AMASS
Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems

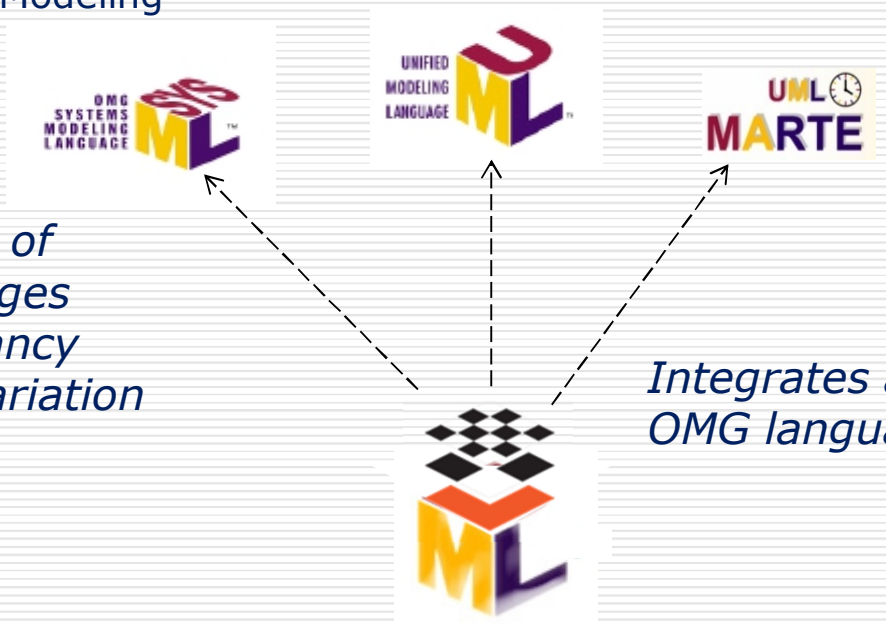
intecs
the Brainware company

The CHESS Modeling Language

Standard profile for System (and Requirements) Modeling

Standard Unified Modeling Language

Standard profile for Modeling and Analysis of Real-Time and Embedded Systems

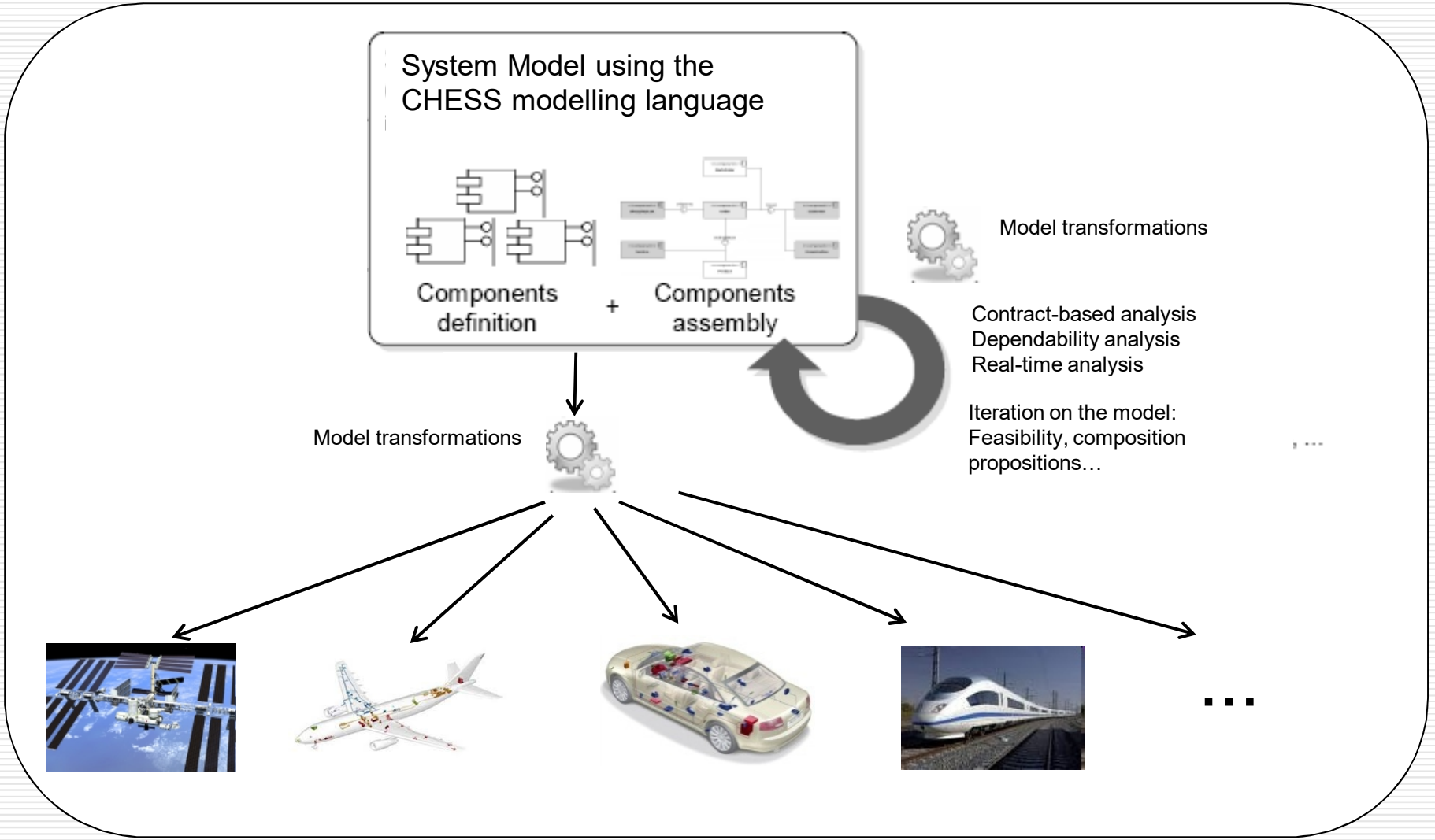


Imports subsets of standard languages
✓ avoid redundancy
✓fix semantic variation points

Integrates and extends standard OMG languages

*In addition, it provides a profile for **Dependability** and **Contract-based modelling***

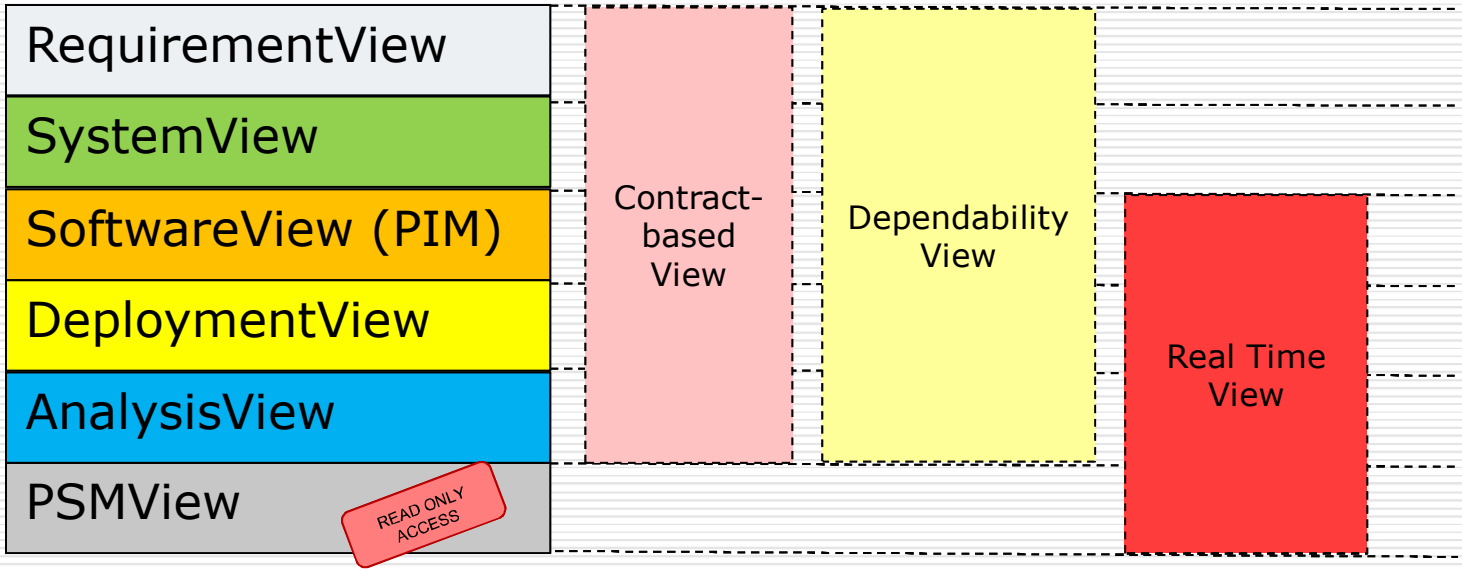
The CHESS methodology– high level view



Major Capabilities and Analysis Tools

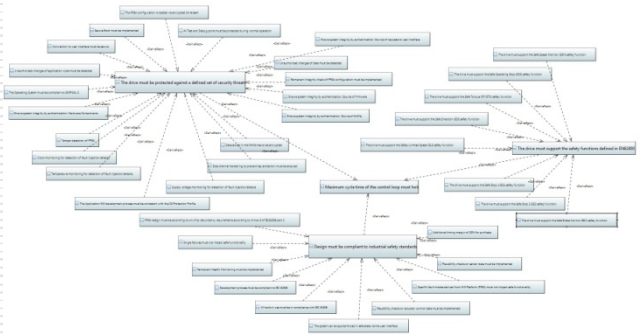
- Model consistency checks
- Failure Propagation Analysis and FMEA/FMECA generation
- State-based Dependability Analysis (by DEEM integration)
- Contract-based Design and Analysis (by OCRA, nuXmv and XSAP integration)
- Safety case generation (by OpenCert integration)
- Real time analysis (by MAST integration)
  - Schedulability and end-to-end response time analysis (with multi-core support)
  - Back propagation of analysis results
- Domain specific needs
  - IMA support
  - AUTOSAR support
- Code generation for Ada (and C)
- Support for run-time monitoring

CHESS Design Views

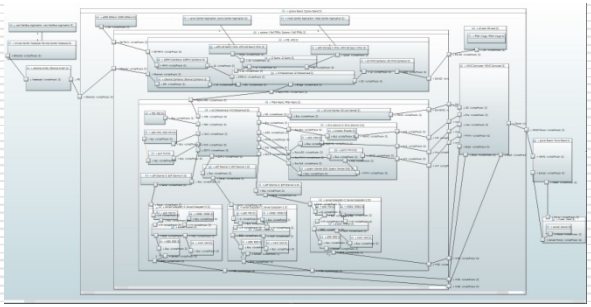


Bird Flight on Views

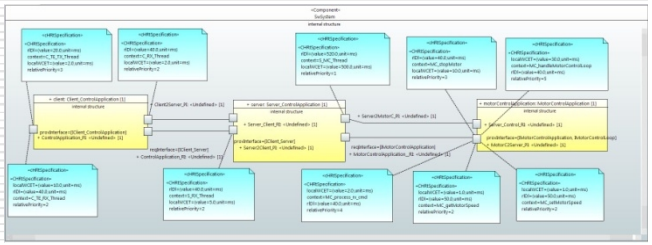
Requirements



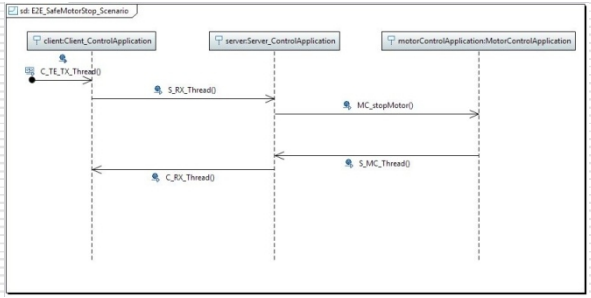
System Model



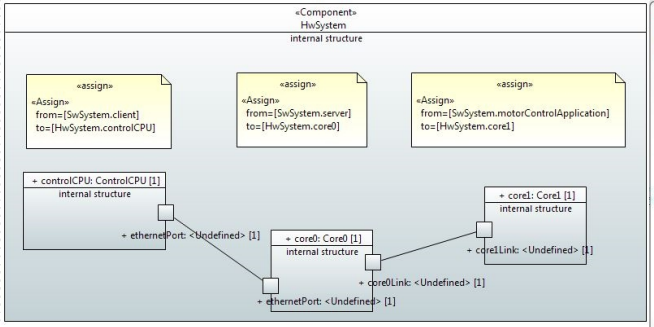
SW Model - static



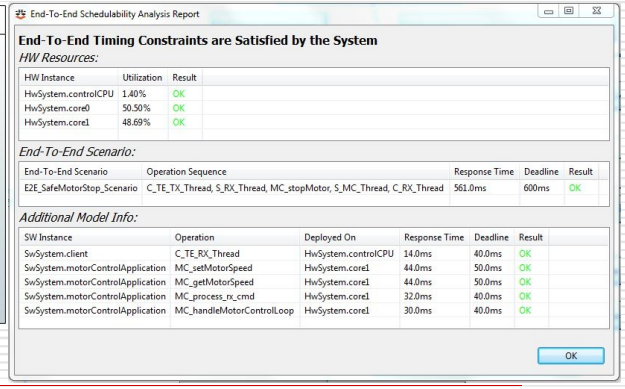
SW Model - dynamic



Deployment



Analysis View

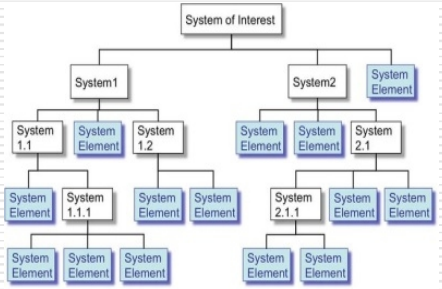


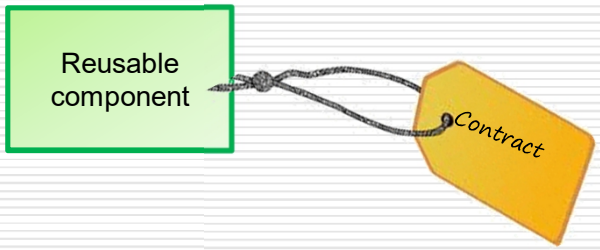
Software View - CHESS component model

- Component
  - Reusable functional unit, decorated with extra-functional constraints
  - Platform Independent
- Container and Connector
  - Implementation of the extra-functional properties of components
  - Factorized implementation
  - Platform Specific (PSM View)

Using Contracts in CHESS

- Use Contracts for System Engineering
  - for lower levels of decomposition to be consistent with the higher ones
  - to formalize conditions for element verification and integration
  - for reuse of abstractions of available components
- Contract-based design benefits
  - compositional reasoning
  - co-engineering
  - separation of concerns
  - systematic virtual integration and verification
  - protection of intellectual property





Contracts-based approach

- Contracts composed of Assumptions and Guarantees
  - Assumptions are properties expected to be satisfied by the environment
  - Guarantee is a statement that holds as long as the environment satisfies the assumption

Contract

Assumption

Guarantee

The conceptual models

System Functional Architecture

System Logical Architecture

System Physical Architecture

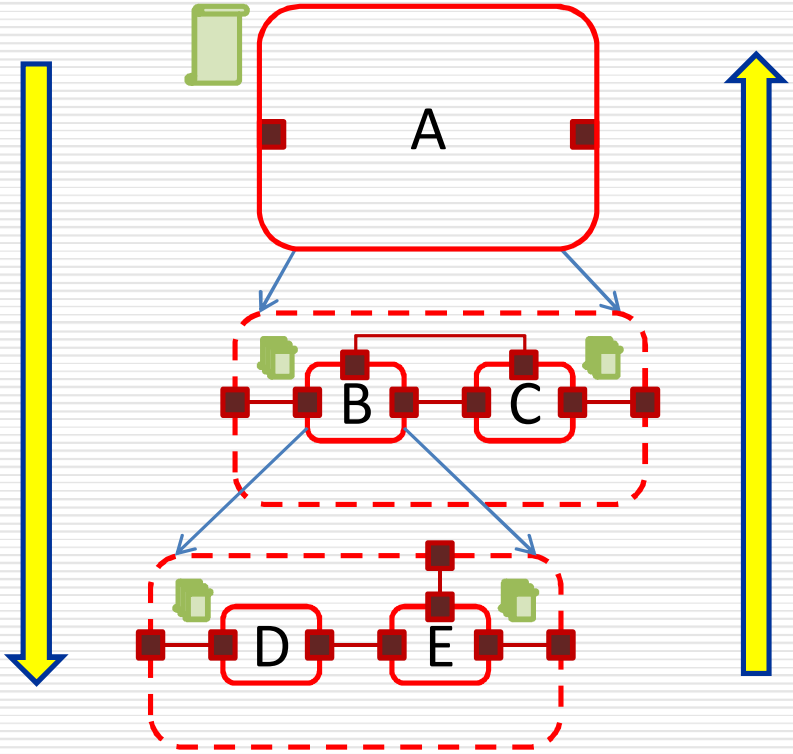
Software Architecture

Step-wise (vertical) refinement process with formal verification of contract refinement
within each conceptual model
and trace relation between corresponding
entities at different conceptual levels

Step-wise refinement

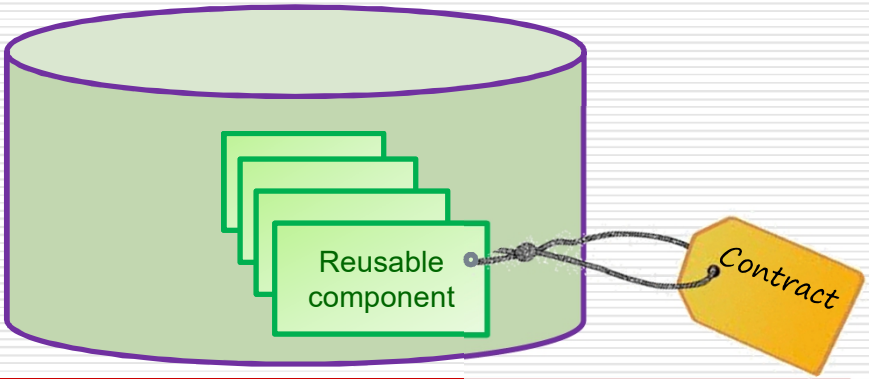
Formal verification

If the refinement steps are proven correct, then any implementation of the leaf components that satisfies the component contracts can be used to implement the system



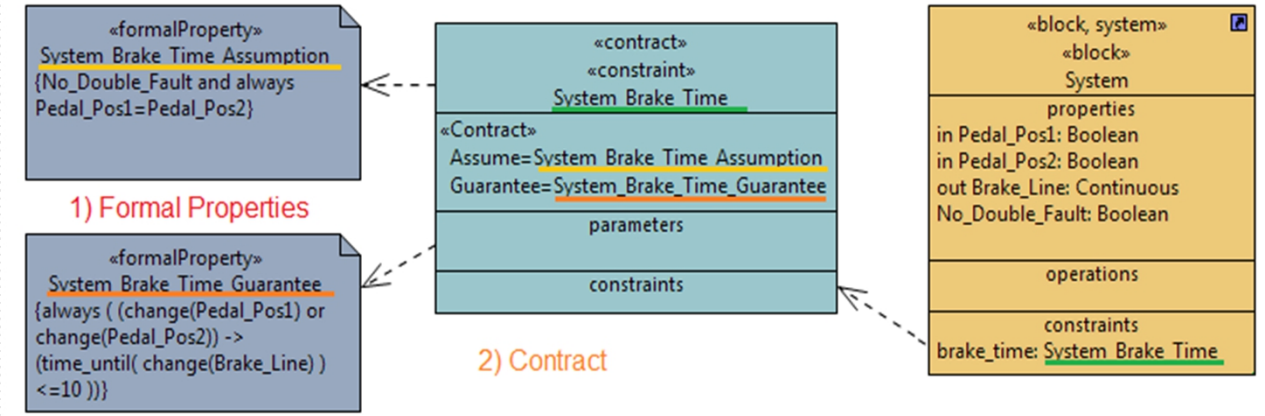
... it is a top-down process

... but is also enables bottom-up exploitation of libraries of reusable certified components

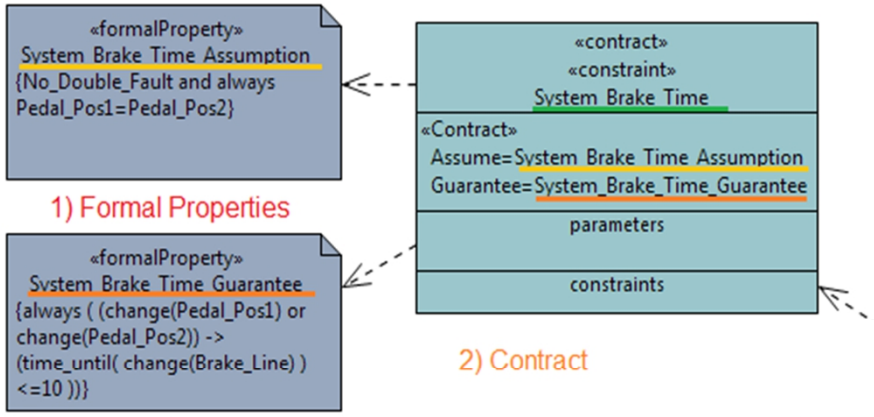


Contract-based View

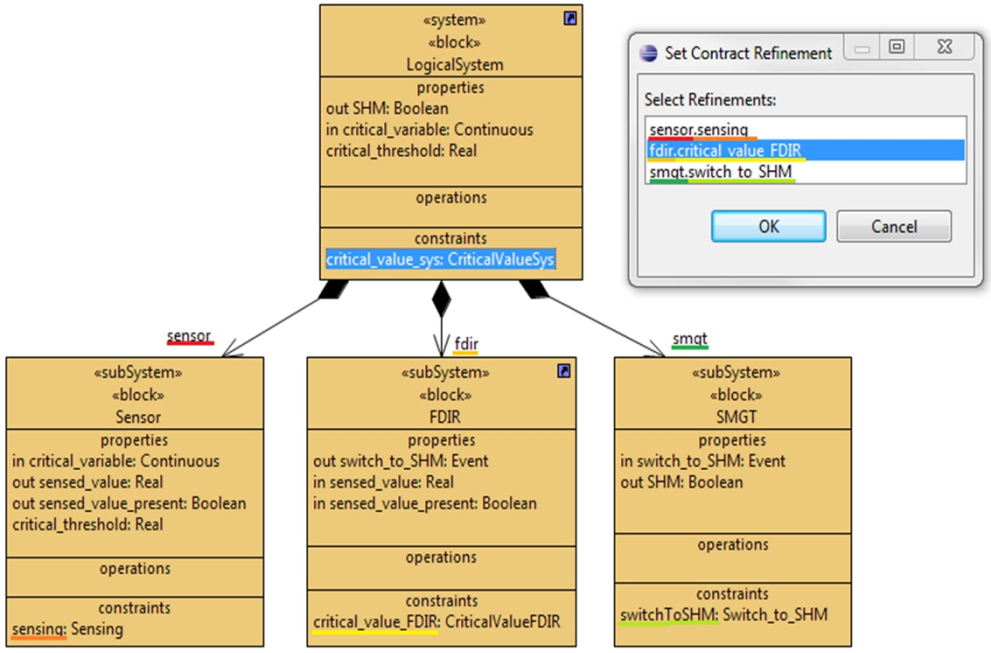
- Requirements formalization
  - Usage of LTL
- Collect formalized requirements as contracts
  - Assumption and guarantee properties
- Assign contracts to system/software/HW platform components
- Enable contract-based analysis



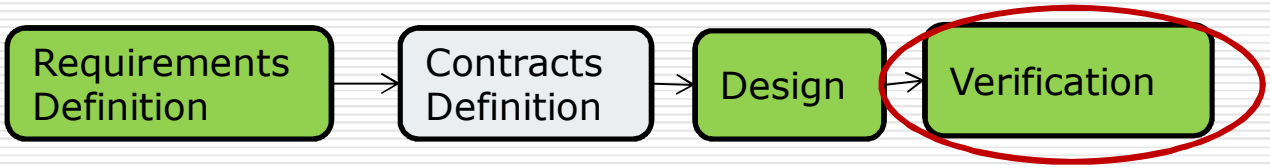
Contracts modelling support



Contracts modelled as a special kind of constraint, owning assumptions and guarantees constraints



Contract-based analysis support



- Seamless integration with OCRA, nuXmv and XSAP tools from FBK
  - Verification of contracts refinements
  - Verification of contracts composition
  - FTA from contracts specification
  - Verification of contracts against component behavior specification

Thank you for your attention

QUESTIONS?