Lessons Learned in a Journey toward Correct-by-Construction Model-Based Development

Silvia Mazzini
Intecs S.p.A.
Pisa, Italy
silvia.mazzini@intecs.it

On the origin of Correct-by-Construction

- Quoting E. Dijkstra: *software productivity is closely related to rigor in design, a sound and predictable method to eliminate software bugs at an early stage*
- … Not first write a program and then test it, but rather provide a mathematical proof of correctness before committing the corresponding algorithm to code
- Essentially… it is about detecting and removing as early as possible any errors that may occur in the development

Pursuing CbyC according to the MDE paradigm: the ideal process

- CbyC principles and the MDE paradigm lead to an ideal process of automated software production
  - from a **formal specification** of the solution
  - through a sequence of (automated) **model transformations**
  - to a **correct implementation**
- Correctness of the involved transformations proven by some algebra

Late Peter Amey's «six principles» for CbyC

- The goals of CbyC can be attained by the application of the following six principles:
  - Specialization
    - Use formal/precise tools/notations for any product of the development cycle
  - Automated step-wise validation
    - use tool support to validate the product of each step
  - Divide-and-conquer
    - break the development down in smaller steps to defeat error persistence
  - Dryness
    - say things only once, to avoid contradictions and repetitions
  - Beware of complexity
    - design software that is easy to validate
  - Rigor and discipline
    - do the hard things first, including thorough requirement analysis and the development of early prototypes

Still a code-centric approach!

Definition and experimentation of an MDE way to CbyC

Lessons learned across different projects

- We want to share our experience in this quest over a decade of work across 4 large R&D projects
  - **ASSERT** (EU FP6 program): the first attempt to realize a model-driven methodology for embedded space software system development with a dedicated **component model**, explicitly focused on CbyC.
  - **CHESS** (ARTEMIS): the realization of a cross-domain model-based, **component-oriented approach** to the development of embedded real time software systems across domains
  - **SafeCer** (ARTEMIS): model-driven technology for composable and reusable safety certification, experimenting with contract-based development processes
  - **CONCERTO** (ARTEMIS): extend the CbyC model-based methodology of CHESS to multi-core processors with the same level of guarantees and also widened the coverage of industrial application domain needs

CbyC in Model-Driven Engineering: the ASSERT Experience

- *Primary goal:* prevention of semantic errors creeping in the user model
  - In particular, in the specification of real-time attributes and in the derivation of real-time properties for software components and of their assembly
- *Main result*: a dedicated **component model** to enable **architectural, rule-based composition**, for the compositional assembly of locally asserted real-time attributes into system-level properties

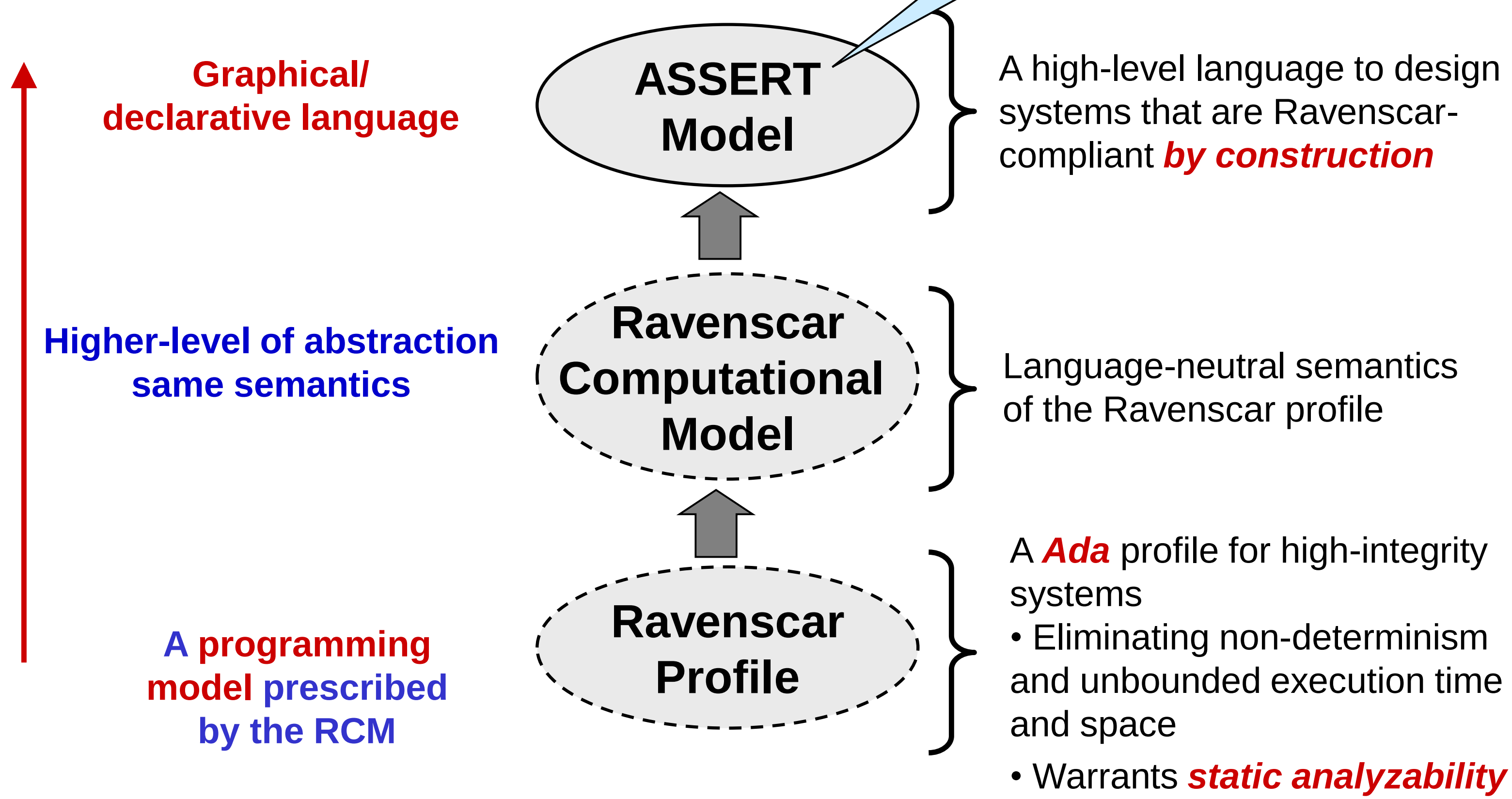
The ASSERT Methodology

- Two levels of abstraction
  - **Platform-independent model** (PIM) as the user space
    - Model of components
    - Expression of functional and timing properties for component interfaces
  - **Platform-specific models** (PSM), generated by automated model transformations, as an analysis and implementation space that captures the concurrency and real-time semantics expressed in the PIM model
    - Feasibility analysis
    - Automatic code generation
- Models conform with a given meta-model
  - For syntax, semantics and constraints on entities, attributes and relations
  - The meta-model makes all the dimensions of interest fit together consistently

CbyC Principles in ASSERT

- Model-based analysis
  - Guaranteed static analyzability
  - Consistent implementation

one single meta-model guarantees the consistency



The CHESS approach

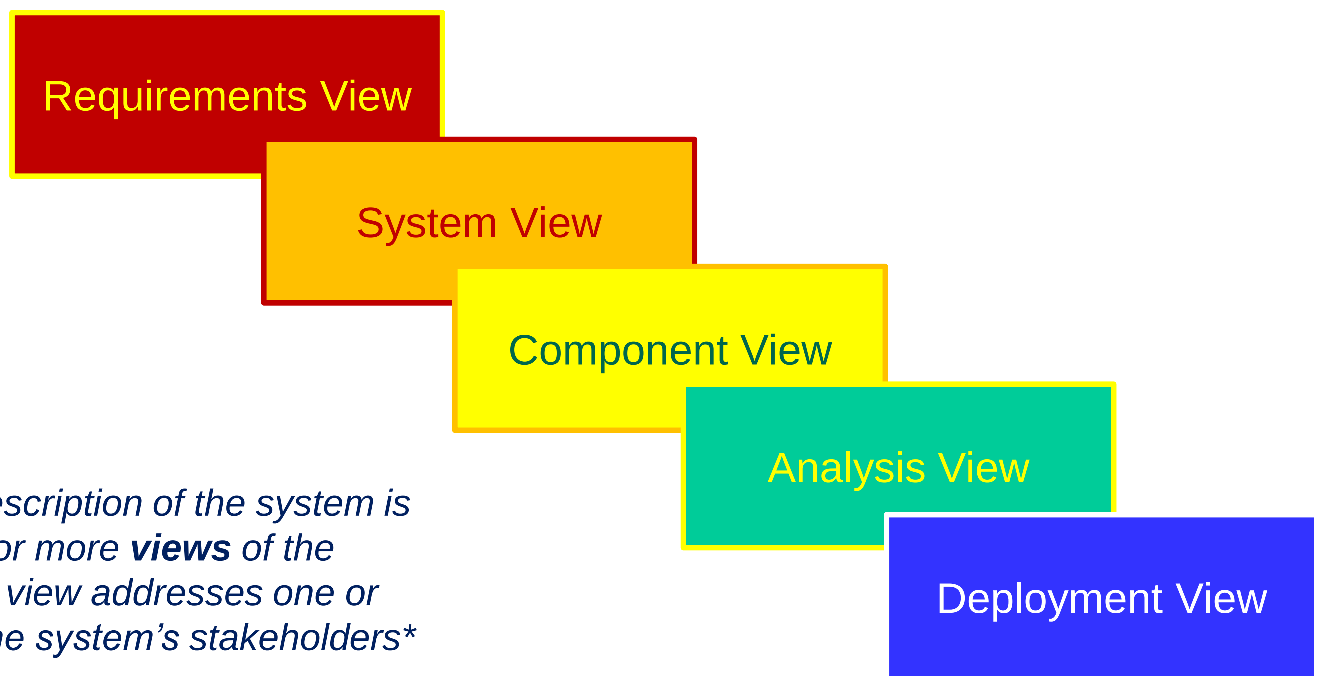
An **open source** solution for the development of critical real-time and embedded systems

- Model-driven engineering
  - Models as the central development artifacts
  - Tool assisted automated development
- Component based development
  - Specialized to capture the non-functional properties of components
    - Real Time
    - Dependability
- **Separation of concerns**

Separation of concerns [1/2]
A multi-view approach

- To sharply separate distinct aspects of design
- Each development actor focuses exclusively on their area of [development] expertise
- Use specialized formalisms, tools and verification techniques for distinct concerns
- Especially functional and non-functional concerns

- Achieved by the use of design views in the user space



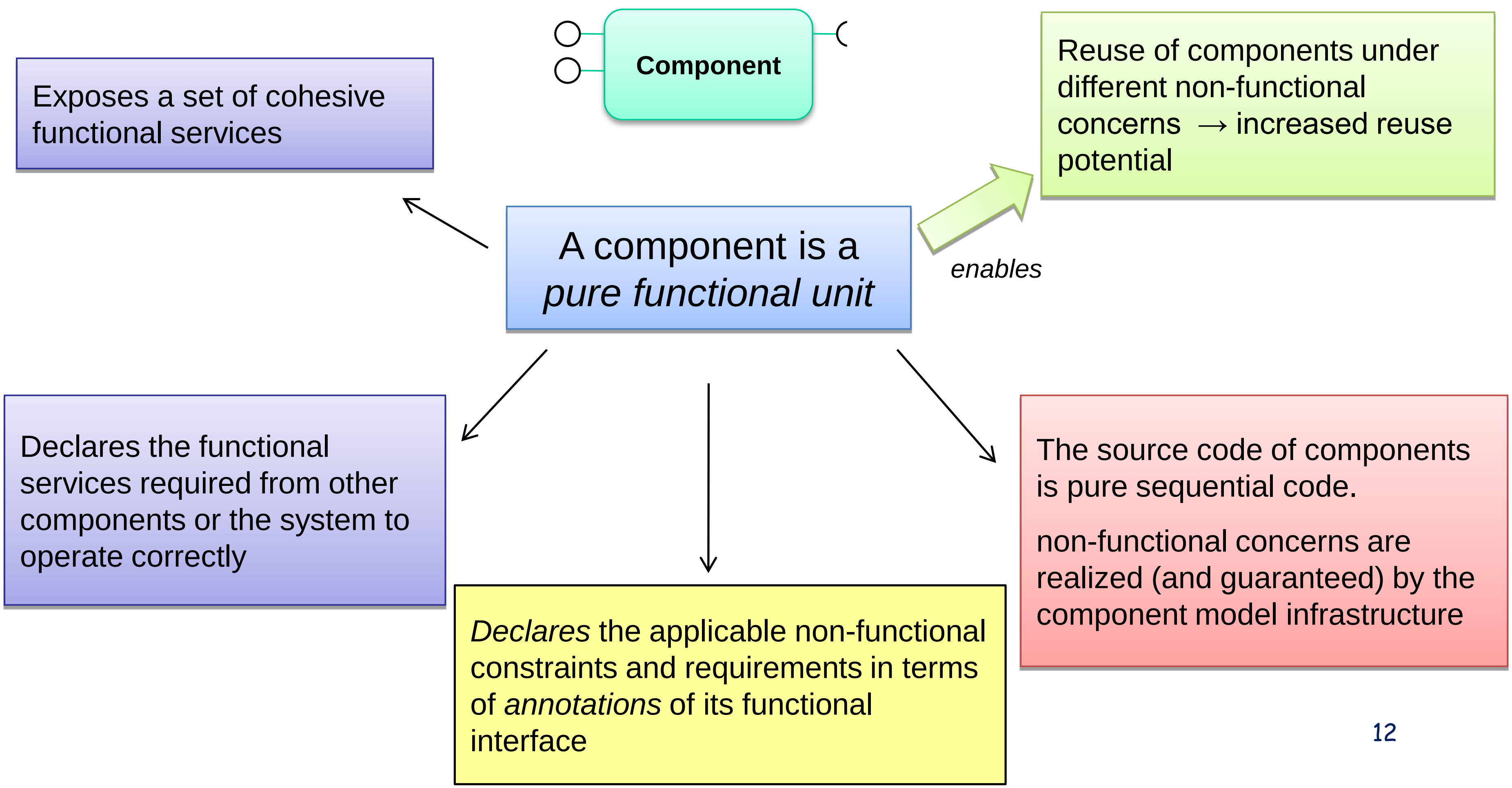
*The architectural description of the system is organized into one or more **views** of the system where each view addresses one or more concerns of the system's stakeholders**

* [ISO/IEC/IEEE42010:2011 Systems and software engineering — Architecture description]

Separation of concerns [2/2]
The CHESS Component Model

Separation of concerns is also achieved by the use of

- A component model that separates components, containers, and connectors and uses them to address distinct concerns

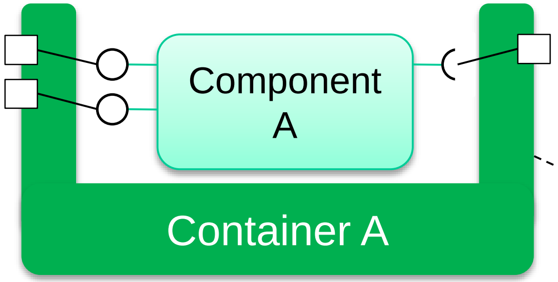


The CHESS Component Model

- **Component**
  - Reusable functional unit, decorated with non-functional constraints
  - Platform Independent
- **Container** and **Connector**
  - Implementation of the non-functional properties of components
  - Factorized implementation
  - Platform Specific
- Composability
  - properties of individual components are preserved on component composition
- Compositionality
  - properties of the system as a whole can be derived as a function of the properties of components

CHESS Container and Connector

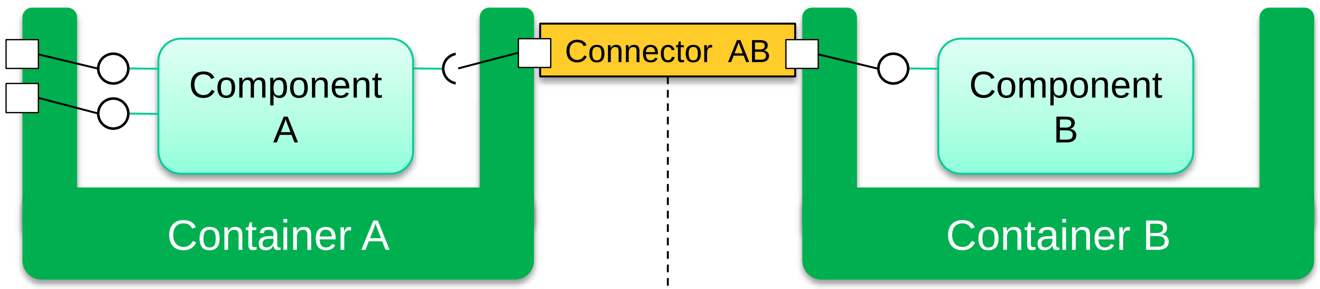
Container



-Wrapper responsible for the declared non-functional attributes

- Provides the component with a mediated connection with the execution platform and the system in general

Connector



- Addresses interaction concerns
- Decouples the component from the other end-point(s) of a communication
- Realizes connection properties (best-effort, at most once, exactly once)
- E.g. procedure/function call, remote message passing, I/O file operation, …

- From the interaction perspective components are black boxes that only expose their interfaces

Component-based modeling with guarantees

CHESS Component Model with properties of

- **Compositionality**
  - the properties of the system as a whole can be determined as a function of the properties of the constituting components and the execution environment
- **Composability**
  - individual components' properties are preserved on component composition, deployment on target and execution
- **Computational model**
  - To relate architectural entities and their properties to analysis equations
  - To statically analyze the system
- **Programming model**
  - To enforce analysis assumptions
  - To express the semantics assumed by the analysis
- **Execution platform**
  - To actively warrant the properties asserted by analysis

Correctness by construction

Non-functional properties can be:

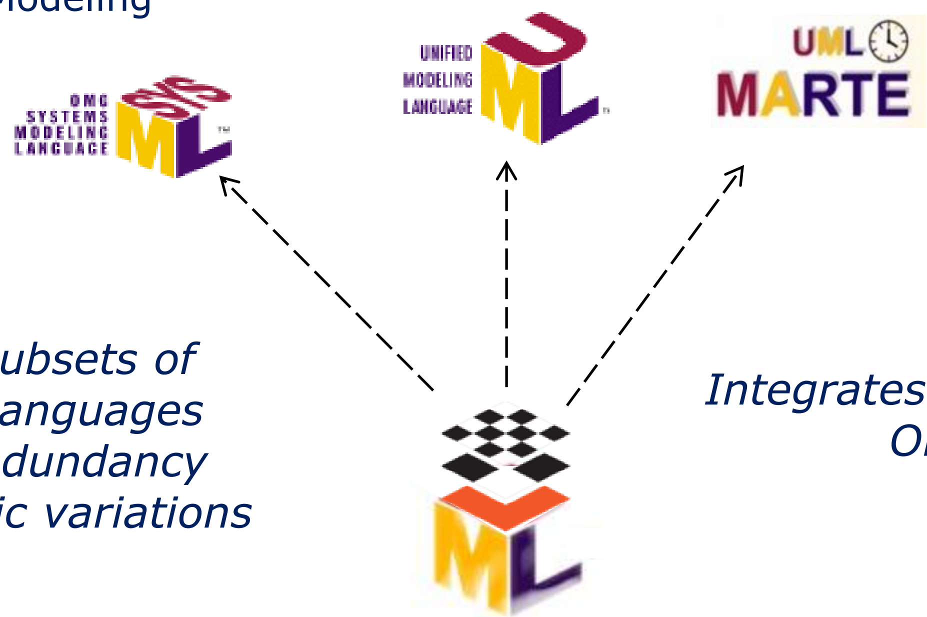
- Specified on the model
- Asserted by static analysis
- Guaranteed in the implementation
- Preserved at run-time

The CHESS Modelling Language

Standard profile for System and Requirements Modeling

Standard Unified Modeling Language

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

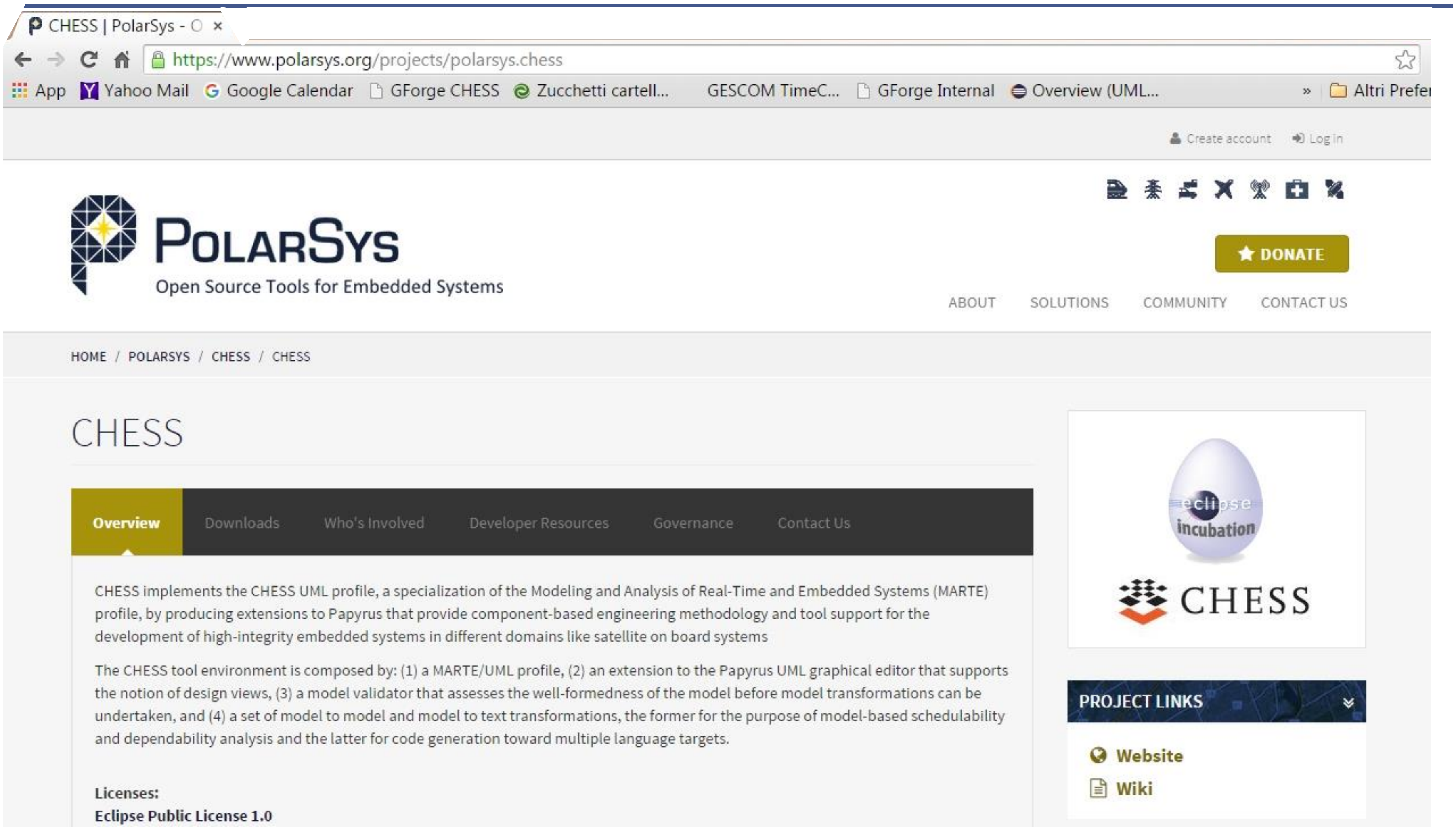


Imports subsets of standard languages
✓ avoid redundancy
✓ fix semantic variations

Integrates and extends standard OMG languages

Introduces a new Dependability Profile

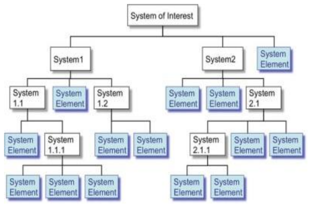
CHESS under the Eclipse PolarSys Initiative

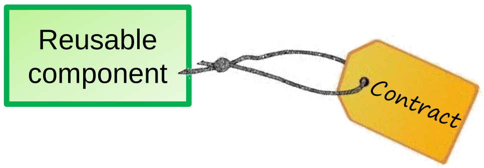


Investment in PolarSys of important players from the industrial and academic world: a reliable community committed in the effort to create and maintain open methods and tools for critical systems, guaranteeing interoperability based on open standards

SafeCer:
Using Contracts

- Use Contracts
  - 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

Contract-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

The conceptual models

Functional Architecture

Logical Architecture

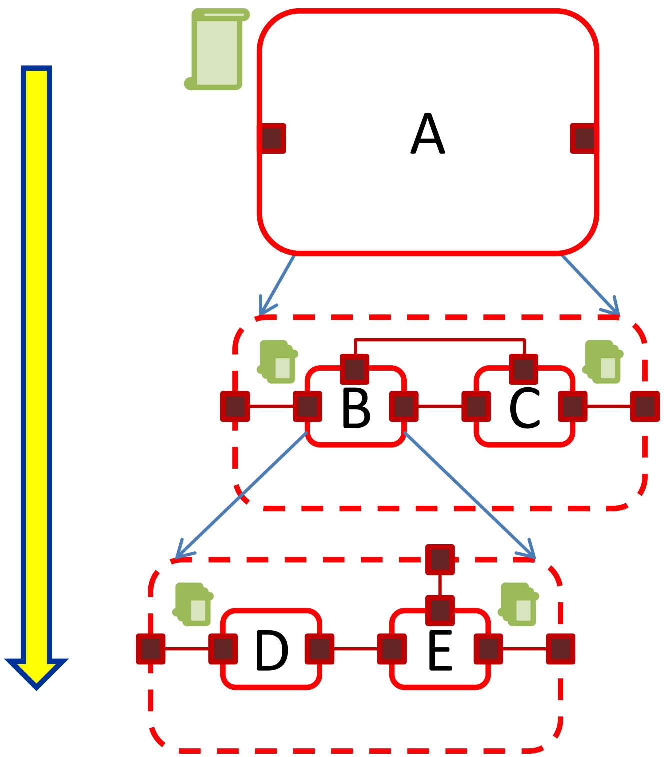
Physical Architecture

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

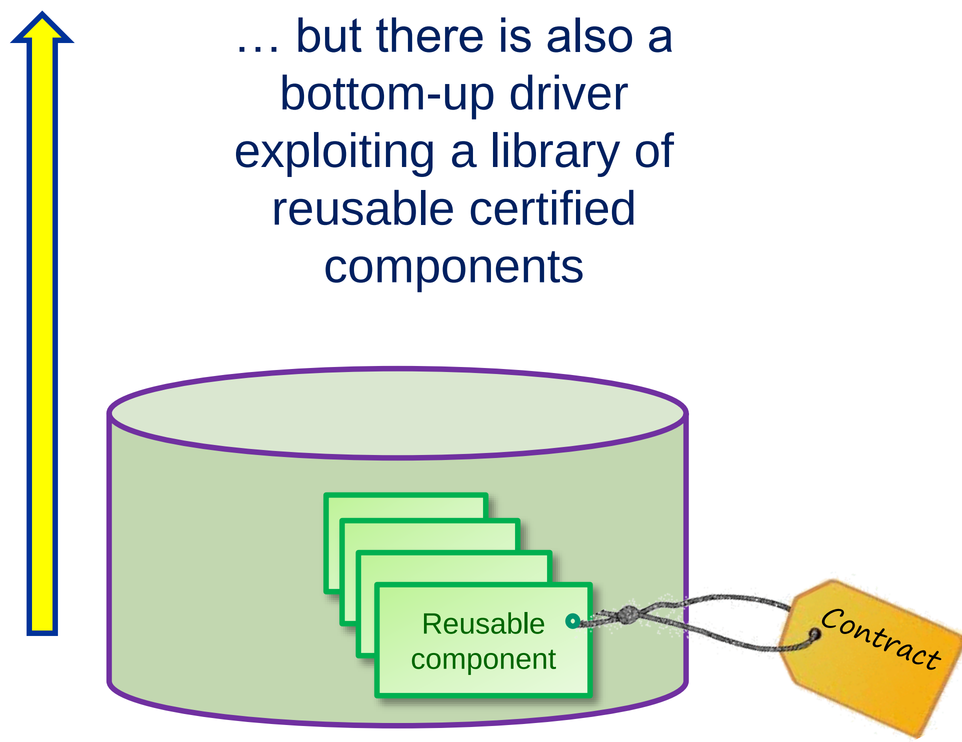
Formal verification

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 …

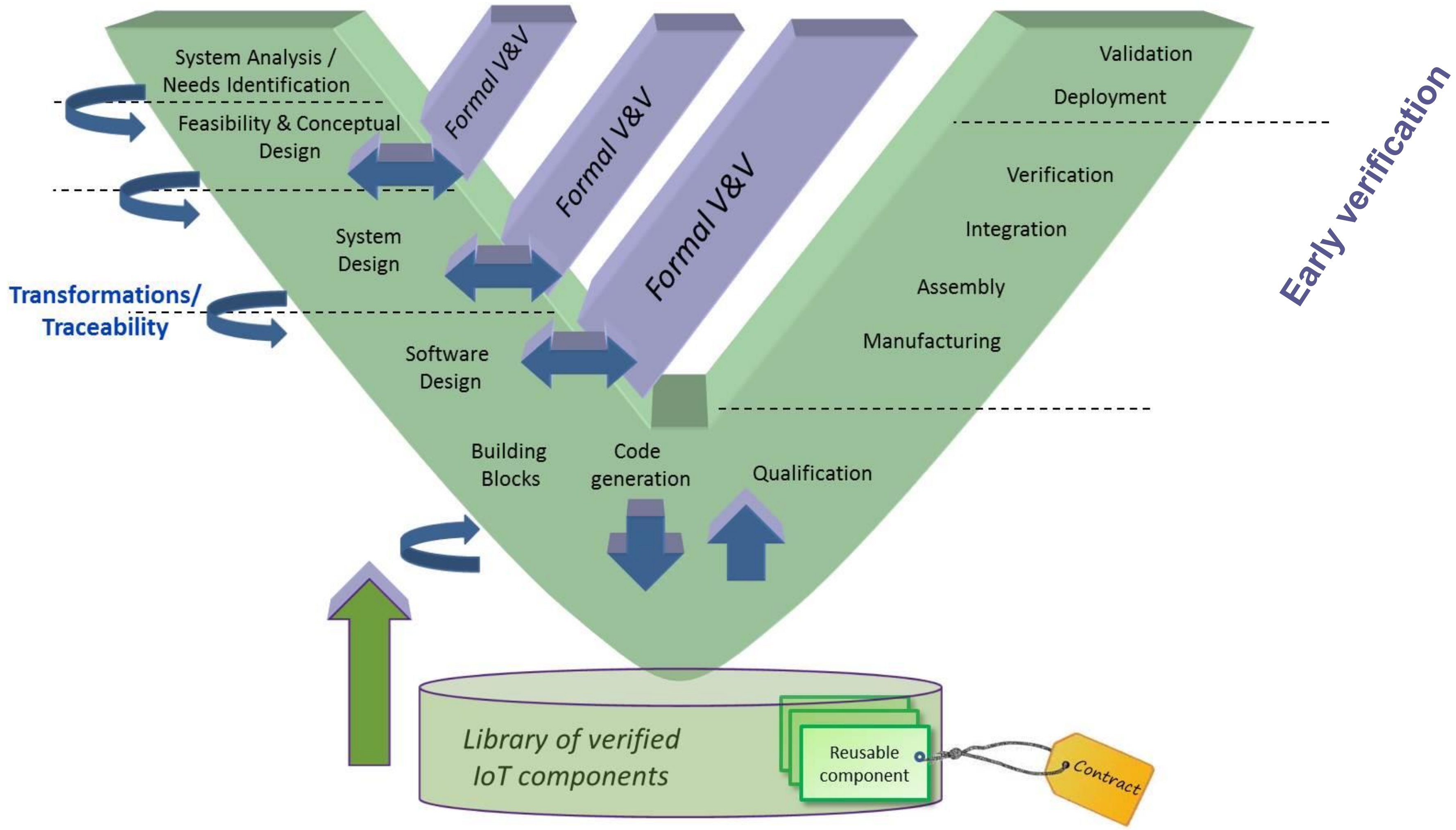
… but there is also a bottom-up driver exploiting a library of reusable certified components



The CHESS Tool-chain
Integration with the OCRA tool

- The OCRA tool by Fondazione Bruno Kessler supports checking of refinement of contracts specified in a linear-time temporal logic
- Integration of OCRA in the CHESS tool-chain provides a framework that assists the user across the entire development process
  - Description of the system and its hierarchical decomposition
  - Definition of requirements associated to components
  - Formalization of requirements as contracts
  - Stepwise refinement process with explicit verification of contract refinements and component implementations
- However…
  - identifying a feasible system decomposition and contract refinement requires engineering experience and human intervention
  - Designing traces between corresponding component in different conceptual levels is responsibility of the user (no automated formal verification)

The CHESS enhanced V-model development process



Further Challenges: CONCERTO

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

ARTEMIS JU Call 2012: ongoing

To extend the CHESS project achievements

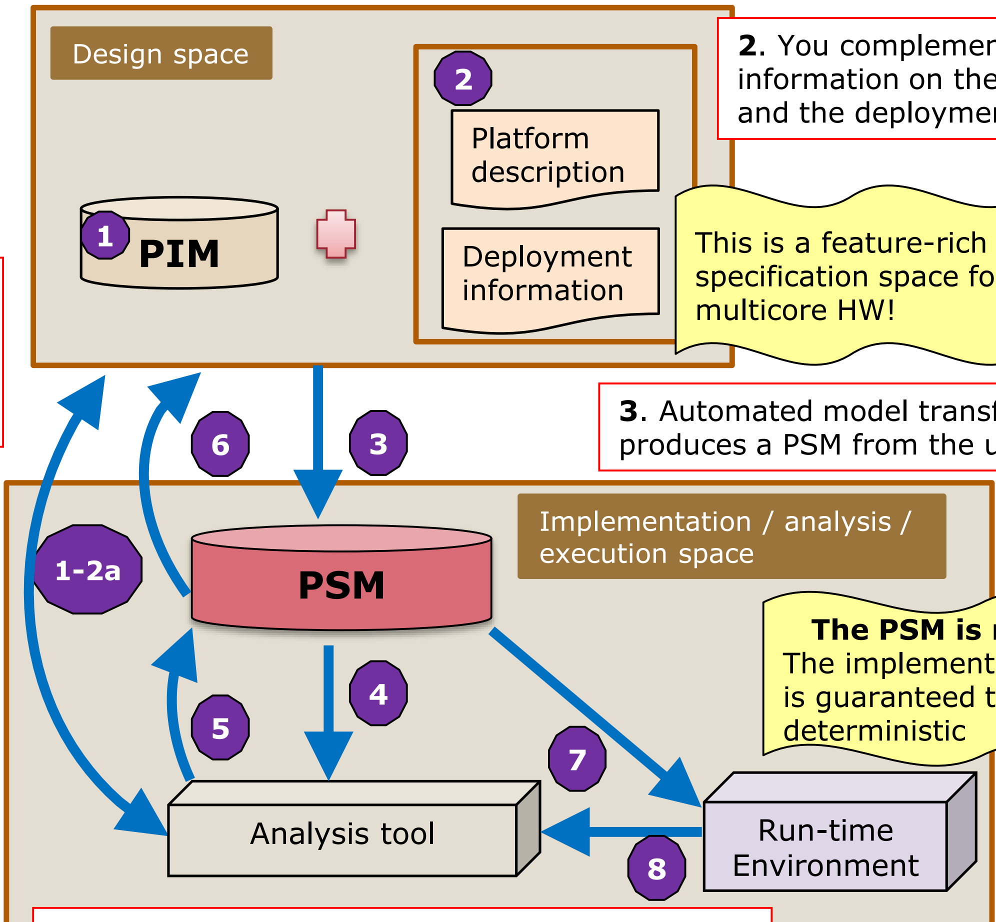
- ❑ Extensions to multicore platforms
- ❑ Support partitioning
- ❑ Address mixed-criticality issues
- ❑ Manage run-time monitoring and back propagation of run-time data
- ❑ Model and clearly represent component hierarchies
- ❑ Support AUTOSAR
- ❑ Wider coverage of industrial domains
  - automotive, medical, offshore platforms, avionics, telecom, space

The CONCERTO process

1. You construct a PIM to represent your solution to your problem, independent of any specific implementation.

2. You complement the PIM with information on the target platform and the deployment plan.

1-2a. Dependability/safety analysis is performed at PIM system/SW and platform specification level, with back propagation of analysis results.



This is a feature-rich specification space for multicore HW!

3. Automated model transformation produces a PSM from the user PIM

4. Real-time relevant analysis is performed on the PSM

The PSM is read-only! The implementation product is guaranteed to be deterministic

5-6. The analysis results are back propagated to the PSM and to the PIM

The user iterates the 1-6 cycle as many times as needed

7-8. The implementation is deployed to the HW, with run-time verification support if needed. Run-time monitoring is activated to grab live data for run-time monitoring analyses and back propagation of results, as in **5-6.**

Addressing Multi-core Processors Platforms

- Multi-core target platforms introduce an extremely high level of complexity for real-time analysis
  - At the state-of-the art predictability analysis in case of multi-core processors yields penalizing results due to the adoption of necessary conservative countermeasures
    - Scheduling so that only one core at a time is active
    - Use strictly partitioned scheduling

The CHESS/CONCERTO solution is based on:

- Advanced feasibility analysis
  - Possibility to perform schedulability and end-to-end response time analysis on different (multi-core) deployments for comparison
  - Back propagation of analysis results to the user model (PSM and then PIM)
- Round-trip analysis methodology
  - Back propagation of run-time data from application execution in its run-time environment for comparison with analysis results and model assumptions
  - Use run-time monitoring to detect/ manage violations

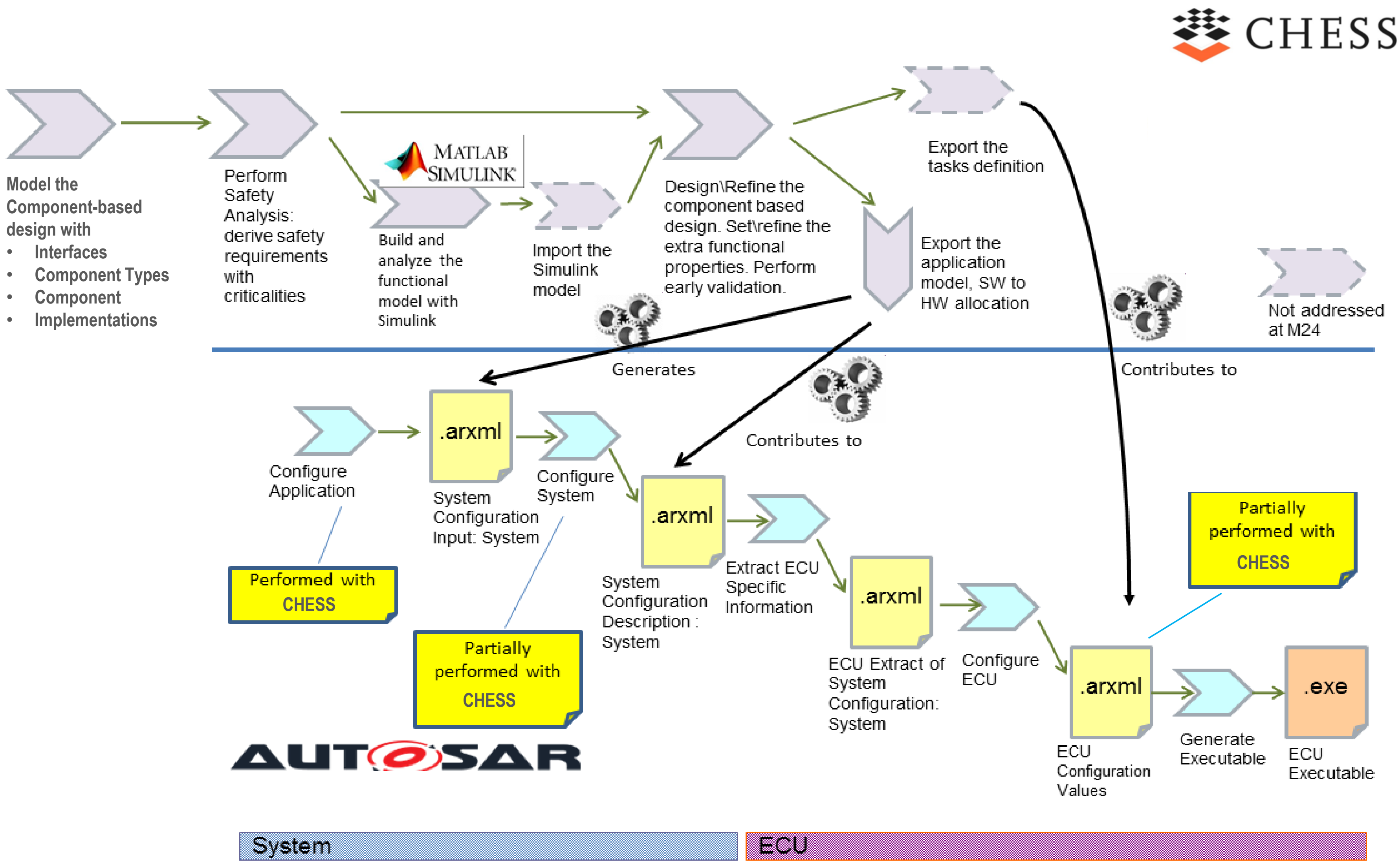
It is a «correct-by-correction» approach: design failures may occur, but they are detected early enough and managed accordingly

CHESS extensions for the Automotive domain

- AUTOSAR (AUTomotive Open System ARchitecture)
  - Open and standardized software architecture for automotive, jointly developed by automobile manufacturers, suppliers and tool developers
- Integrating CONCERTO with AUTOSAR
  - Sound model transformations were developed from CONCERTO to AUTOSAR ✓
    - CONCERTO component model entities are mapped to semantically equivalent AUTOSAR ones
  - The vice-versa was not feasible (AUTOSAR->CONCERTO) ✗
    - AUTOSAR component model has a richer set of constructs
    - AUTOSAR allows higher degree of modeling freedom
    - … but this freedom comes at the cost, for instance, of run-time semantics of operations specified by the user in the AUTOSAR model not being guaranteed, by construction, to be statically analyzable for feasibility ⚠

CONCERTO and AUTOSAR can complement each other,
but no complete bi-directional integration is currently possible

Automotive: CHESS integration with AUTOSAR



Conclusions

- The ASSERT and the CHESS development processes and modelling steps had a strong connotation of CbyC
- SafeCer proposes a rigorous stepwise contract refinement approach for system and software design.
  - decompositions and refinements may have a more *tentative* nature than assertive, requiring backtracks, as in correctness-by-correction
- Lessons learned in CONCERTO
  - the wider the coverage of non-contiguous industrial domains, the more difficult the application of CbyC
  - not enough design and implementation prescriptions are known to enforce correctness, to guide the development in a top-down fashion
  - the satisfaction of some (modelling and semantic) constraints had to be deferred to later stages, enabled by ad-hoc transformations toward specialized analyses (e.g., for dependability, conformance to given restrictions, feasibility in the time domain)
  - substantial deflection of CbyC into correctness-by-correction

Thank you for

your attention!

Questions?