



Model-Based Design and Simulation for Runtime Assurance

Xenofon Koutsoukos and Joseph Porter



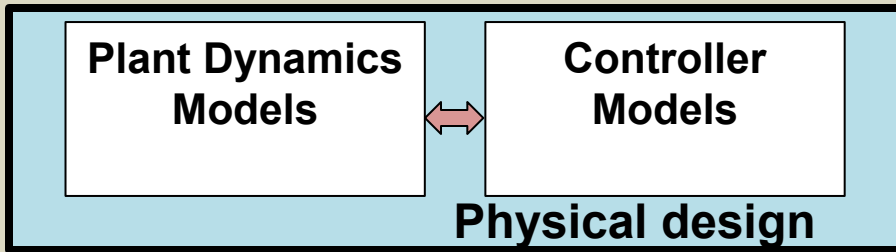
Outline



- Research objectives
- Modeling frameworks and languages
- Software design
- Behavioral analysis
- Dependability analysis
- Model-based testing
- Applications and evaluation

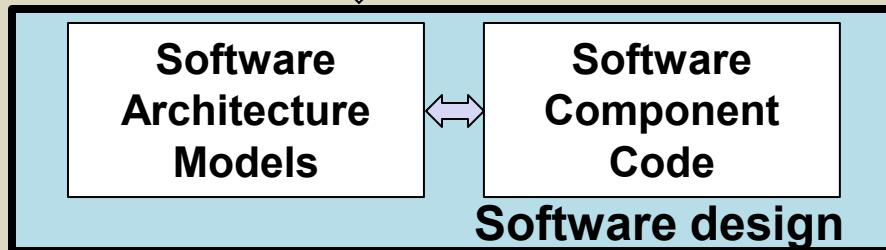


Simplified Model-Based Design Flow



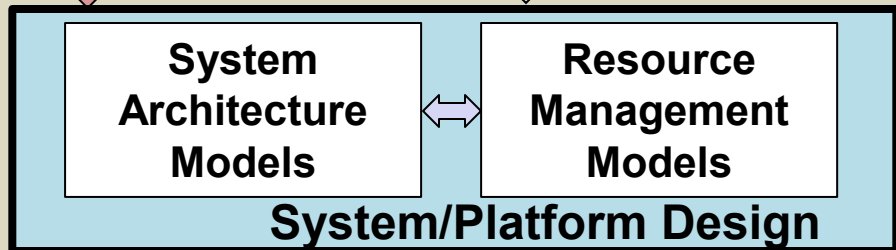
Controller dynamics is developed without considering implementation uncertainties (e.g. word length, clock accuracy) optimizing performance.

Assumption: Effects of digital implementation can be neglected

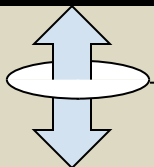
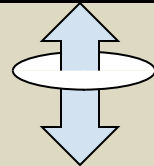
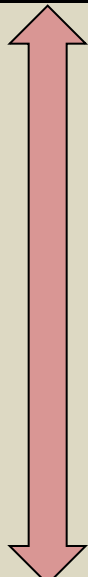


Software architecture models are developed without explicitly considering systems platform characteristics, even though key behavioral properties depend on it.

Assumption: Effects of platform properties can be neglected

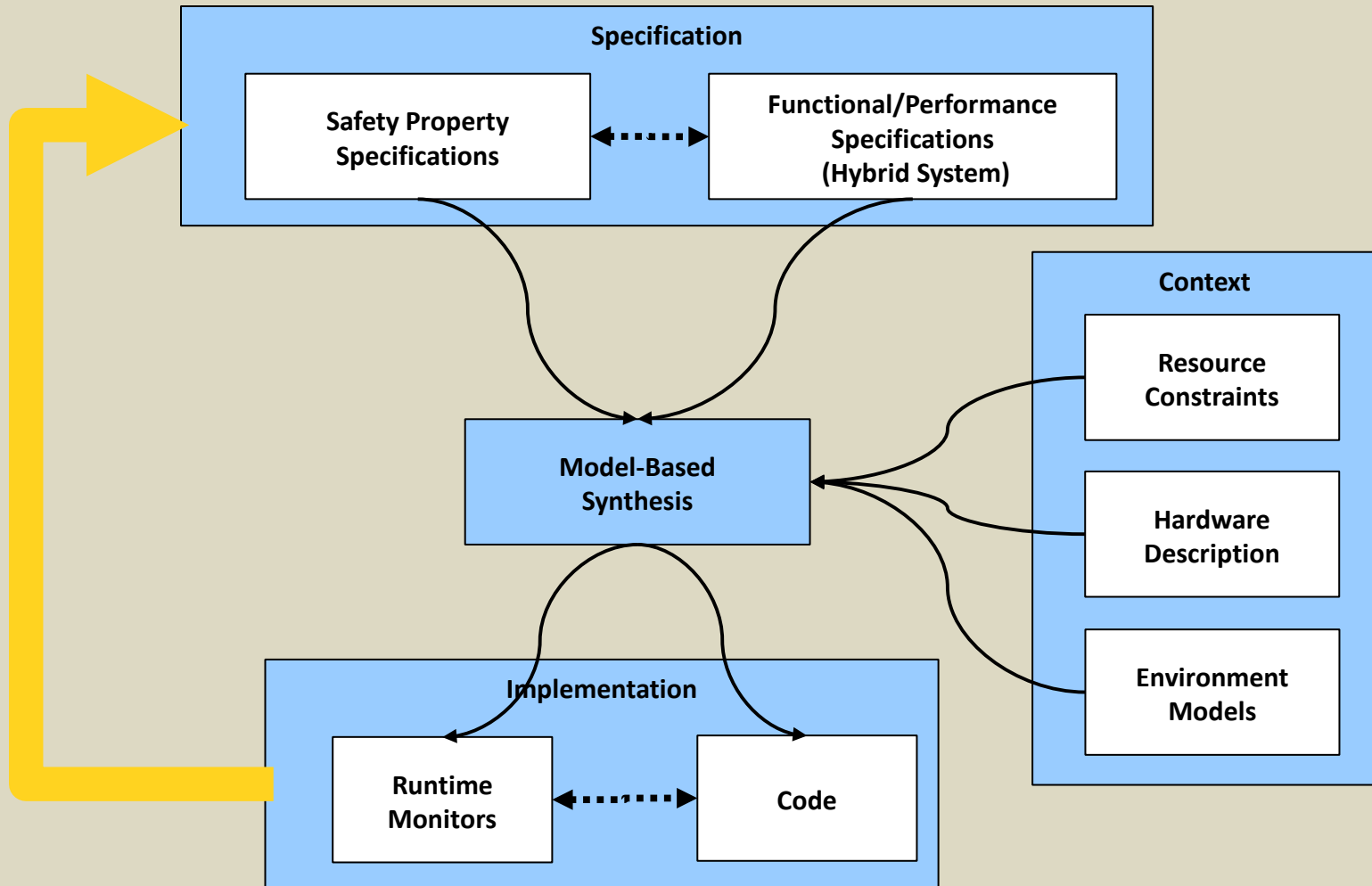


System-level architecture defines implementation platform configuration. Scheduling, network uncertainties, etc. are introduced, which may require re-verification of key properties on all levels.





Model-Based Design for RTA





Modeling Frameworks and Languages



- Customizable abstractions for domain specific modeling languages
 - Model-integrated computing
- Heterogeneous models of computation
 - Ptolemy
- Flexible design flow
 - Model transformations
 - Incremental design

G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-integrated development of embedded software. Proc. of the IEEE, Jan 2003.

J. Eker, J. Janneck, E. A. Lee, et al. Taming heterogeneity – the Ptolemy approach. Proc. of the IEEE, Jan 2003.

G. Karsai, A. Agrawal, F. Shi, and J. Sprinkle. On the use of graph transformation in the formal specification of model interpreters. J. of Univ. Computer Science, 2003.

J. Porter. Compositional and Incremental Modeling and Analysis for High-Confidence Distributed Embedded Control Systems. PhD thesis, Vanderbilt University, 2011.



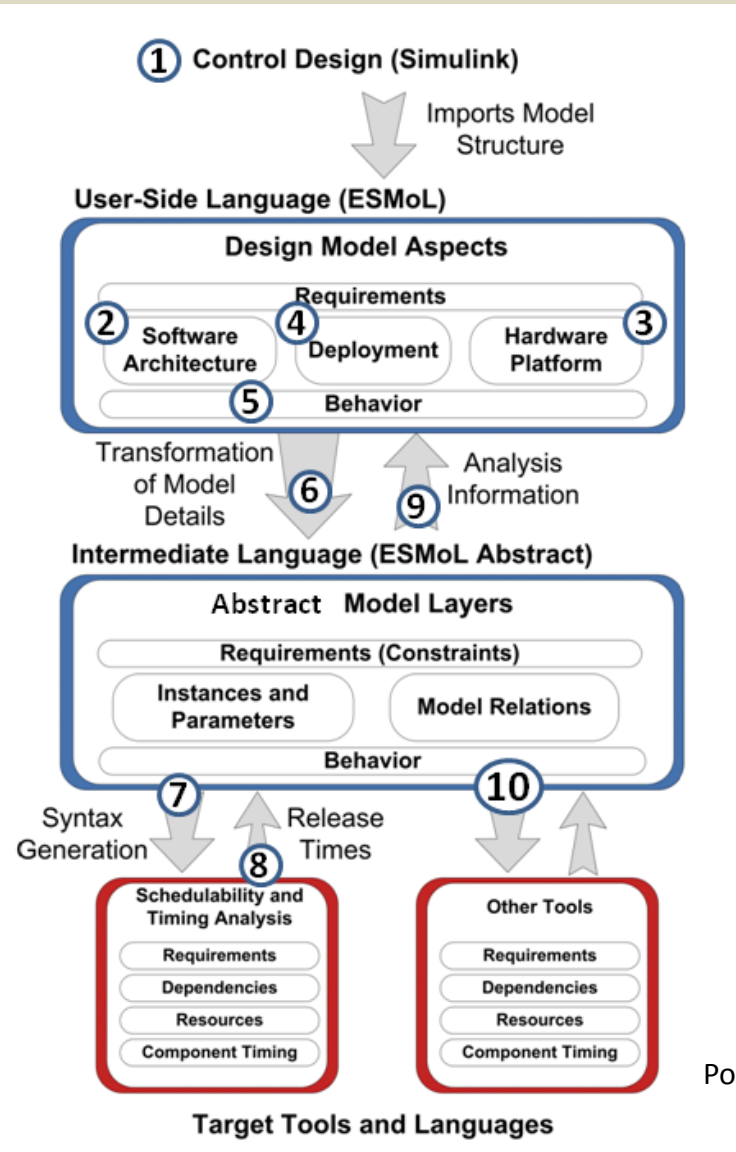
Modeling and Design Tools



- Commercial tools
 - Matlab/Simulink/Statflow, SCADE, UML/SysML
- ACME Studio
- Architecture Analysis and Design Language (AADL)
- Behavior-Interaction-Priority (BIP)
- Embedded Systems Modeling Language (ESMoL)



ESMoL Design Flow



Single iteration of the design and assessment flow:

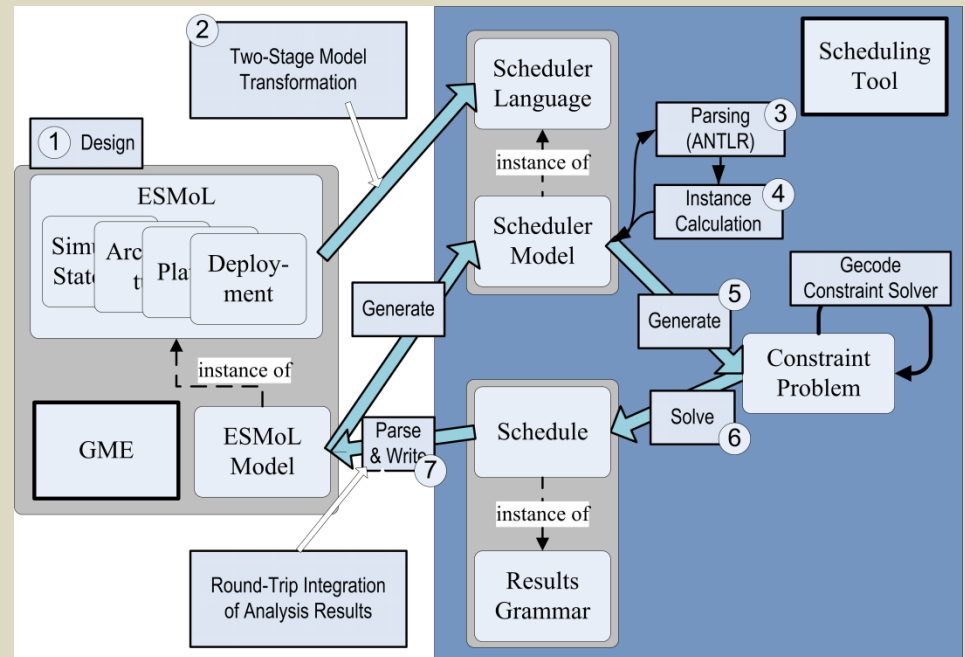
1. Import a Simulink control design into an ESMoL models.
2. Specify software component functions and interfaces, and instantiate the components into a logical software dataflow model.
3. Specify the hardware topology for a time-triggered distributed processing network.
4. Define deployment of the logical dataflow to the hardware, and
5. give timing parameters .
6. Transform the ESMoL model (via the Stage 1 transformation) into a model in the ESMoL Abstract language, resolving all implied relationships and structural model inferences.
7. Transform ESMoL_Abtract models into analysis models.
8. Import results from the analysis back into the ESMoL Abstract
9. and ESMoL models.
10. Create platform-specific simulations and generate deployable code using the Stage 2 transformation.



Software Design



- Automated software design based on models
 - Generation of software code for guards
 - Platform-specific glue code generation
- Model-based integration of schedulability analysis
 - AADL and ESMoL
 - Incremental methods
- Performance analysis





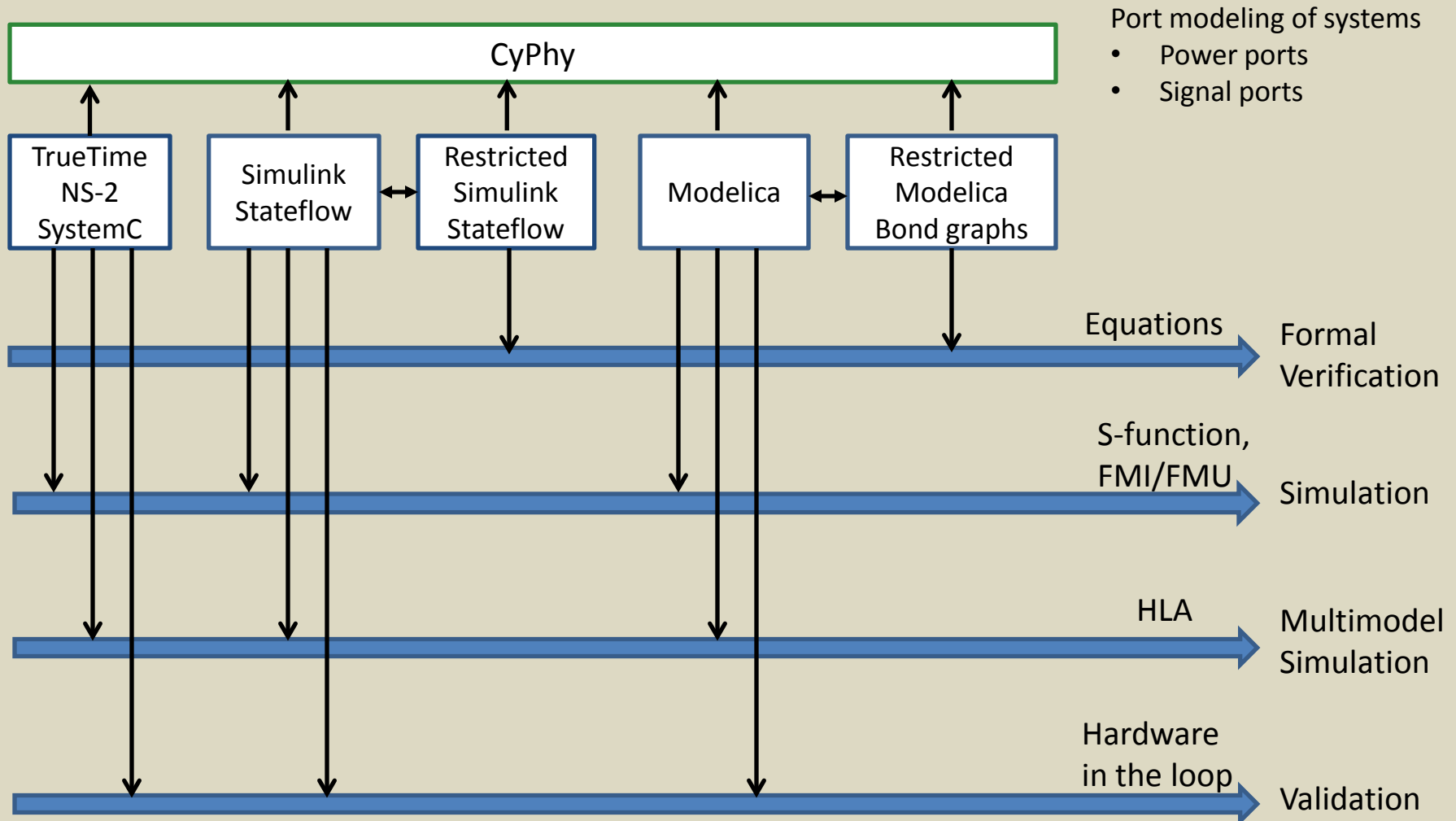
Simulation-based Analysis



- Research objective: Capture effects of hardware platform and software implementation on the control design
- Hardware/software co-simulation
 - Register Transfer Level (RTL)
 - Transaction Level Modeling (TLM)
- System-level performance simulation
 - Automated synthesis of Simulink models with TrueTime blocks
- Frameworks for simulation integration
 - High Level Architecture (HLA)
 - Functional Mockup Interface (FMI)



Integration Framework

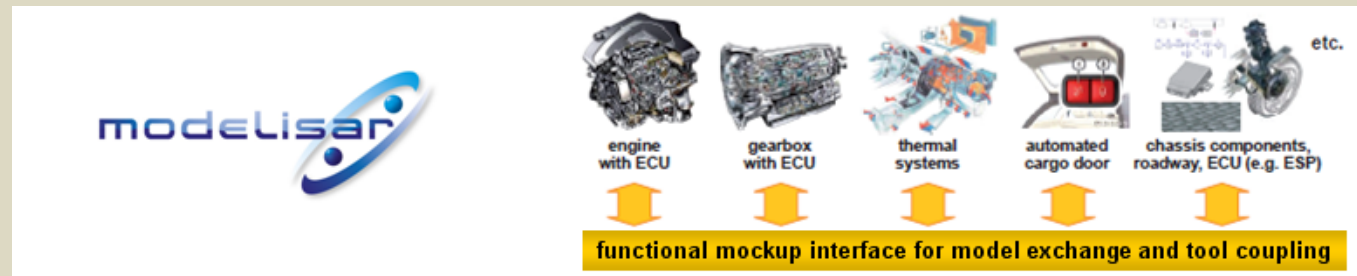




Functional Mockup Interface



- The Functional Mockup Interface definition is one result of the ITEA2 project MODELISAR (in support of AUTOSAR)
- The intention is that dynamic system models of different software systems can be used together for software/model/hardware-in-the-loop simulation and for embedded systems
- The Functional Mockup Interface defines (a) C-header files to interact with the equations of a model or to perform co-simulations with other simulators and (b) XML schema files to inquire information about model and interface variables
- FMI Support in Tools: Matlab/Simulink, Modelica, ControlBuild, LabView, SimulationX,...





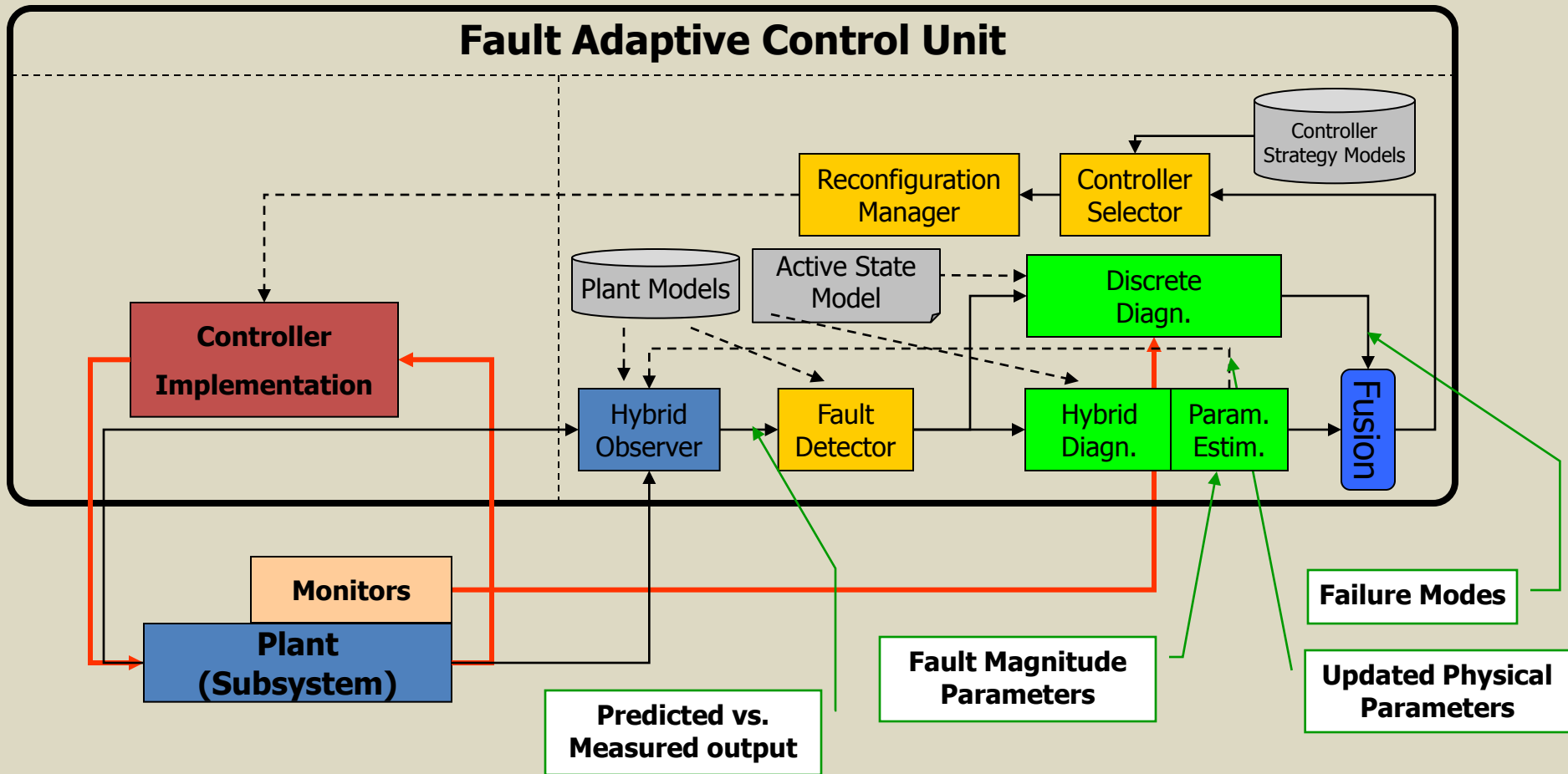
Formal Verification



- Hybrid system verification
- Model-based software verification and design
 - Transparency and simplicity
- Composition platforms
 - Time triggered architecture
 - ARINC 653
- Behavior-Interaction-Priority (BIP)
 - Verify state invariants such as deadlocks
- Formal verification of code generated from Simulink models

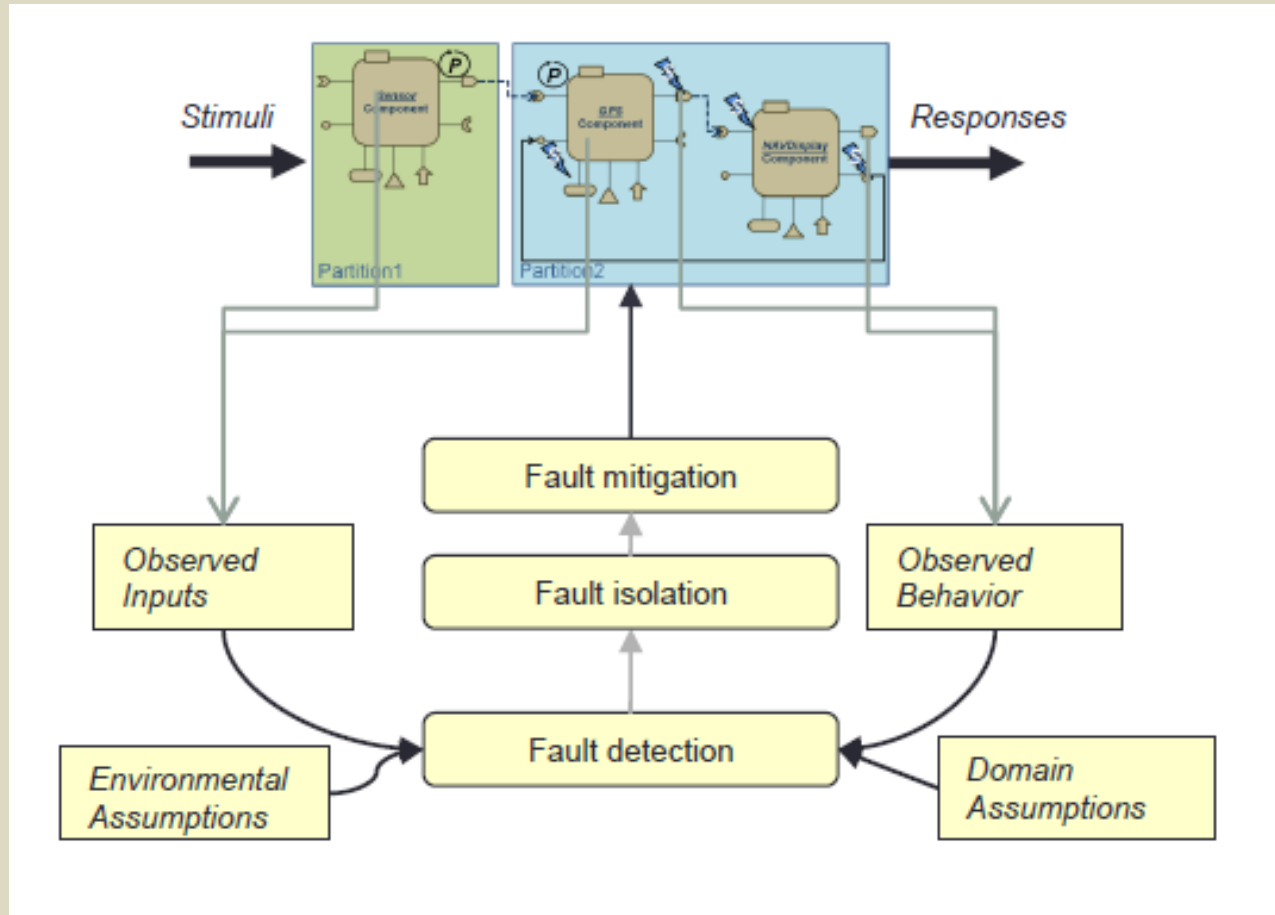


Fault Adaptive Control Technology





Software Health Management



G. Karsai, G. Biswas, S. Abdelwahed, N. Mahadevan, and E. Manders. Model-based software tools for integrated vehicle health management. Space Mission Challenges for Information Technology, 2006. SMC-IT 2006.

A. Dubey, G. Karsai, and N. Mahadevan. Model-based software health management for real-time systems. In IEEE Aerospace Conference, March 2011.



Model-Based Testing



- Design model evaluation is complicated
 - different scales (component, subsystem, system)
 - in different contexts
 - Does nominal behavior meet requirements?
 - Does the design properly mitigate fault requirements?
 - Simulation vs. verification
 - for different refinements
 - Initial idealized controller design
 - Platform effects
- EDA industry uses test bench models to automate evaluation of designs.



Test Benches



- Model-Based Test Automation
 - For RTA similar tests will need to be run on safety controllers, performance controllers, and guarded controllers
 - Test bench abstraction manages the complexity of the design flow and model refinements.
- Test Vector Generation
- Test Space Reduction



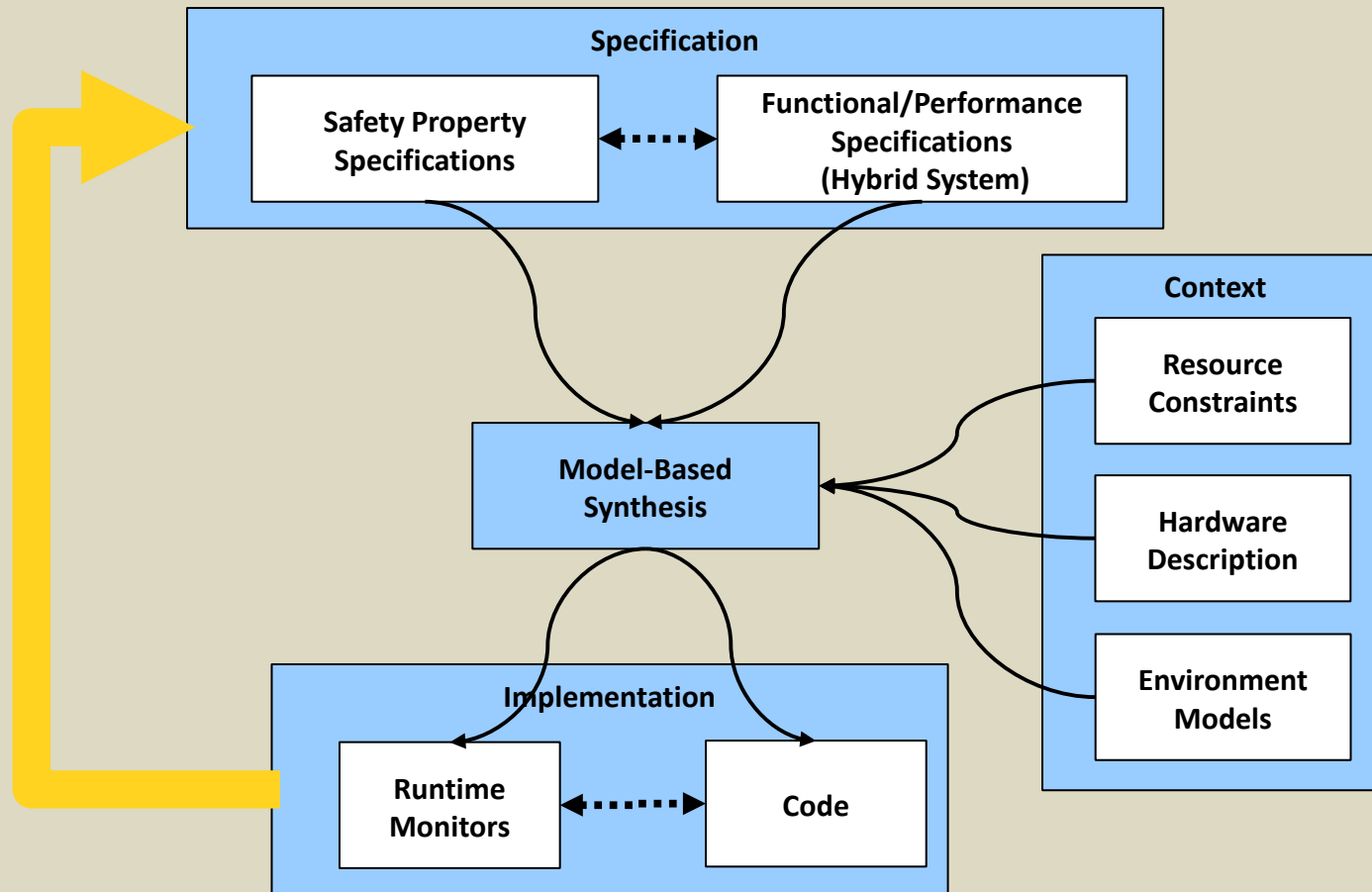
Test Interpretation



- Validating runtime data against simulation
- Parameter space exploration
- Exhaustive property evaluation
- Trajectory-based performance assessment
- Debugging implementation models
- Defining regression models



Challenges in RTA



Interactions between design-time hybrid dynamical models and runtime implementations models are not adequately addressed



Modeling Languages and Tools



- Support for specification of functions and behavior of the elements required by RTA
- Support for updating designs based on runtime properties with reasonable time and cost
- Support for incremental design techniques



Software Design



- Generation of assured software code for runtime components
- Modeling of the effects of platform and controller timing variances on the bounds represented by the guard
- Capturing platform and deployment models which can be used to assess the validity of design changes on end-to-end analyses (e.g. latency of tasks in a processing chain).



Simulation-based Analysis



- Capture effects of hardware platform and software implementation on the control design
 - New model-based methods that are capable of refining control models with implementation details
 - Efficient and scalable simulation techniques of heterogeneous systems



Formal Verification



- Verifiable transformation of design models into verification languages including the translation of properties that need to be verified
- Design-time formal verification techniques for the implementation of the safe controllers
- Formal verification must address the runtime properties of the controlled plants and control platforms
- Automatic control software synthesis that attaches a certificate of correctness to the generated code



Dependability Analysis



- Fault detection, isolation, and recovery in complex hybrid control systems
- Runtime software health management
- Design of “self-adaptive” systems that are robust to software, hardware, and physical faults



Model-based Testing



- Scaling of test coverage methods to complex RTA designs
- Automatic reduction of the size of the test space, prioritizing tests
- Development of test bench modeling concepts for definition of tests, automation of all kinds of design-time and runtime evaluation, and interpretation of results



Conclusions



- Model-based design provides a powerful framework for the development of safety-critical systems
- Analysis methods can be integrated in the design process based on formal models at different levels of abstraction
- The main challenge in RTA lies on the integration of heterogeneous models that allow system evaluation of the control design while incorporating runtime properties that depend on the software and hardware platform
- Define the system architecture of the system before its implementation
- Analyze the constraints imposed on the system by the architecture from the beginning of the design cycle until the final implementation