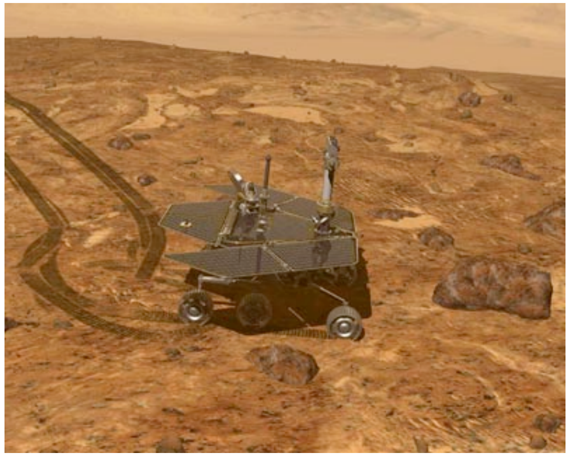
Software Verification for Space Applications

Part 2. Autonomous Systems

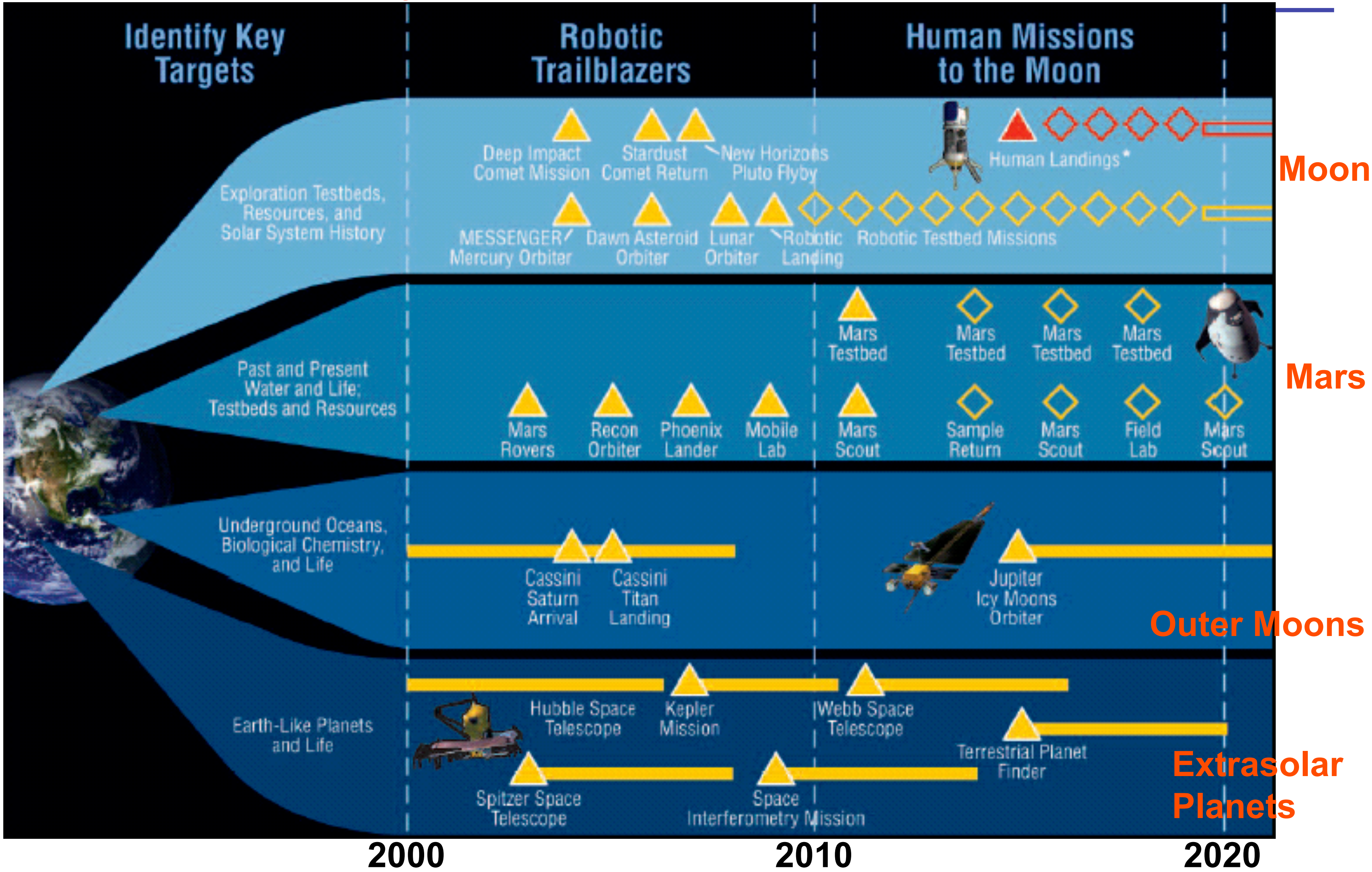


G. Brat
USRA/RIACS

Main Objectives

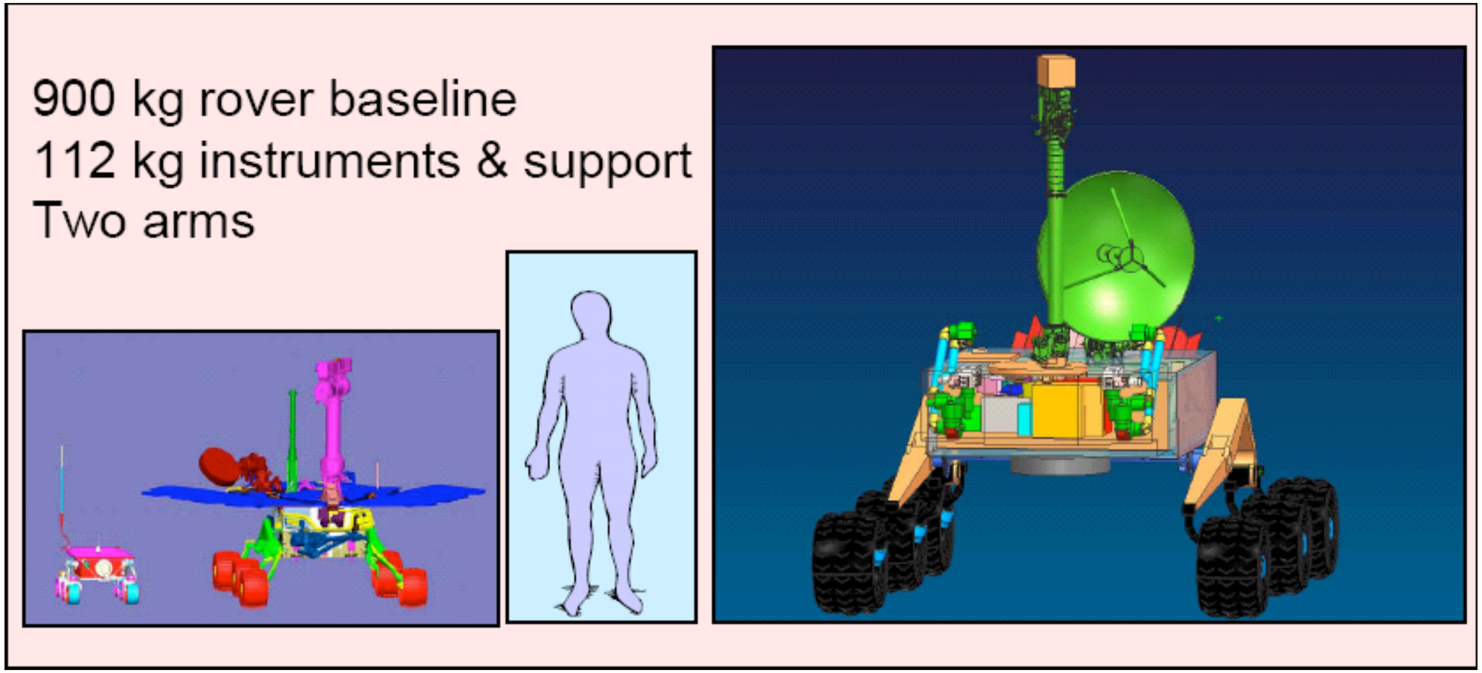
- Implement a sustained and affordable human and robotic program to explore the solar system and beyond;
- Extend human presence across the solar system, starting with a return to the Moon by the year 2020, in preparation of the exploration of Mars and other destination;
- Develop the innovative technologies, knowledge, and infrastructures, both to explore and to support decisions about the destinations for human exploration;
- Promote international and commercial participation in exploration to further U.S. scientific, security, and economic interests.

Many Robotic Missions



Mars Science Laboratory

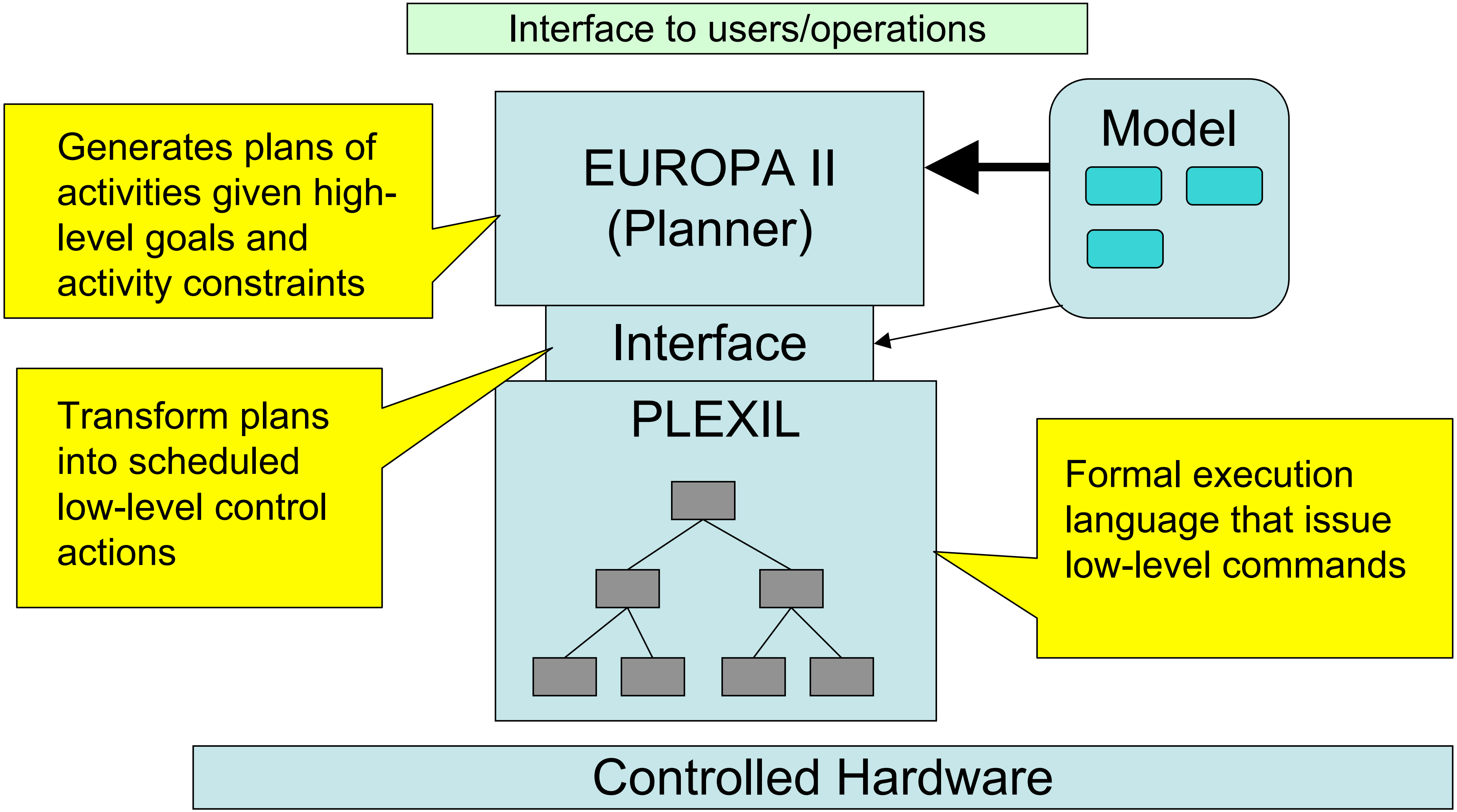
- Mission:
  - Long range traverses (< 6km)
  - Collect samples
  - Analyze samples on-board



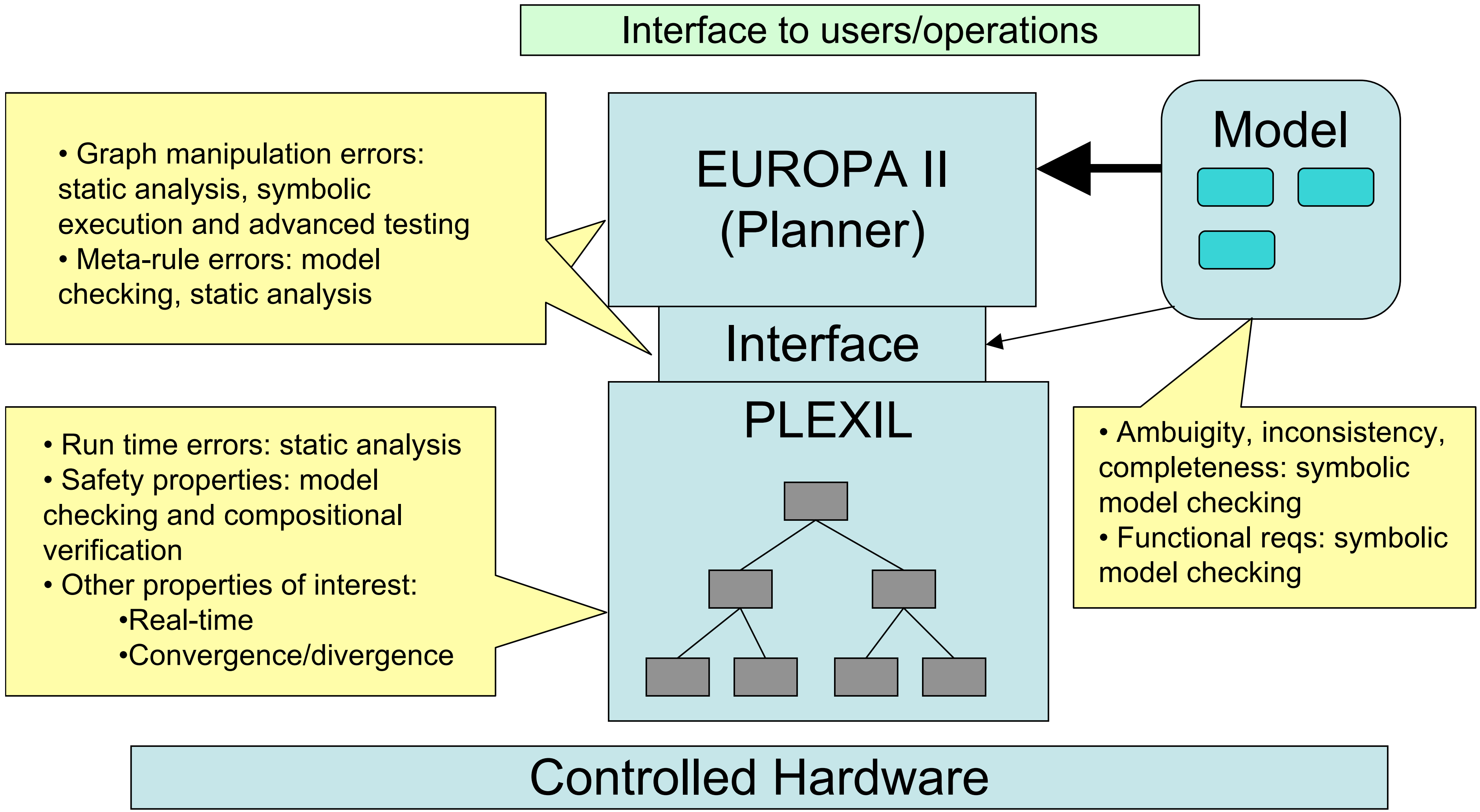
NASA Software Challenges

- Need to develop three systems for each mission:
  - Flight software
  - Ground software
  - Simulation software
- Flight software
  - Rovers will require more adaptable software to do long traverses for example
- Ground software
  - Need planning software for planning operations
  - Need autonomous execution for uploading and executing commands on ISS or on-orbit
- V&V of a different type of software systems

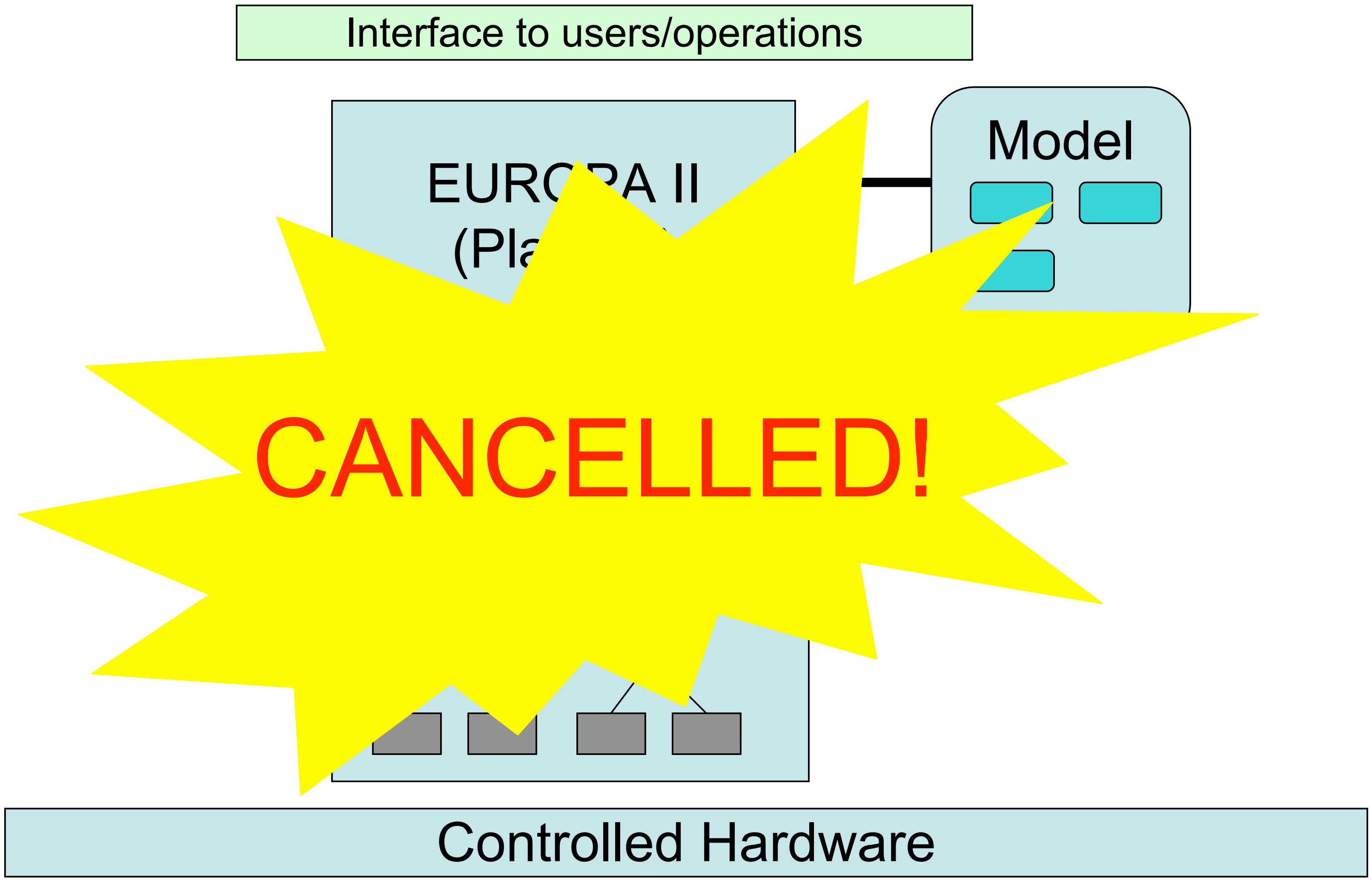
Autonomous systems: 2005



V&V Strategy

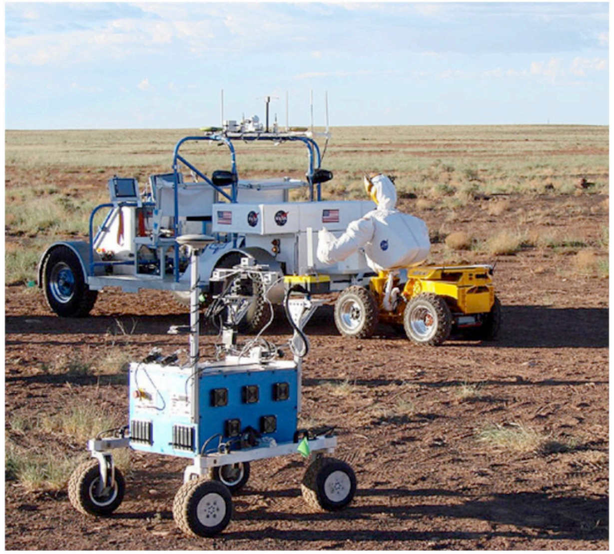

Interface to users/operations
• Graph manipulation errors: static analysis, symbolic execution and advanced testing
• Meta-rule errors: model checking, static analysis
EUROPA II (Planner)
Model
Interface
PLEXIL
• Run time errors: static analysis
• Safety properties: model checking and compositional verification
• Other properties of interest:
•Real-time
•Convergence/divergence
• Ambuigity, inconsistency, completeness: symbolic model checking
• Functional reqs: symbolic model checking
Controlled Hardware

Cancel at the end of 2005

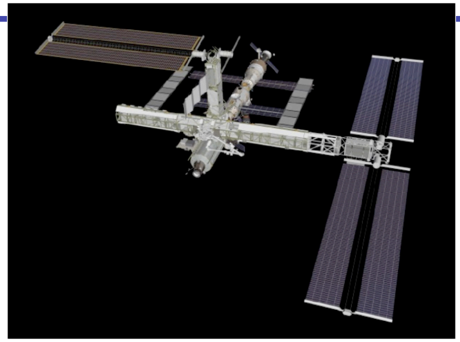


Autonomy for Operations Project: 2006

- Autonomy for Operations
  - PIs: Jeremy Frank & Ari Jonsson
  - PM: Robert Brummett
- Project goal:
  - Develop and mature needed automation software
  - capabilities for Constellation mission operations, onboard
  - control, crew assistance and robotics.
- Core capabilities
  - Human in-the-loop automation
  - Monitored execution
  - Decision support
  - Operation requirement studies
  - Simulation and testbeds
  - Application and prototypes
  - Verification



Background



- **Mission Operations**
  - Operating procedure generation
  - Space flight operations planning
  - Remote system operations (nominal and off-nominal)
  - Support of crew control (nominal and off-nominal)
- **Crewed Spacecraft Operations**
  - Spacecraft systems operations (nominal and off-nominal)
- **Robotic Operations**
  - Explorers and scouts on the lunar surface
  - Assistants and tools for human explorers
- **Lunar Infrastructure Operations**
  - Control of habitats, communications and power equipment, etc.
- **Unmanned Spacecraft Operations**
  - Remote system operations (nominal and off-nominal)

Operation challenges

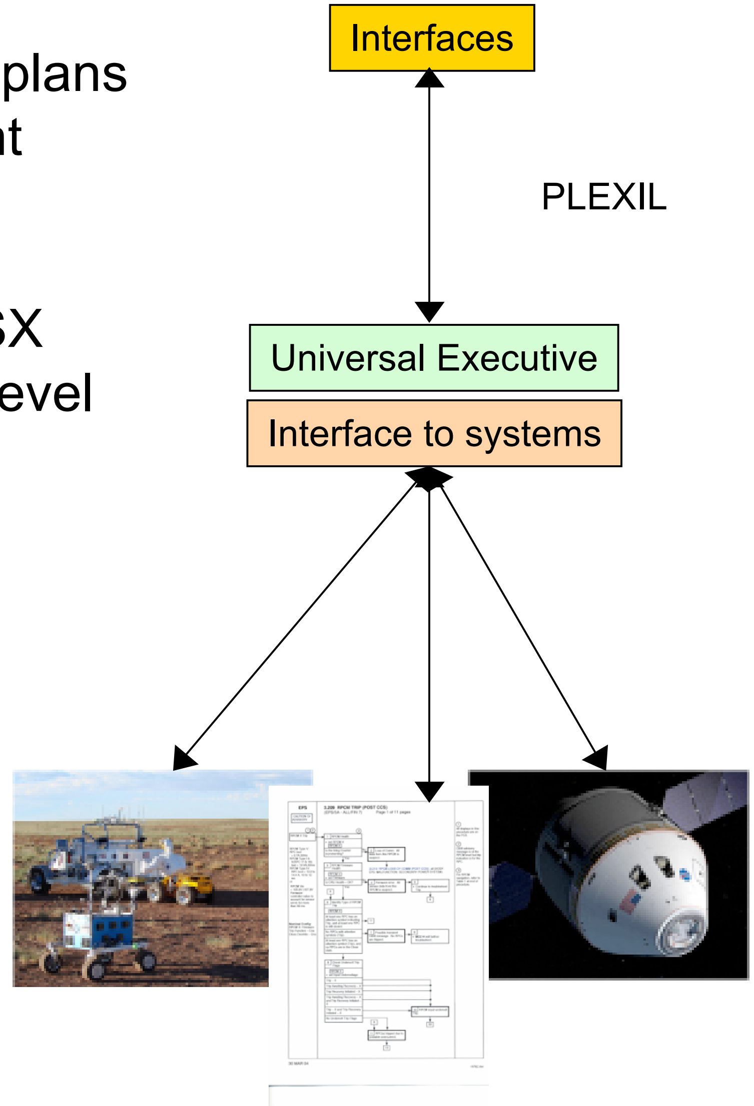
- **Mission Operations**
  - State of art : Many tools, lack of interoperability
  - Need: Flexible, evolvable and sustainable mission operations paradigm
- **Crewed Spacecraft Operations**
  - State of art : Crew relies on ground to support and control operations
  - Need: Crews able to operate systems and own tasks more independently
- **Robotics Operations**
  - State of art: Requires multiple operators for command and monitoring
  - Need: Effective sustainable robot operations with less human oversight
- **Lunar Surface Operations**
  - State of art : Ground-based operation of most surface assets
  - Need: Effective sustainable robot operations with less human oversight
- **Unmanned Spacecraft Operations**
  - State of art: Requires direct human command and monitoring
  - Need: Effective and reliable operations with less human oversight

Approach: A4O

- **Key elements of technology**
  - Re-usable, interoperable and adaptable architecture
    - **Data-driven general and re-usable modules**
    - **Common data specifications support adaptability, evolvability and interoperability of tools based on standards developed by CSI**
  - Automation capabilities
    - **Monitoring and analysis of telemetry and system states**
    - **Decision Support: From help for users to on-board decision-making**
    - **Execution: Carry out decisions and plans, from humans and automation**
  - Human interaction support
    - **Adjustable automation allows humans to handle more or less as needed**
    - **Assistance provides summary of information, options, evaluations, warnings**
    - **Complementary capabilities based on computational power**
- **Flexible and reusable - on ground and on board**
  - Enable transition from initial manual flights to sustainable operations
  - Same core capabilities used on ground, in flight and on lunar surface

Executive

- **Executive**
  - Lightweight engine for executing PLEXIL plans
    - Small memory and processor footprint
  - General and reusable
    - Same engine for many applications
  - Compiles on VxWorks, Linux, Solaris, OSX
    - Simple, well defined interface to low level control
  - Commanding interface
    - Sensing interface
  - Provides tools for users
    - Verifying, validating, simulating, and debugging
- **Applications**
  - Drives procedure execution automation
  - Executes plans for on-board operations
  - Runs K10 rover activity plans on board



Procedure representation

- **Procedures**
  - Notion generalizes a number of existing concepts:
    Command sequences, plans, checklists, diagnosis procedures, etc.
- **Procedures for both humans and automation**
  - PRL: Human-understandable; e.g., operations procedures
  - PLEXIL: Machine-understandable; e.g., plans and command sequences
  - Need a combination to enable adjustable automation
- **Procedure Representation Language (PRL)**
  - Combines ISS procedure schema with PLEXIL schema
  - XML-based language
- **Elements of PRL**
  - **Meta data** provides names, context, version, etc. for procedure
  - **Control data** provides logical control and safety conditions
  - **Steps and nodes** structure procedure for human readability
  - **Instructions** specify instructions, commands, etc.

Executive validation

- Main focus: how to validate procedures?
- We have five major efforts under way
  - Definition of formal semantics of PLEXIL language
  - Model-based generation of test plans for PLEXIL
  - Model checking of PLEXIL procedures
  - Simulation of PRL procedures
  - Model checking of PRL procedures

Procedure representation

- **PLEXIL**
  - Plan Execution Interchange Language
    - For describing plans, sequences, procedures, scripts, etc.
  - Simple syntax that is very powerful
    - Timed command sequences, event driven sequences, monitors
    - Concurrent execution, repeating sequences, etc.
    - Contingencies, conditionals, etc.
  - Designed to facilitate validation and certification
    - Guarantees unambiguous execution
    - Provides guarantees against deadlocks
    - Simple syntax facilitates validation and checking
  - General and reusable
- **PLEXIL is logical automation core of PRL**
  - Control logic and safety conditions in PRL map to PLEXIL
  - Execution semantics and properties of PLEXIL extend to PRL

Model checking of procedures

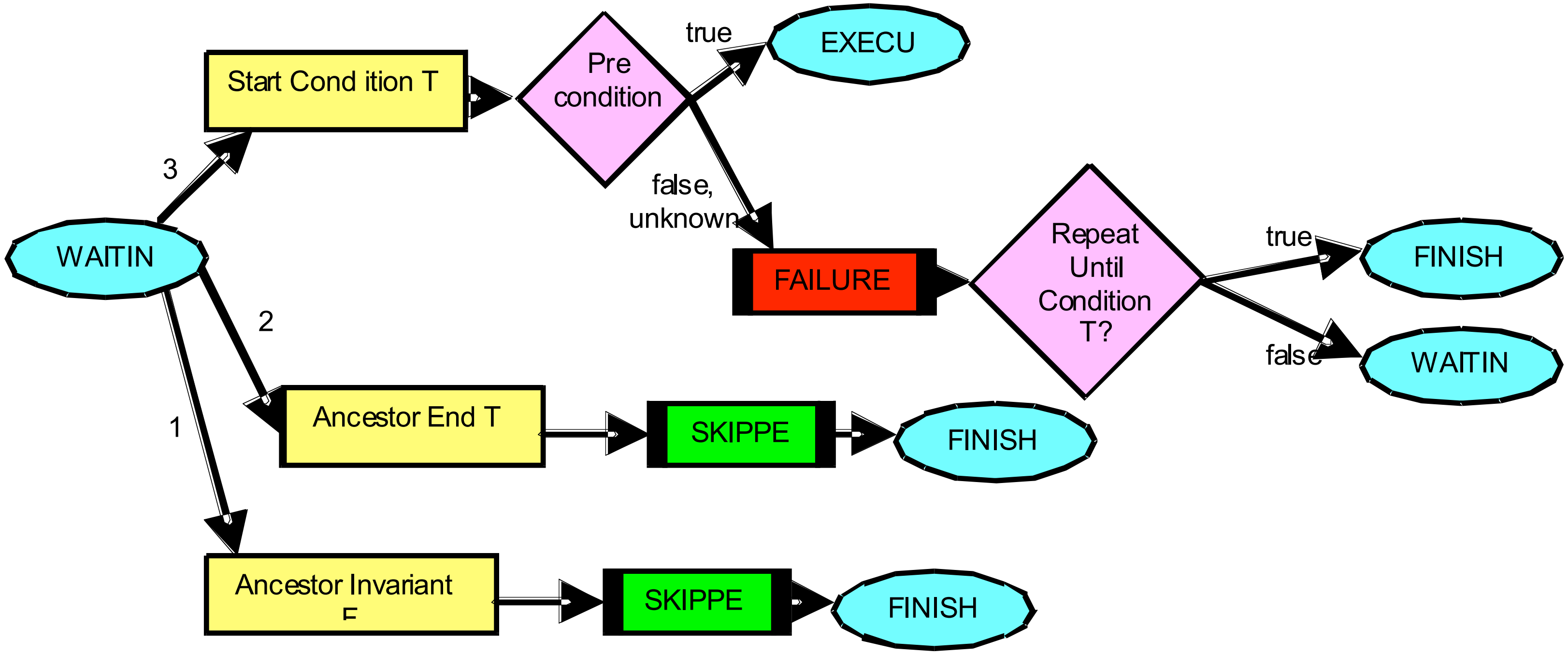
- We investigate two ways of applying model checking to procedures
- Compositional model checking using LTSA:
  - Build Labelled Transition System Analyser (LTSA) models for
    - underlying physical system (e.g., using FSM models for simulation)
    - procedures
  - Define safety properties of interest for the procedures
  - Model check the LTSA models using compositional techniques to alleviate the state explosion problems
- SMART model checking:
  - Build SMART models of PLEXIL macros
  - Check for deadlock and behavioral correctness properties
  - Investigate scalability of the approach by defining appropriate abstractions

Formal semantics of execution language

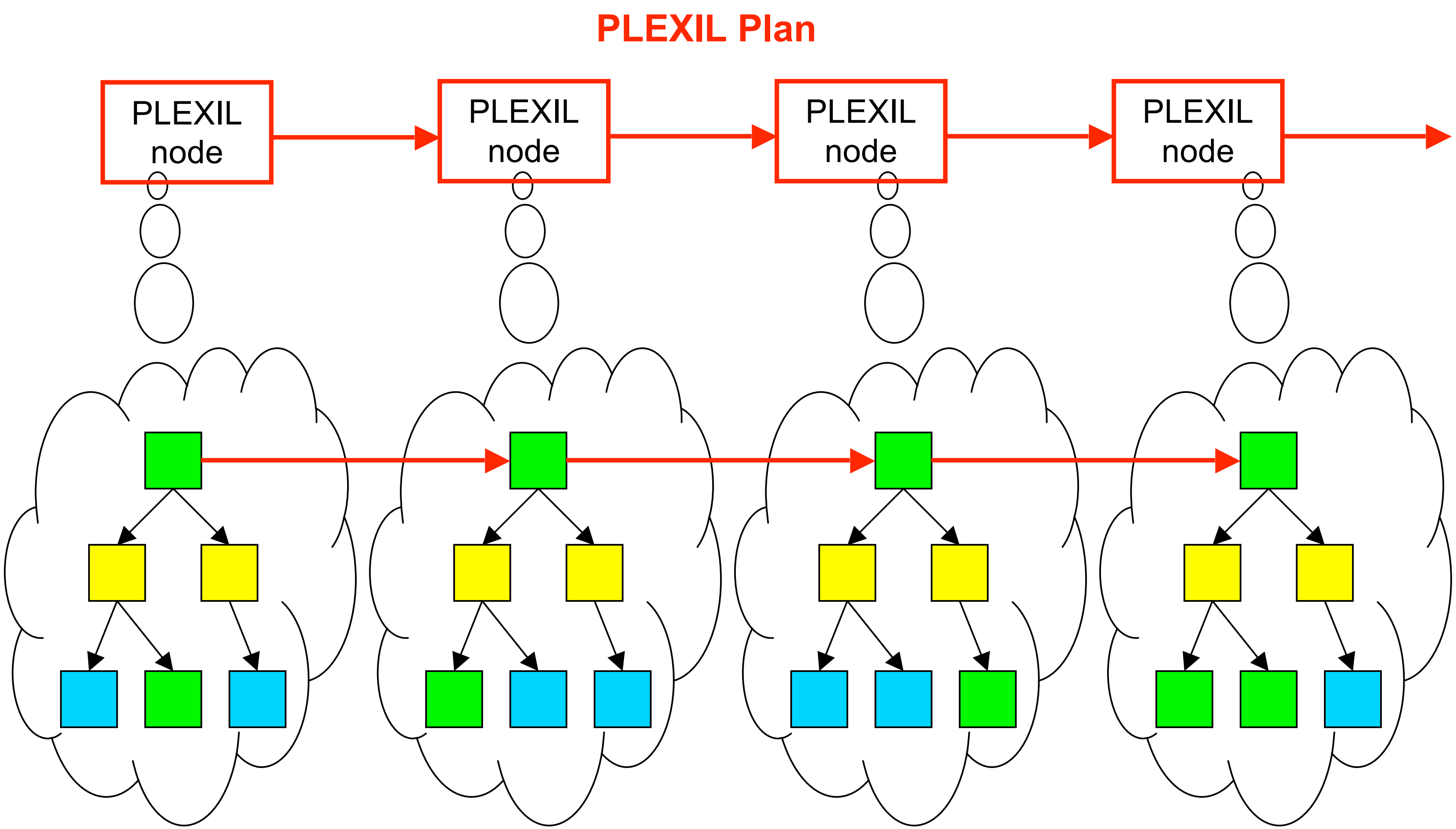
- The definition of formal semantics of PLEXIL language is necessary for the development of formal verification tools
- Our approach:
  - Described behavioral formal semantics of PLEXIL in LTSA models
    - Detection of subtle execution errors in PLEXIL models
    - Automatic translation of PLEXIL procedures into LTSA models
  - Described formal semantics of PLEXIL in PVS
    - Prove determinism and behavioral determinism for the PLEXIL language

Behavioral models for PLEXIL

- Behavioral model for the state *waiting* of a PLEXIL node

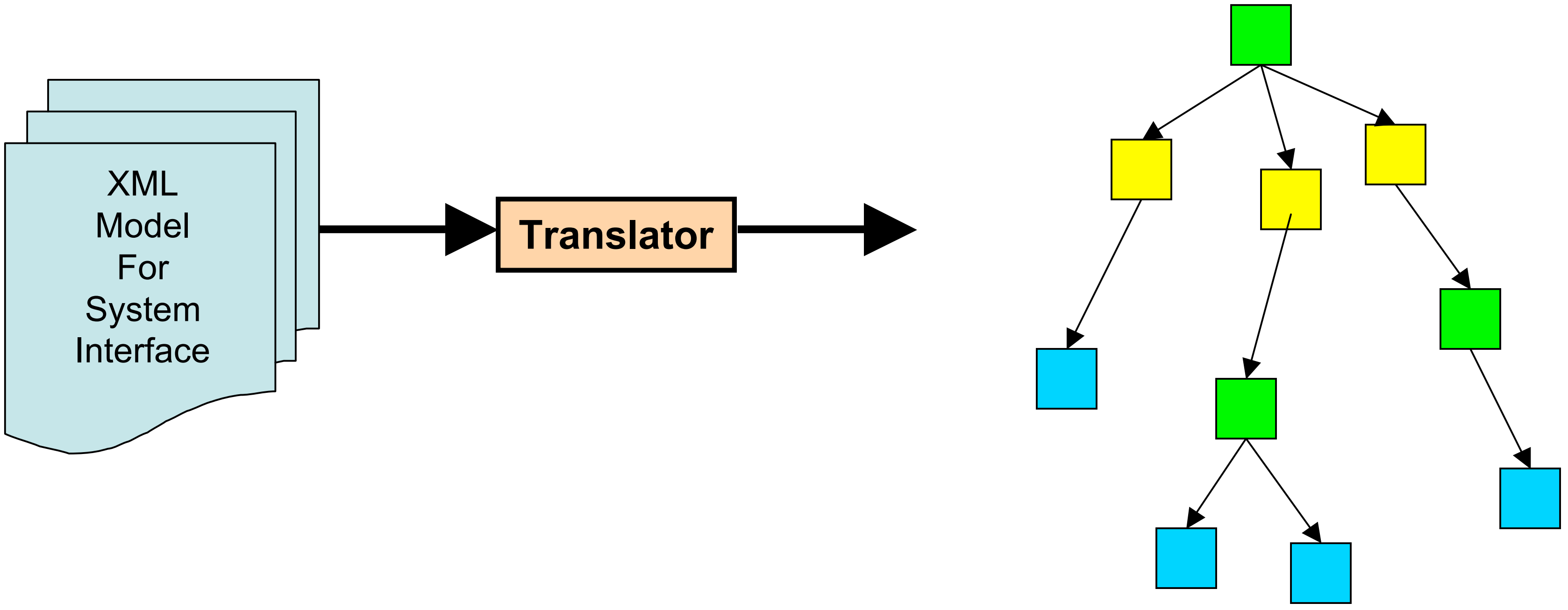


Composition of node models



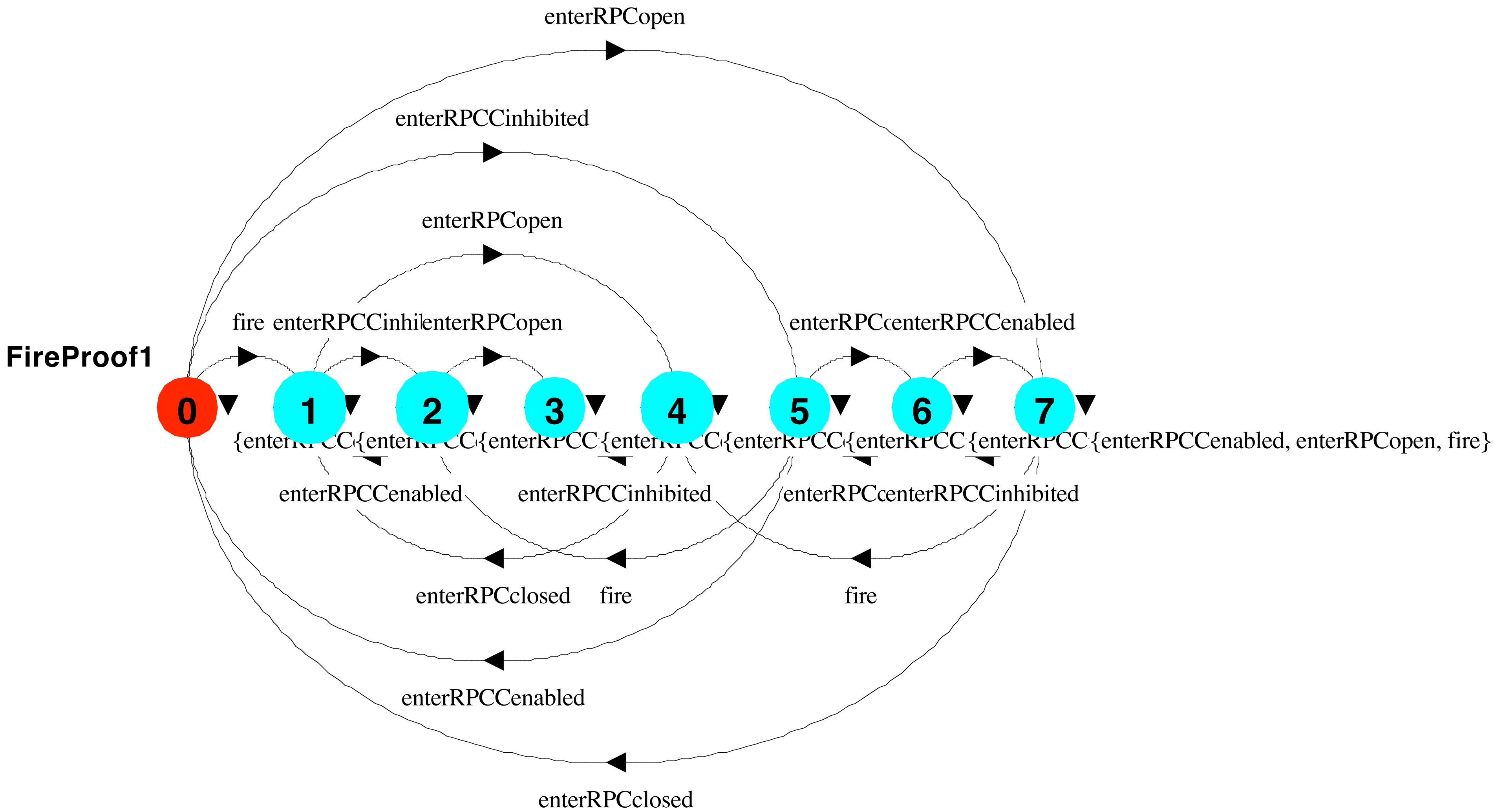
Composed LTSA Model for PLEXIL Plan

Translation of System Models

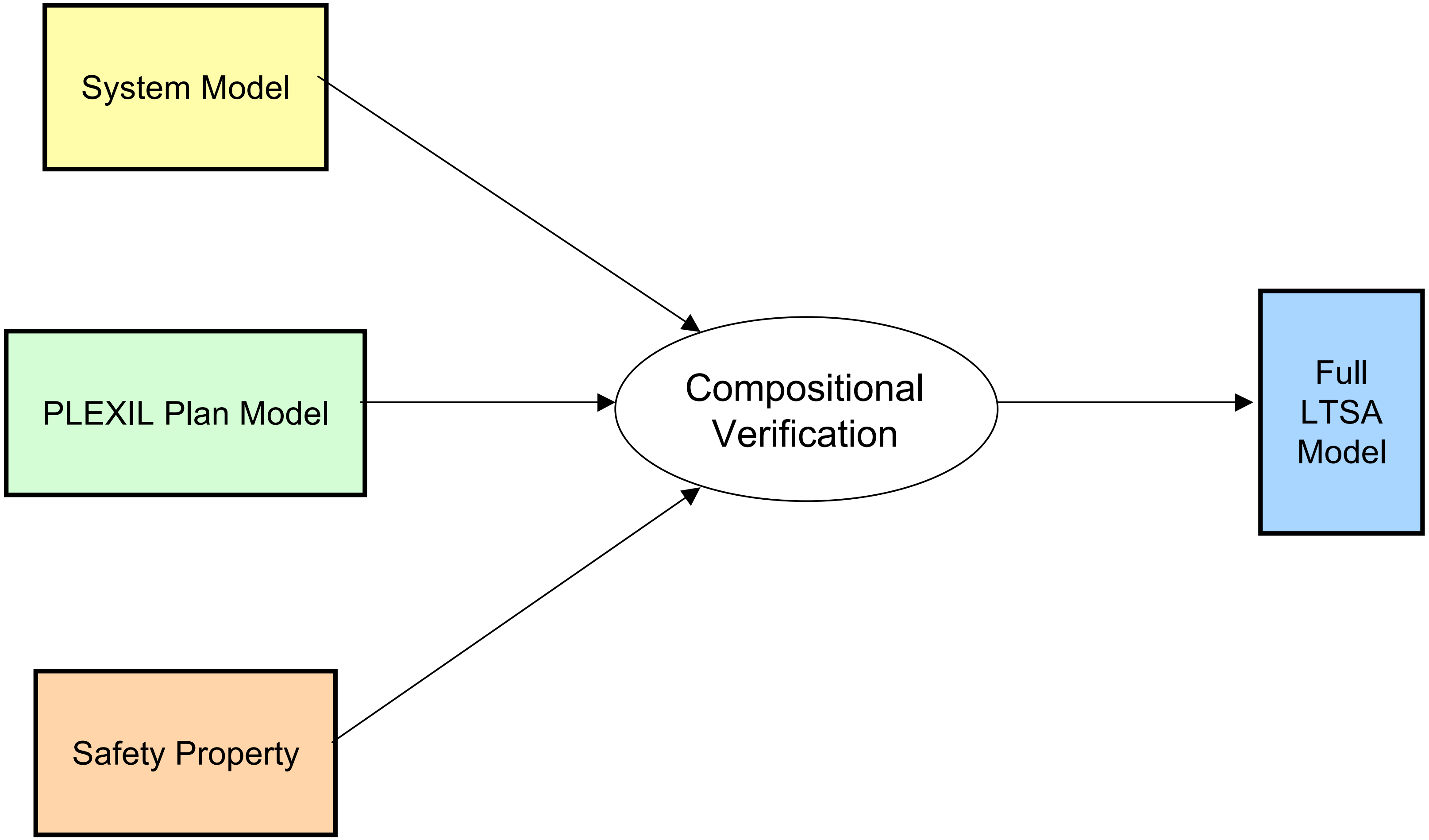


LTSA Model for System Interface

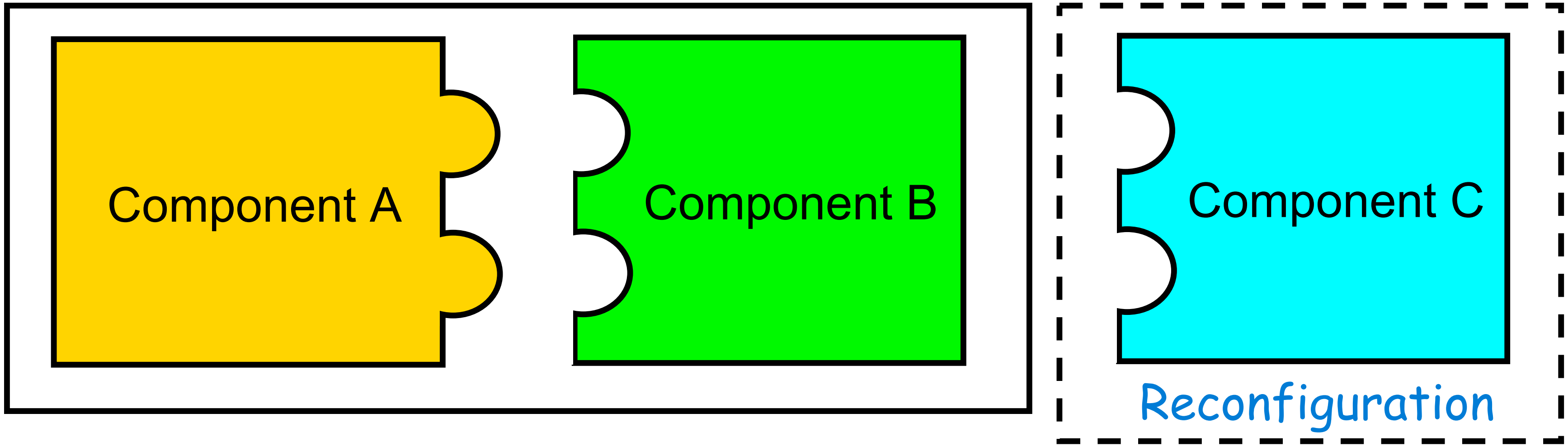
Example of safety property in LTSA



Compositional Verification

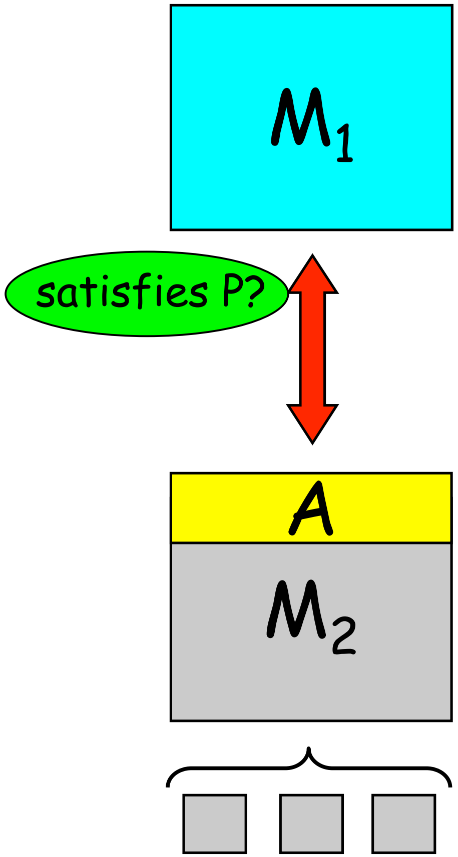


Compositional V&V



- Design-level: decompose (architecture)
  - establish contracts (assume-guarantee pairs) between components to guarantee key system-level properties
- Code-level: verify and test
  - verify or test each component against its individual contracts
- Reconfiguration
  - verify new components against contracts of substituted ones

Compositional Verification

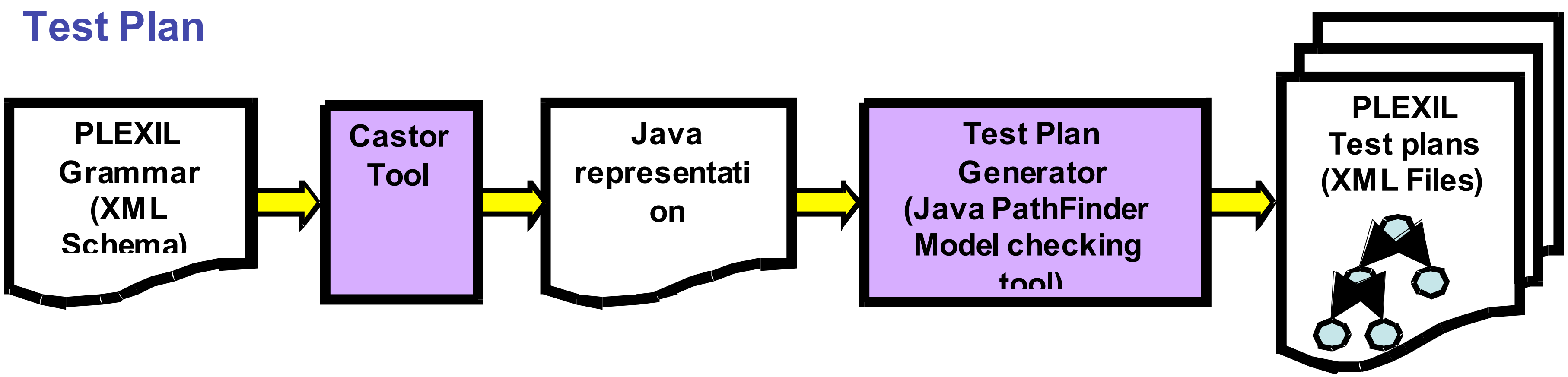


- Decompose properties of system ($M_1 \parallel M_2$) in properties of its components
- Does M_1 satisfy P?
  - typically a component is designed to satisfy its requirements in *specific* contexts / environments
- Assume-guarantee reasoning: introduces assumption A representing M_1's "context"
- Simplest assume-guarantee rule

$$\frac{\begin{array}{lccc} 1. & \langle A \rangle & M_1 & \langle P \rangle \\ 2. & \langle \mathit{true} \rangle & M_2 & \langle A \rangle \end{array}}{\langle \mathit{true} \rangle \; M_1 \parallel M_2 \; \langle P \rangle}$$

"discharge" the assumption

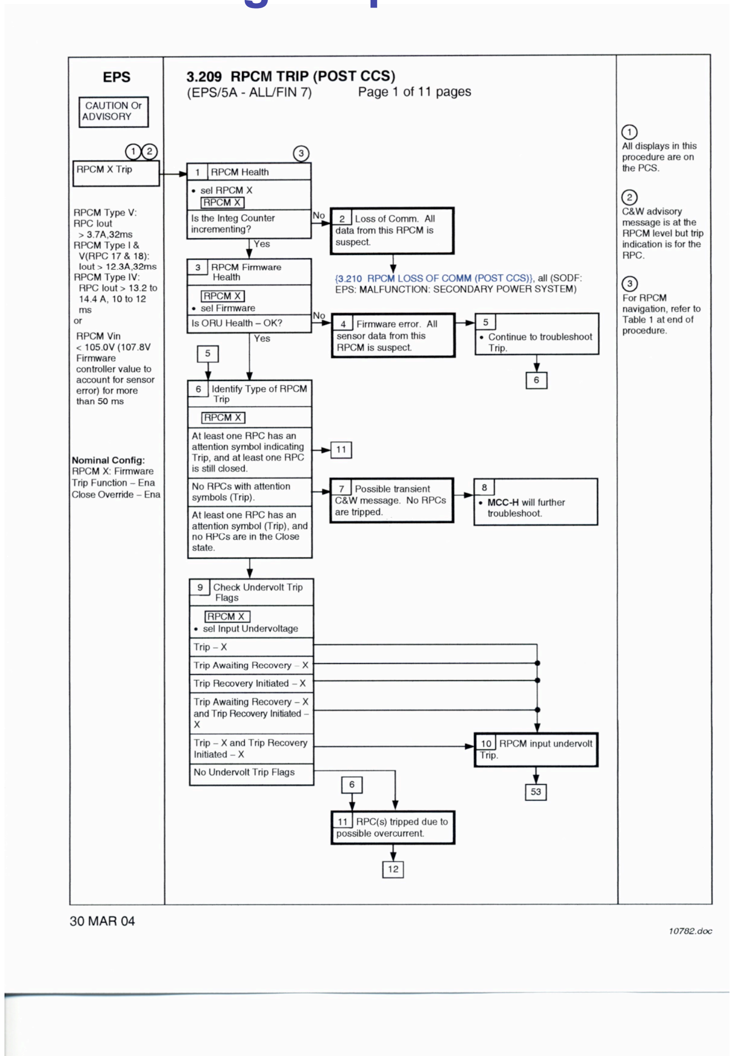
Model-based Plexil testing



- The goal is to automatically generate procedures for testing PLEXIL based on the PLEXIL grammar
  - The Castor-based translation is done
  - The test plan generation is inherited from previous research

PRL Example

Original procedure

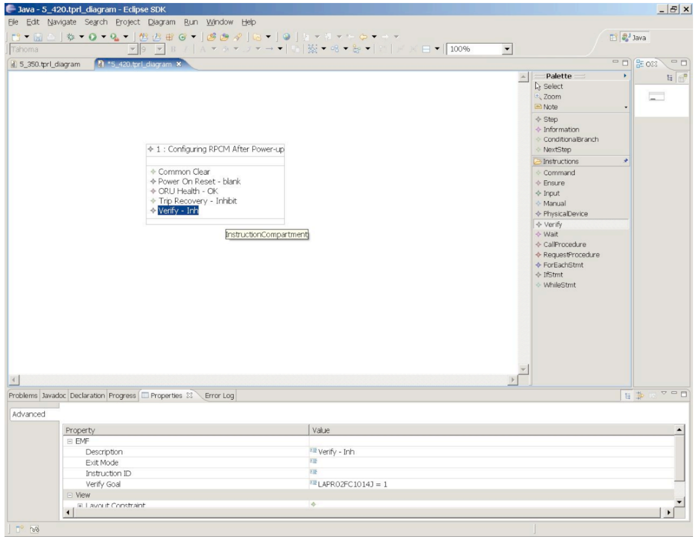


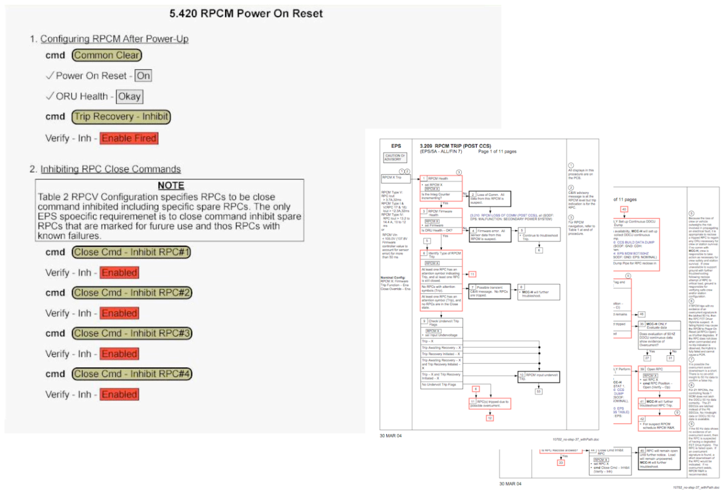
Encoding in PRL

```
<Step stepId="step3">
  <StepTitle>
   <StepNumber>3</StepNumber>
     <Text>RPCM Firmware Health</Text>
   </StepTitle>
   <InstructionBlock>
     <Instruction instructionID="step3_i1">
       <VerifyInstruction>
         <VerifyGoal>
            <TargetDescription>
               <Text>Verify ORU Health OK</Text>
            </TargetDescription>
....
```

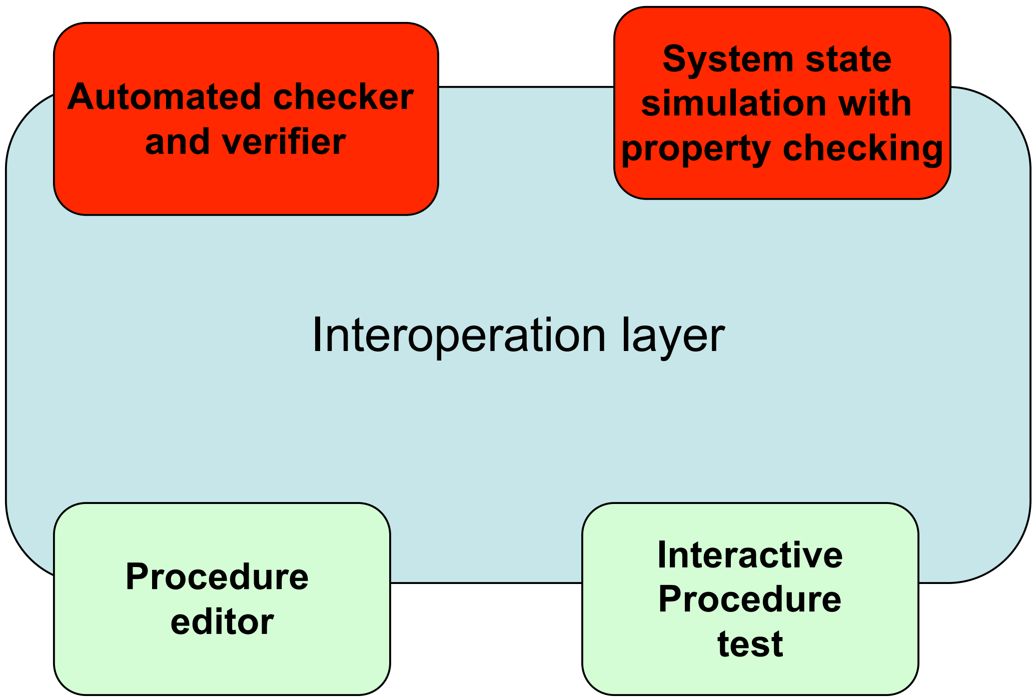
Procedure authoring and checking

- **Authoring**
  - Graphical and Textual Editing
  - Syntax checking and Syntax constraints
- **Viewing**
  - Static and Dynamic views on procedures
- **Procedure Checking**
  - Check procedures against flight rules
  - Check procedures against constraints
  - Assist in evaluation of simulation results
  - General interface supports plug and play of validation components
- **Configuration and workflow management**
  - Support workflow, including repositories, signoffs, etc.





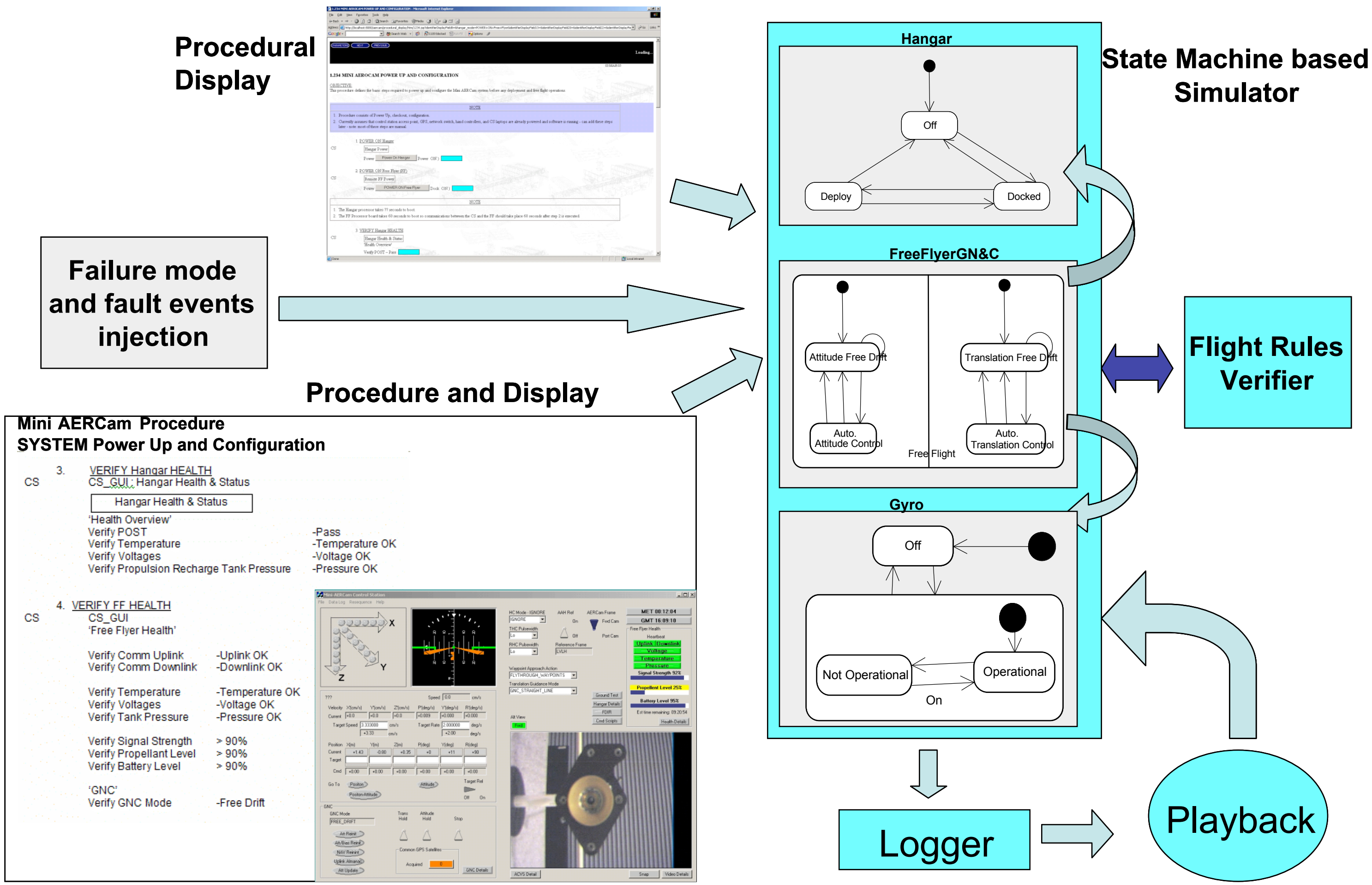
Procedure editing environment



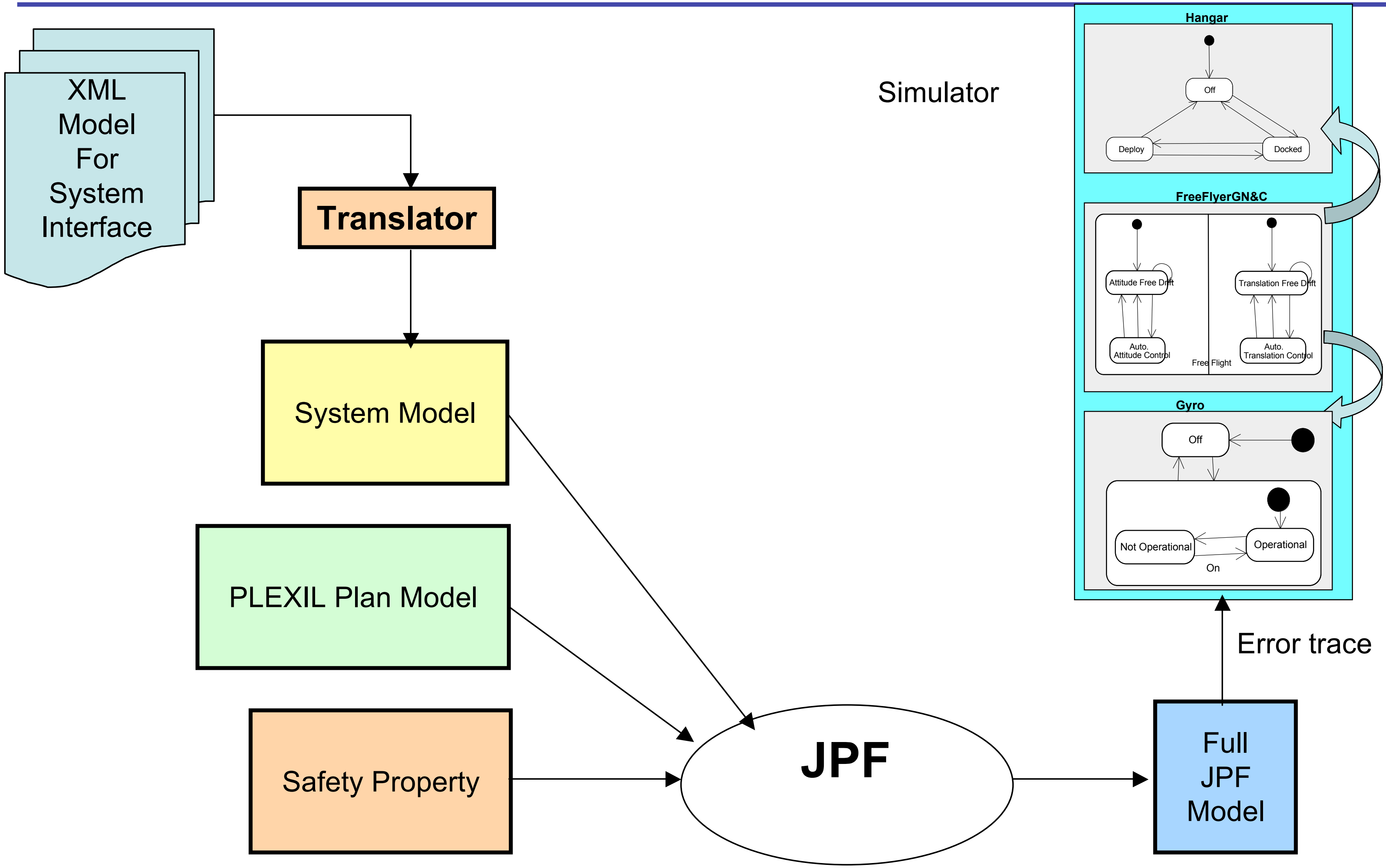
Simulation of PRL procedures

- Build finite state machine (FSM) models describing the underlying physical system (at least, its interface to the operator world)
- Simulate the execution of the procedure in conjunction with the FSMs
- Identify missing pre-conditions for nominal state execution

Model-based simulation of procedures

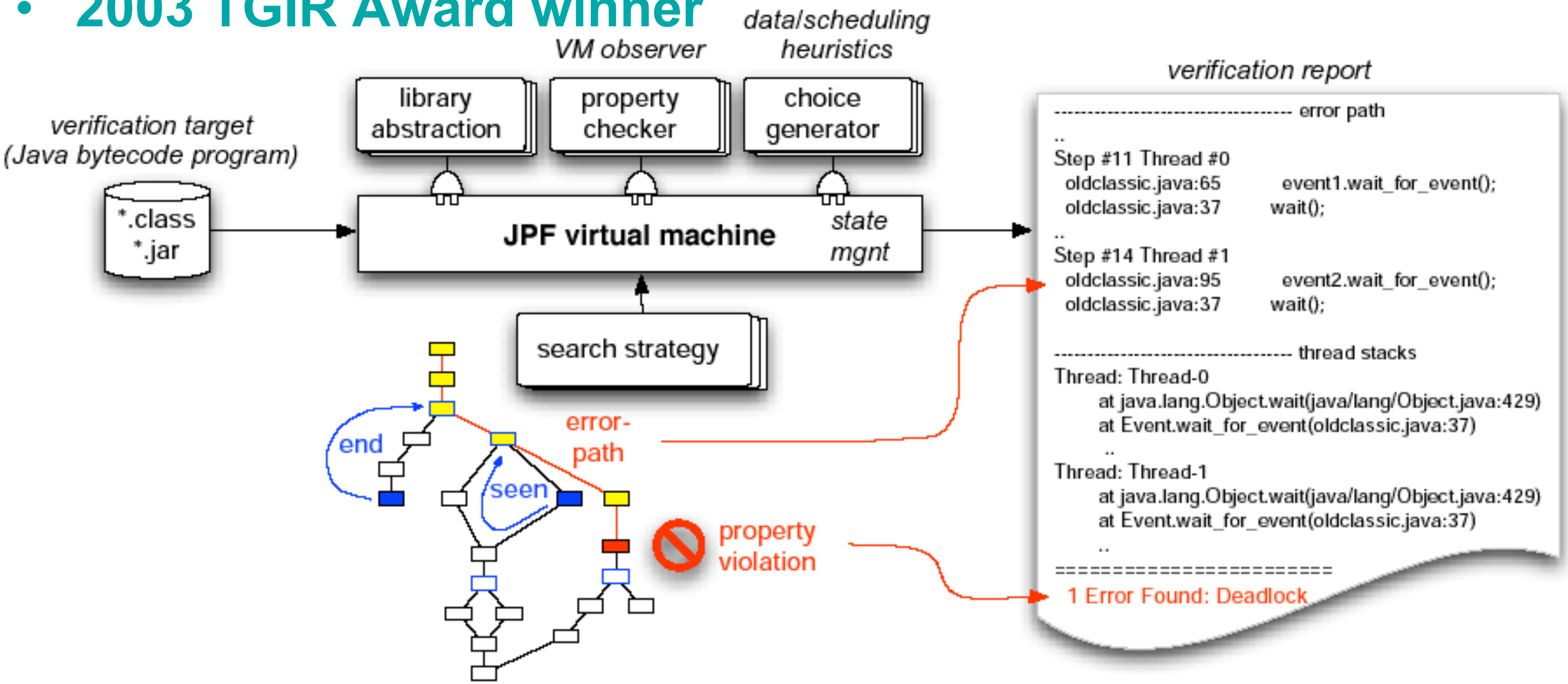


Model checking of PRL Procedures



Java Pathfinder

- **It is an extensible explicit state software model checker for Java byte code.**
- **Open-sourced on 28 April 2005**
  - **http://sourceforge.net/projects/javapathfinder/**
- **2003 TGIR Award winner**



Decision Support V&V

- Validation of planning models by translating them into model checking models
- Validation of plans and plan robustness
- Automatic generation of test cases to test against flight rules

Validation of planning models

- The goal is to study validation of planning models by translating them into SAL model checking models
- Approach:
  - Definition of a simple planning language, called APPL (A Plan Preparation Language), based on NDDL that is more amenable to formal verification
  - Automatic translation from APPL models to NDDL models
  - Automatic translation from APPL models to SAL models
    - We also study the relationship between APPL and the language unifying NDDL and Casper
  - Investigation issues of representation in SAL so that scalability problem can be avoided
    - For example, the representation of time and timers

Automatic generation of tests for planner

- The goal is to automatically generate test cases for planners so that we can test against flight rules
- Process:
  - Modeling flight rules in appropriate language
    - We started with LTL (linear temporal logic), but are considering others
  - Generate coverage conditions that cover flight rules according to "unique cause" criterion
    - "Unique cause" is an extension of the commonly used MC/DC coverage criterion mandated by the FAA
  - Generate test case in the form of Europa goals (or partial plans) using the coverage conditions

Test case generation for NDDL