A Symbolic Model Checking Approach to On-Board Autonomy

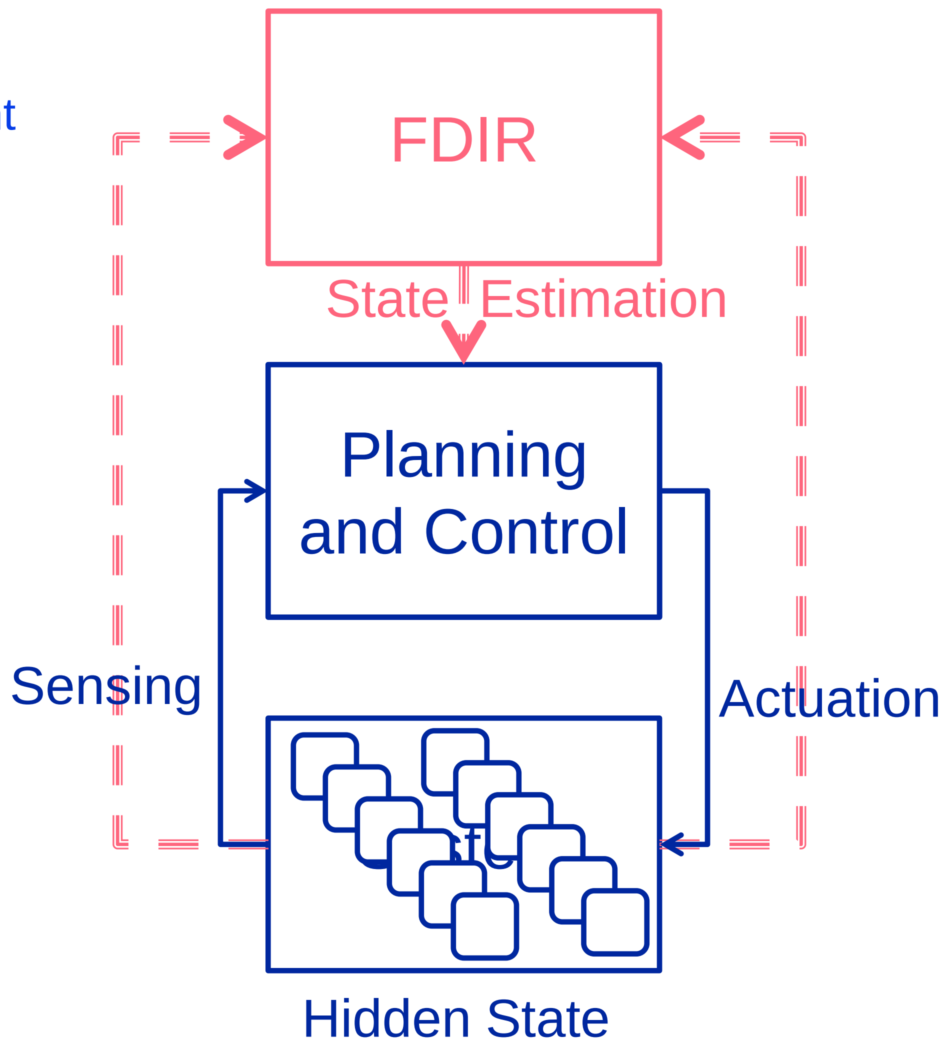
Alessandro Cimatti
Embedded System Unit
Fondazione Bruno Kessler
Trento, Italy
cimatti@fbk.eu

j.w.w. L. Bonetti, M. Bozzano, R. Cavada, A. Griggio, A. Micheli , S. Mover, M. Roveri, A. Tchaltsev, S. Tonetta, J.-P. Katoen, V.Y. Nguyen, T. Noll, R. Wimmer, A. Hoffman, M. Niezette, K. Kapellos, R. Steel,

We gratefully acknowledge the support of the European Space Agency contracts OMC-ARE, COMPASS, IRONCAP.

Complex autonomous systems

- Example: planetary rover
  - Communication unavailable, lags
  - Unpredictable, hostile environment
- Complexity
  - System composed of multiple heterogeneous subsystems
  - Functions: navigate in unknown terrain, drill, acquire sample
  - Conflicting objectives: do science vs preserve integrity
- Resource constraints
  - time, power, …
- Limited observability
- Possible faults
- Fault Detection, Isolation and Recovery
- Operation in degraded modes

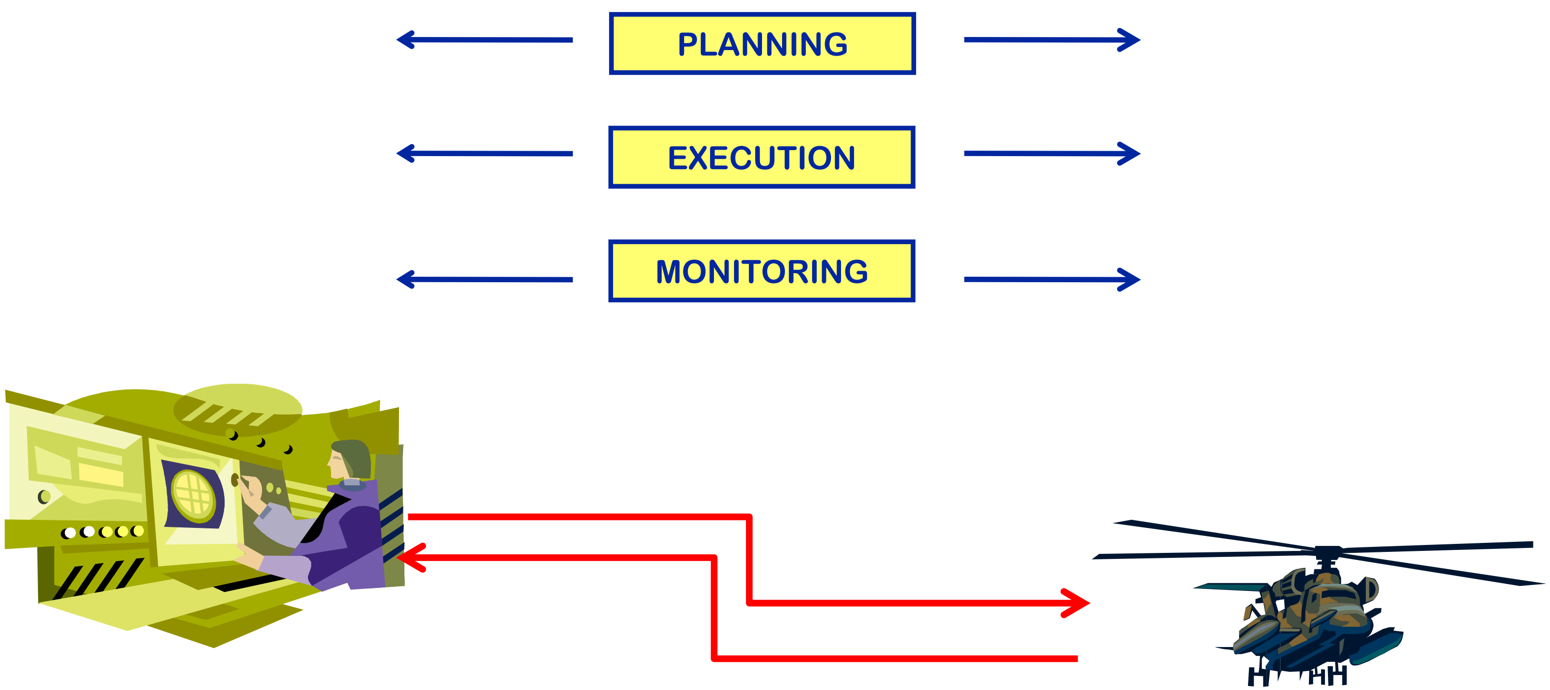


Design vs operation activities

- Design phase activities
  - Requirements validation
  - Functional correctness
  - Safety/dependability assessment
  - Diagnosability
- Operation phase activities
  - Planning
  - Execution Monitoring
  - Fault Detection, Fault Identification/Isolation
  - Fault Recovery
  - Replanning

Autonomy levels in operation

- Or, where are operation activities carried out?



Ground

On-board

Autonomy levels

- E1: Exec under ground control
- E2: Exec of pre-planned mission operations on-board
  - Action sequence planned on ground, lower level execution on board
  - Very common, applied to spacecrafts
- E3: Exec of adaptive mission operations on-board
  - High-level tasks planned on ground, adaptive execution on board
  - Foreseen in future missions
- E4: Exec of goal-oriented mission operations on-board
  - High-level mission goals on ground, all the rest on board
  - Currently at prototypical level

Key remarks

- The level of autonomy has a direct impact on the type of plan...
  - produced by the planning system (or team)
  - dealt with by the on-board executor
- The reasoning processes on-ground and on-board must be tightly related!
  - E.g. interpret on ground what happened on board
  - more CPU but less information

In this talk…

- How to support the design phase?
  - Helping designers to gain confidence
  - Build more predictable systems
  - Write more reliable software
  - Assess behaviour under faults
- How to support the operation phase?
  - Generate better plans
  - Monitor execution
  - Perform diagnosis
  - Support replanning
  - Recalibrate control strategies
- A comprehensive approach to autonomy based on symbolic model checking

Structure of the talk

- Motivations
- Support for design activities
  - The COMPASS project
- Support for operation activities
  - Discrete case
    - The OMCARE project
  - Continuous case
    - The IRONCAP project
- Conclusions

Model Checking in a nutshell …

- Reactive System
  - infinite computation, interacting with environment
  - communication protocol, hw design, control software, OS
  - modeled as a state transition system
- Requirements
  - desirable properties of system behaviour
  - modeled as formulae in a temporal logic (CLT, LTL, PSL, …)
- Does my system satisfy the requirements?

- Model checking
  - search configurations of state transition system
  - detect violation to property, and produce witness of violation
  - conclude absence of violation

Properties

- Safety properties
  - nothing bad ever happens
    - » never (P1.critical & P2.critical)
    - » always (P1.critical -> (P1.critical until P1.done))
  - state transition system can't reach a bad configuration
- Liveness properties
  - something good will happen
    - » always (P1.trying -> eventually P1.critical)
  - state transition system can not exhibit a bad cycle

Symbolic Representation

- State variables as variables in a logical language
  - x, y, z, w
- A state is an assignment to state variables
  - The bitvector 0011
  - The assignment { z, w }
  - The formula $\neg x \wedge \neg y \wedge z \wedge w$
- A set of states is a set of assignments
  - can be represented by a logical formula
  - $x \wedge \neg y$ represents {1000, 1001, 1010, 1011} or a larger set, if more variables are present
- Set operations represented by logical operations
  - union, intersection, complementation as disjunction, conjunction, negation
- I(X), B(X) are formulae in X
  - Is there a bad initial state?
  - Is $I(X) \wedge B(X)$ satisfiable?

Symbolic Representation

- Symbolic representation of transitions?
- Transition
  - pair of assignments to state variables
- Use two sets of variables
  - current state variables: x, y, z
  - next state variables: x', y', z'
- A formula in current and next state variables
  - represents a set of assignments to X and X'
  - a set of transitions
  - R(X, X')

BDD-based Symbolic Model Checking

- Based on Binary Decision Diagrams
  - canonical representation for logical formulae
  - boolean operations, quantifier elimination
- I(X), R(X, X'), B(X)
  - each represented by a BDD
- Image computation: compute all successors of all states in S(X)
  - based on projection operation
  - exists X.(S(X) and R(X, X'))
- Reachability algorithm
  - Expand new states until bug, or fix point

SAT-based symbolic model checking

- Use SAT solver instead of BDDs
- Represent I(X), R(X,X'), B(X) as CNF formulae
  - much smaller size than BDDs!

- *Bounded model checking* [BCCZ99]
- Focus on finding bugs
  - give up proof of correctness
  - try to falsify property, i.e. witness to violation
  - within given resource limit (bound)

Bounded Model Checking

- State variables replicated K times
  - X_0 , X_1, …, X_{k-1}, X_k
- Look for bugs of increasing length
  - $I(X_0) \wedge R(X_0, X_1) \wedge \ldots \wedge R(X_{k-1}, X_k) \wedge B(X_k)$
  - bug if satisfiable
  - increase k until …
- Other techniques:
  - K-induction, interpolation, abstraction-refinement, …
  - Advanced use of SAT solvers: incrementality, unsat core

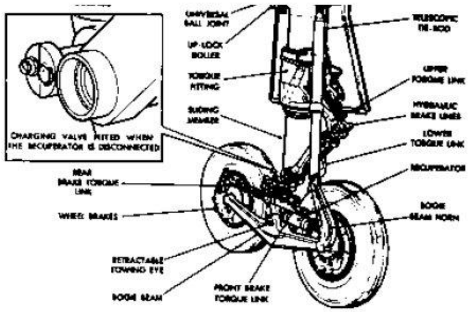
Thirty years of research…

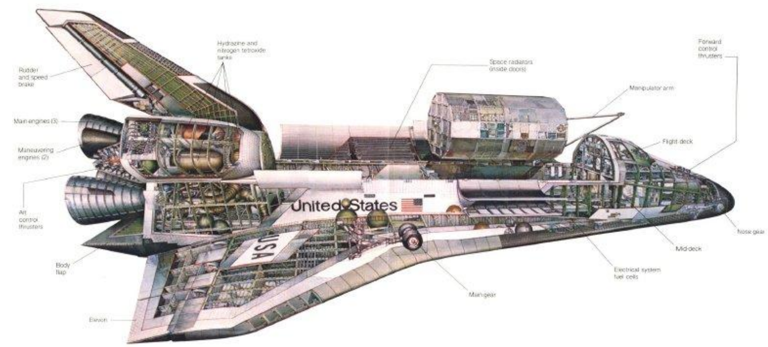
- The tecnology is becoming stronger
  - Standard practice in hardware design
  - Increasingly used in model-based development of critical software
    - Railways, avionics, …

- The NuSMV model checker
  - **`http://nusmv.fbk.eu/`**

- Key focus: functional verification!
  - But functional verification is not the end of the story…

From component to system-level design





Issues with current state of the practice

- SW verified in isolation from the target HW
- Limited support for specifying fault models and degraded modes of operation
- Safety and reliability models are separate from design models
- Different formalisms and analysis techniques for evaluating different aspects
- Limited support for analyzing timed and probabilistic properties
- No coherent approach to analyze effectiveness of FDIR (Fault Detection, Identification and Recovery)

System-Software Co-Engineering!

The COMPASS Project

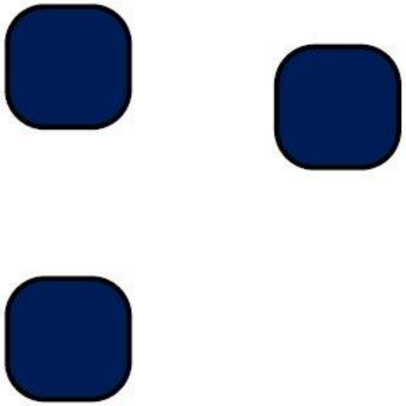
- COMPASS
  - Correctness, Modeling, and Performance of Aerospace Systems
- Integrated system-software co-engineering
  - A general-purpose specification formalism: the SLIM (System-Level Integrated Modelling) language
  - A comprehensive methodology based on formal methods
  - A toolset implementing the methodology
  - Demonstration and evaluation on industrial-size case-studies from the aerospace domain

- Consortium composed by RTWH, FBK-irst, TAS-F

The SLIM Language

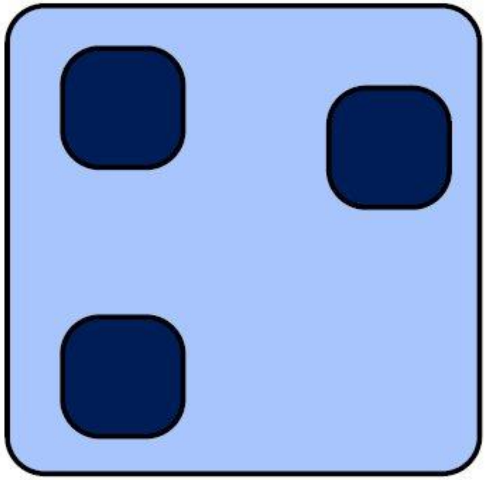
- An extension of AADL
  - Architecture Analysis and Design Language
  - Design language standardized by SAE (Soc. Automotive Engineers)
  - + EMA = Error Model Annex
- Designed to cover:
  - Degraded modes of operation
  - Qualitative and quantitative (probabilistic) properties
  - Probabilistic faults and recovery
  - Observability requirements
  - Property language covering functional correctness, safety and performability
  - Timed and continuous behavior
- Formal semantics defined in terms of
  - Networks of Event-Data Automata (NEDA)
  - Labeled Transition Systems (LTS)

The SLIM language



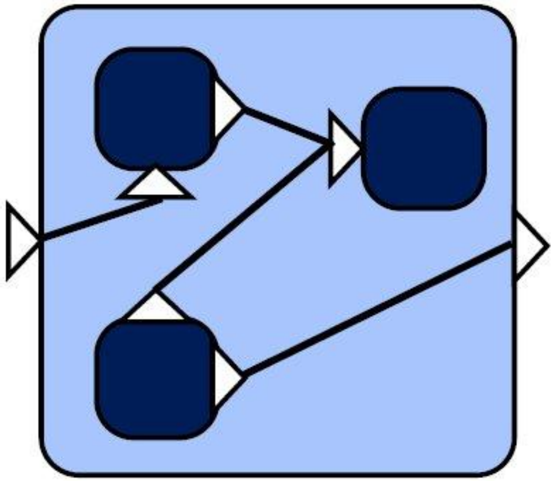
- Features:
  - Component-oriented (HW, SW, composite)

The SLIM language



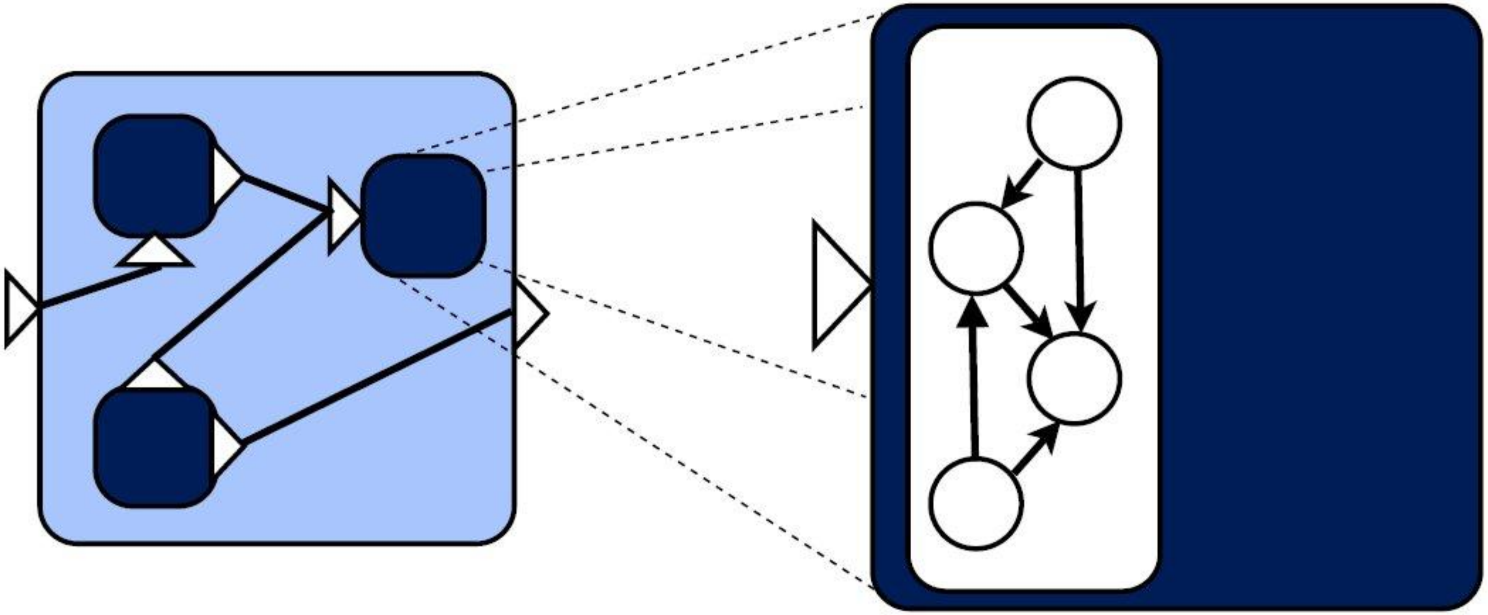
- Features:
  - Component-oriented (HW, SW, composite)
  - Hierarchy of super- and sub-components

The SLIM language



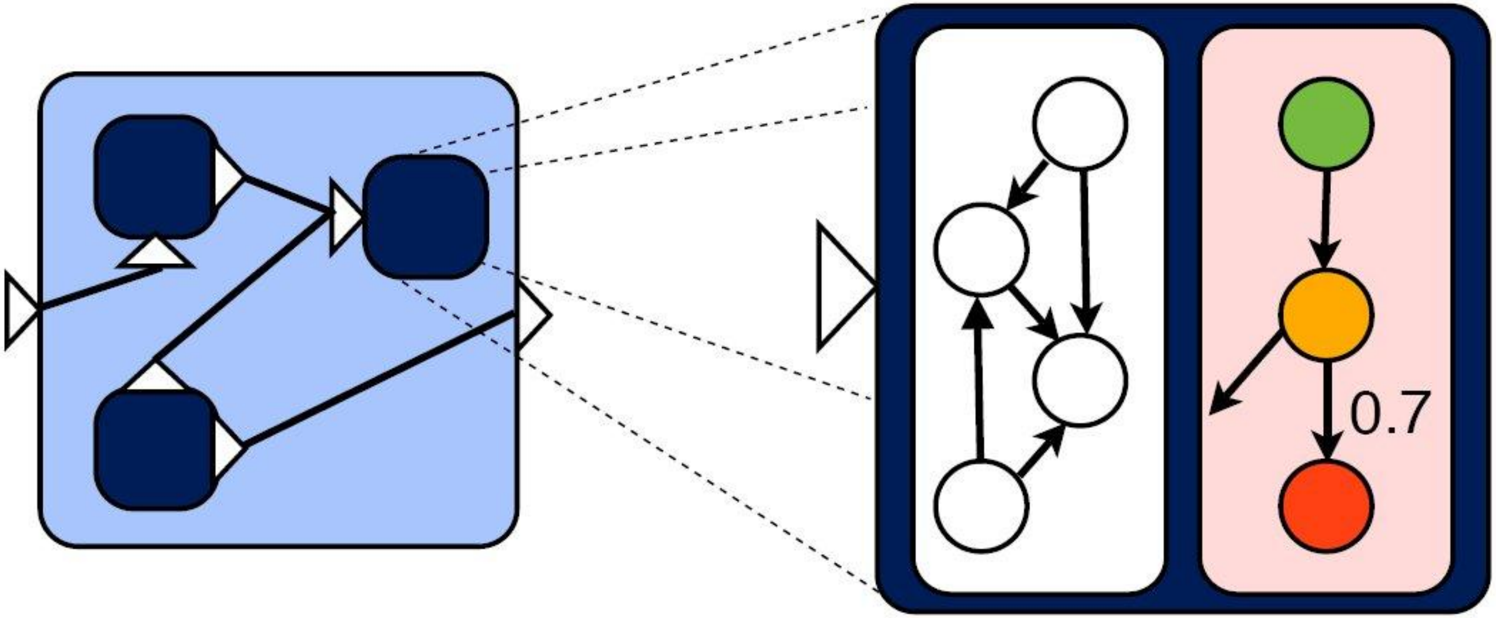
- Features:
  - Component-oriented (HW, SW, composite)
  - Hierarchy of super- and sub-components
  - Event and data ports

The SLIM language



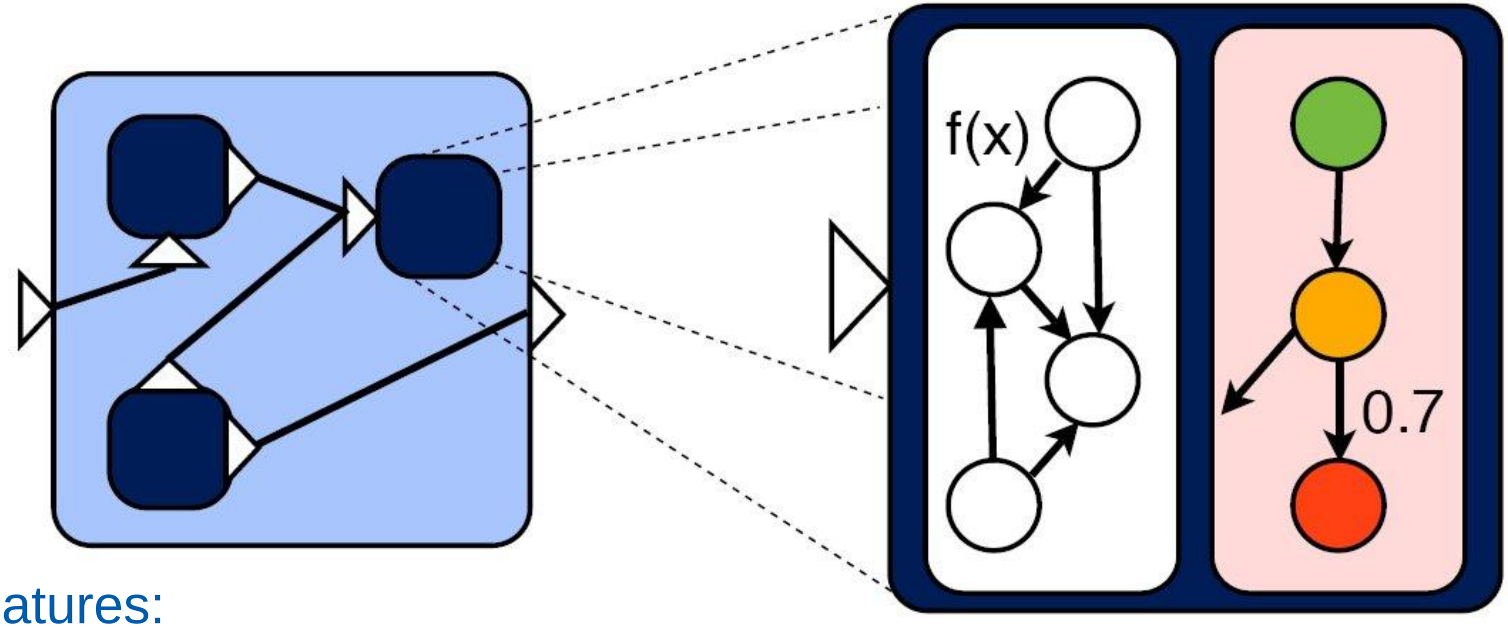
- Features:
  - Component-oriented (HW, SW, composite)
  - Hierarchy of super- and sub-components
  - Event and data ports
  - Functional behavior

The SLIM language



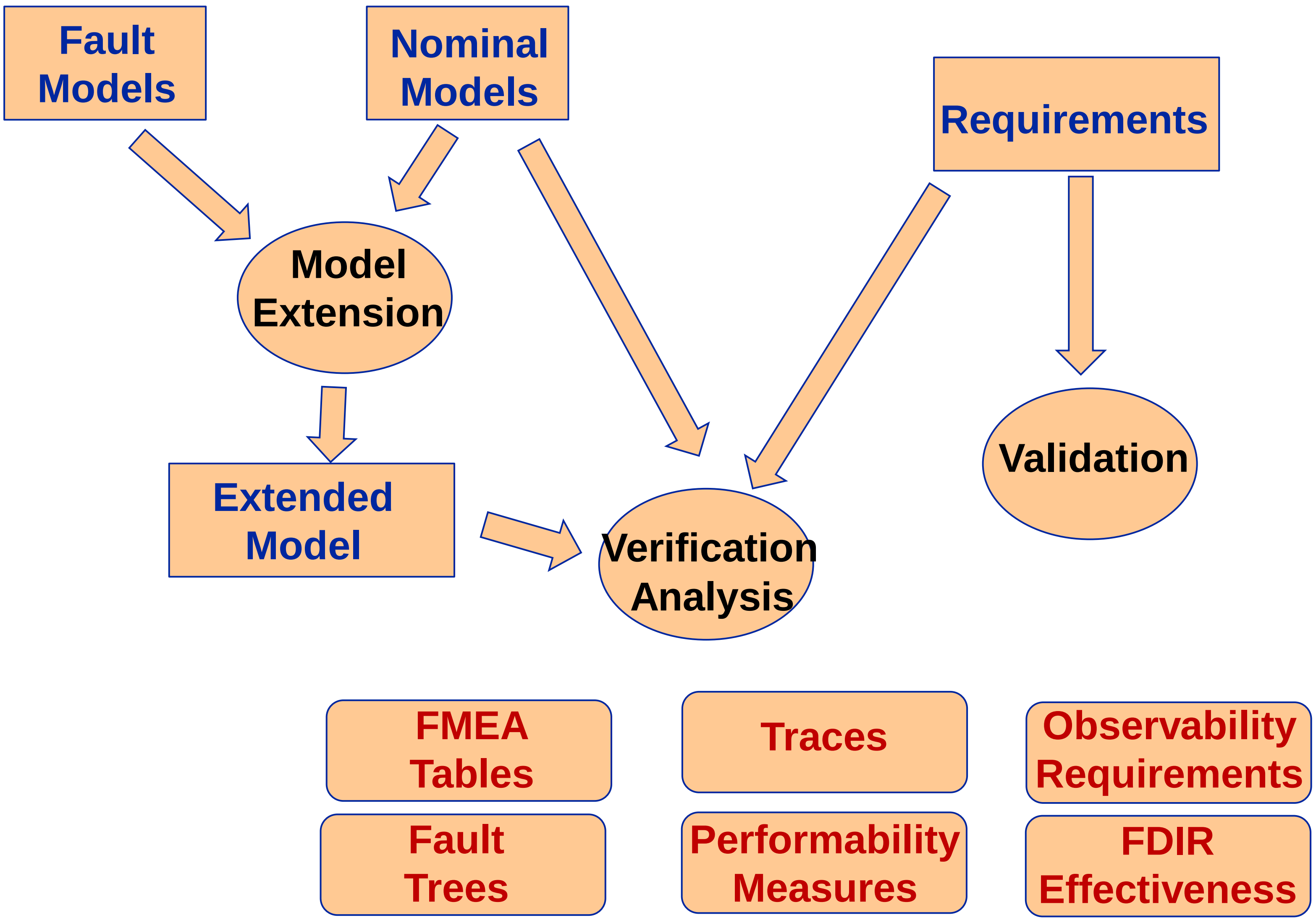
- Features:
  - Component-oriented (HW, SW, composite)
  - Hierarchy of super- and sub-components
  - Event and data ports
  - Functional behavior
  - Probabilistic error behavior (AADL Error Model Annex)

The SLIM Language



- Features:
  - Component-oriented (HW, SW, composite)
  - Hierarchy of super- and sub-components
  - Event and data ports
  - Functional behavior
  - Probabilistic error behavior (AADL Error Model Annex)
  - Hybrid behavior (not in AADL)

The flow of design phase



Requirements Validation

- The error is in the requirements, not in the system
  - a real user need
- Validate system requirements *before* detailed design and implementation
  - *“Are we capturing the right system?”*
- Available functionalities:
  - Property simulation
  - Check logical consistency
    - Are there any contradictions?
  - Check property strictness
    - Are the properties strict enough to rule out undesired behaviours?
  - Check property weakness
    - Are the properties weak enough to allow desirable behaviours?

- A whole research line on its own:
  - Temporal logic satisfiability engines
  - Diagnostic information: unsatisfiable cores
  - Relevant projects
    - Formal requirements validation of European Train Control System [ERA]
    - OthelloPlay [MRS research award]

Functional Correctness

- Correctness verification
  - *“Are we building the system right?”*

- Available functionalities:
  - Model Simulation
    - » Animate model to produce execution traces
  - Property Verification
    - » Check that a specification holds in all model traces
    - » E.g*. “always (voltage >= 5.8)”*

Safety Analysis

- Safety analysis
  - Evaluate hazards and risks
  - Check system behavior in presence of faults

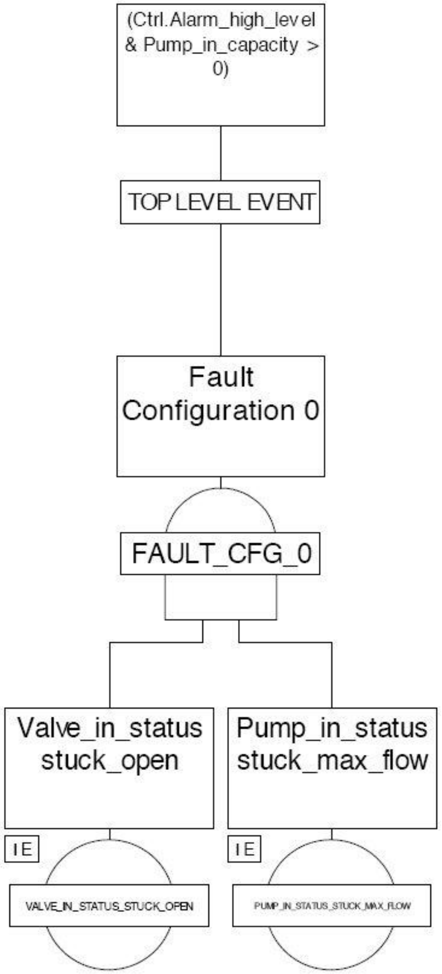
- Modeling combined nominal and faulty behaviour:
  - Nominal model annotated with possible faults
    - “Valve stuck at open”, “jammed engine”
  - Select model behaviour under fault
    - E.g. “constant value”, “ramp down until stop”
  - Combined behaviour automatically extended
    - Fault variables model presence of faults
    - Mutiplex nominal/faulty behaviour

- Analyses:
  - Fault Tree Analysis (FTA)
  - Failure Modes and Effects Analysis (FMEA)

- Based on the FSAP tool
  - Various UE projects: ESACS, ISAAC, MISSA
  - Recent book on topic [BV10]:

Safety Analysis

- Fault Tree Analysis (FTA)
  - Find the minimal combinations of faults that may cause a top event
    - » E.g.: "*Which combinations of faults may cause alarm to be raised*"
- Reduction to parametric model checking
  - Parameters are failure mode variables
  - Intuition:
    - » Find violation to property
    - » Extract assignment to fault variables
    - » Accumulate, block, and iterate until fix point



Safety Analysis

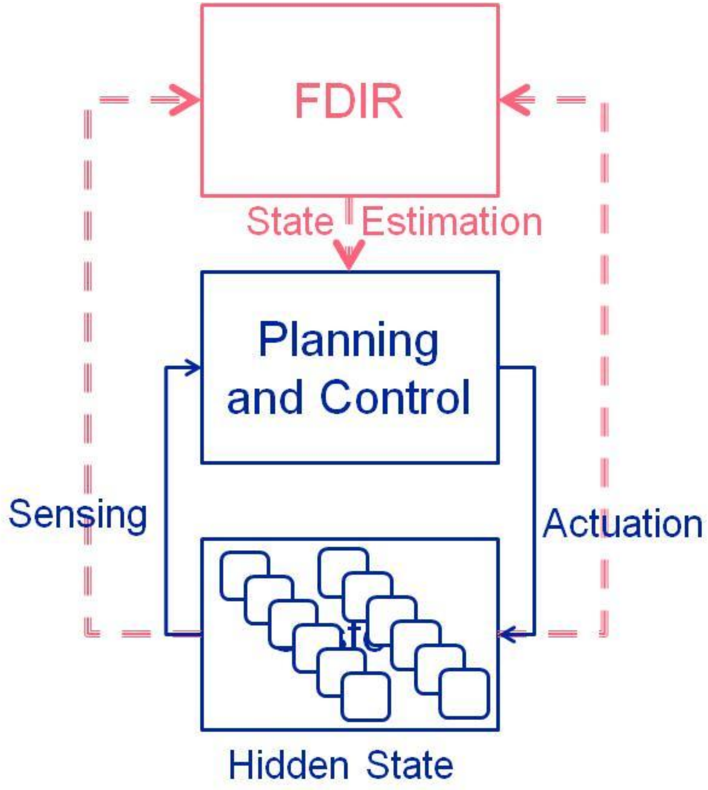
- Failure Modes and Effects Analysis (FMEA)
  - Analyze the impact of fault configurations on a set of system properties
    - » E.g. *"What are the consequences of a battery failure: i) on the output voltage of the power generator? ii) on the output alarm?"*

Ref. No.	*Item*	*Failure mode*	*Failure cause*	*Local effects*	*System effects*	*Detection means*	*Severity*	*Corrective Actions*
1	Pump	Fails to operate	Comp. broken No input flow	Coolant temperature increases	Reactor temperature increases	Temperature alarm	Major	Start secondary pump Switch to secondary circuit
2	Valve	Stuck closed	Comp. broken	Excess liquid	Reactor pressure increases	Coolant level sensor	Critical	Open release valve
3		Stuck open	Comp. broken	Insufficient liquid	Reactor temperature increases	Coolant level sensor	Critical	Open tank valve

- Reduction to model checking
  - Failure mode variables suitably constrained
  - Simplify extended model
  - Solve multiple properties in simplified model

FDIR effectiveness analysis

- Fault Detection
  - *"Will given FDIR procedure always detect a fault?"*
- Fault Isolation
  - *"Will given FDIR procedure identify the fault responsible for an event?"*
- Fault Recovery
  - *"Will given FDIR procedure recover from a fault?"*

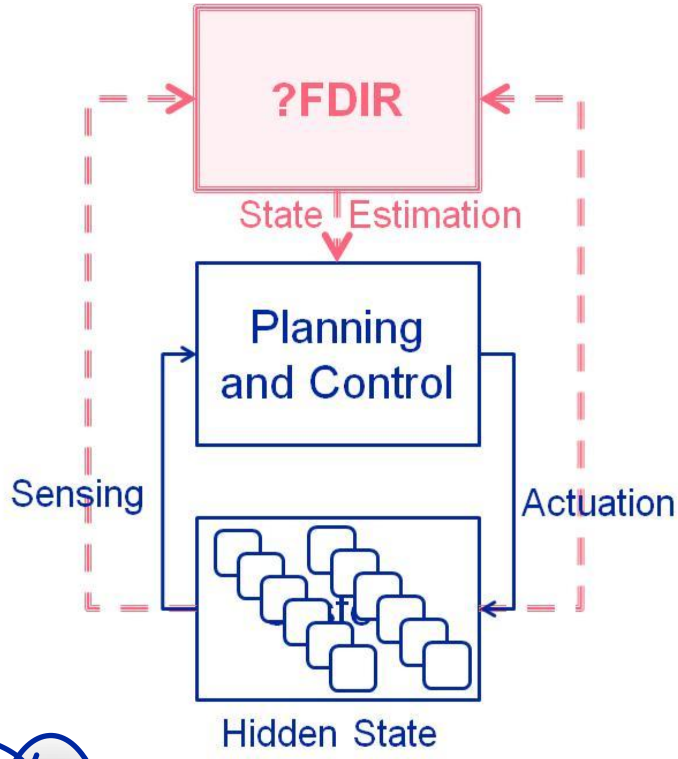


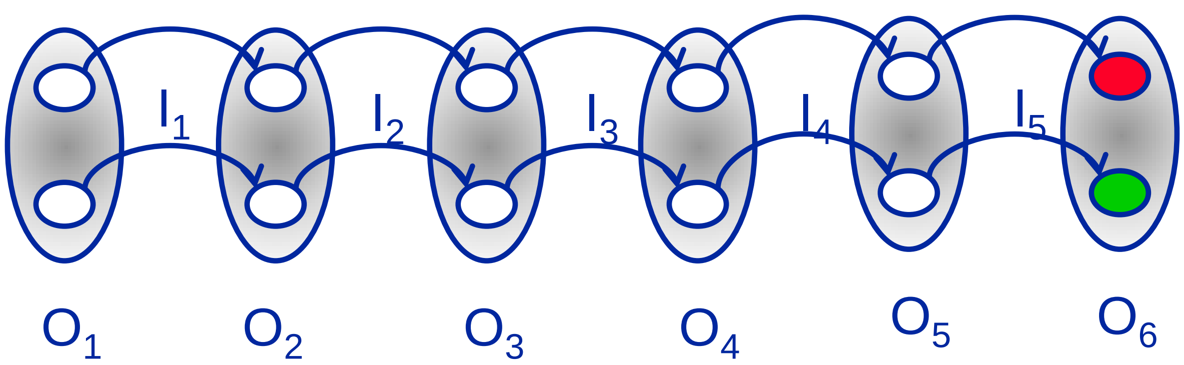
- Solved by direct reduction to model checking of extended model
  - Analysis of closed loop behaviour
    - » system + controller + FDIR

Diagnosability Analysis

- Diagnosis feasibility
  - *"Is there a diagnoser for a given property?"*
- Diagnoser synthesis
  - *"Find a good sensors configuration"*

- Diagnosability re-cast to model checking in the twin plant model:
  - *Twin plant: synchronous product of the model of the plant with itself imposing equality of the actions and of the observations*
  - *There is no pair of execution one reaching a bad state, the other reaching a good state, with identical observations*





Industrial Evaluation

- Thorough evaluation by industrial partners
- Several case studies developed
  - Thermal regulation function
  - Thermal line class 3
  - Satellite modes and FDIR procedures
- Positive evaluation results
- Code delivered to and accepted by ESA
  - Package includes comprehensive documentation

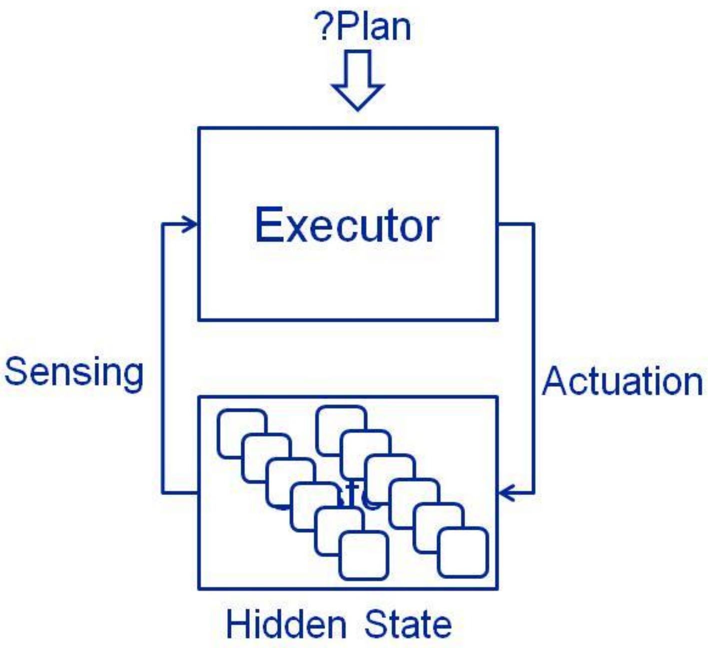
- Licensing: we are looking forward to it…
  - However, we are waiting for lawyers (as usual)
  - More intricate than expected
  - Distinction between EU and NON-EU member states
- Get in touch if interested in forthcoming distribution

Structure of the talk

- Motivations
- Support for design activities
  - The COMPASS project
- Support for operation activities
  - Discrete case
    - The OMCARE project
  - Continuous case
    - The IRONCAP project
- Conclusions

Planning via Symbolic Model Checking

- Representation of planning domain as symbolic finite state machine
  - Rich representational model, closer to actual modeling languages
  - Nondeterministic action effects
  - Multiple initial states
- Key insight: action sequence associated with multiple runs!

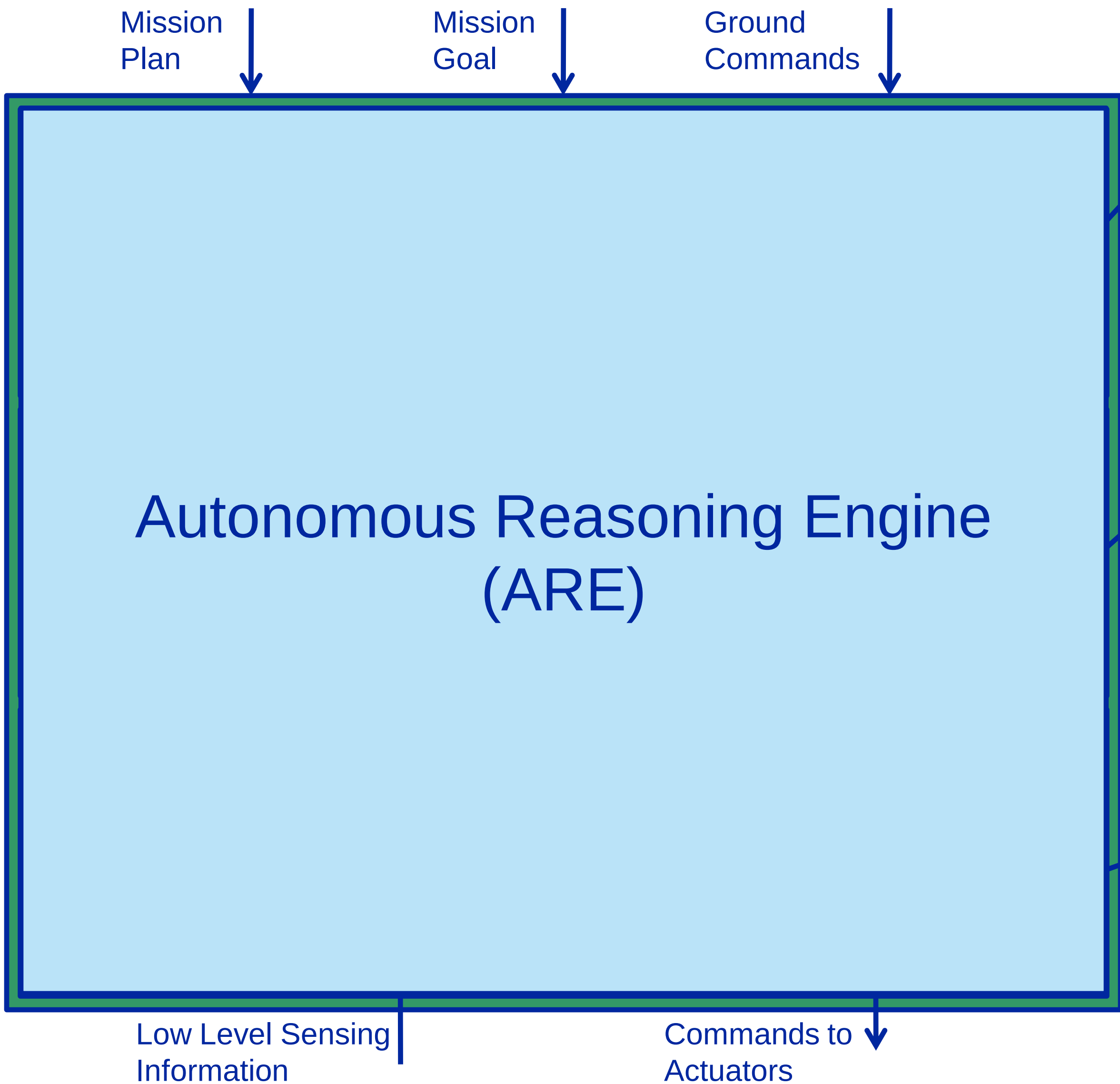


- Problem classification in terms of
  - Goal achievement
    - weak, strong, strong cyclic
  - Observability
    - full, partial, null
  - Structure of goals
    - assertions, temporally extended, knowledge goals
- Many techniques for planning with nondeterminism
  - Symbolic algorithms based on model checking primitives
    - strong post-image, …
  - BDDs to represent belief states

The OMC-ARE study

- Demonstrate the applicability of
  - model based reasoning, and
  - model checking techniques
- to increase autonomy of on-board reasoning
  - on-board re-planning
  - on-board plan validation
  - execution and monitoring
  - fault detection identification and recovery

The ARE Architecture



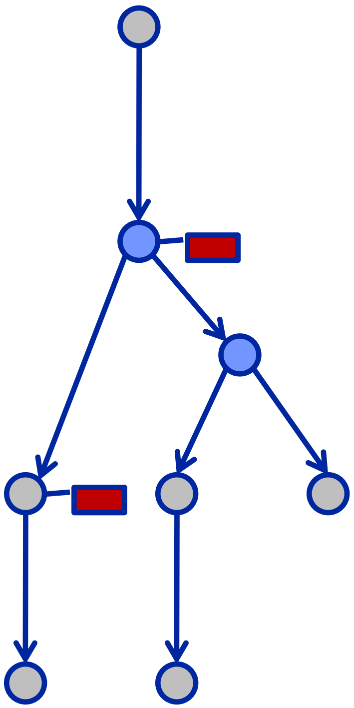
DELIBERATIVE LAYER: provides facilities for goal-driven planning, plan validation, and system-level FDIR.

EXECUTIVE LAYER: provides facilities to execute and monitor the correct execution of the current mission plan.

CONTROL LAYER: provides low level interactions with the controlled system (sensor acquisition and commands to actuators sending).

The OMCARE framework

- Why planning via symbolic model checking?
  - Deals with nondeterminism
  - Model validation
  - Same model on board and on ground
  - Reasoning about faults as model checking
  - Strong conditional plans
- Extension 1: assumption-based planning
  - Generate plans under suitable *Assumptions*
  - Resulting plans *annotated* with run-time checks (assertions)
    - Sufficient to detect if assumptions violated
- Extension 2: a simple model of resources
  - Actions extended with simple model of resources
    - Each action has estimate on minimal and maximal resource consumption
    - Each resource has lower and upper bound in a state
    - interval arithmetic
  - Used in plan validation and run-time monitoring
    - Planning not aware of resources
  - Built-in property checking
    - Each resource should not go below a certain minimum level R_{min}
    - Notions generalized to belief states
    - conservative approach, loses precision
  - Connection between logical framework and computation via estimators



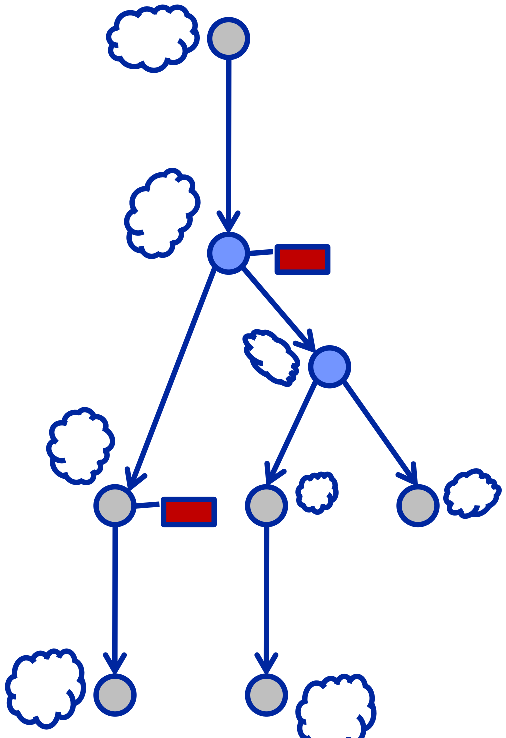
Reasoning in belief space

- Models *run-time uncertainty* on controlled plant status
  - resulting from partial observability
    - e.g. faults may not be directly observable
  - several states compatible with currently available information
  - indistinguishable states collected into a *belief state*
- Action in a belief state
  - applicability conditions must hold in all states
  - result belief state is set of all possible successors
- Observations “split” belief states
  - refine belief states to the states compatible with observation

Validation of given plan

- Ensure properties of given plan
  - Ensure action applicability
  - Ensure planning-time assumptions
  - Resources within limits

- Algorithm based on progression of belief states
  - Associate belief state and resources to each control point in plan tree
  - Belief states must satisfy annotations (assertions)
  - Success if final belief state included in the goal
  - Resources progressed and compared w.r.t. R_{min}



Strong planning under partial observability

- Forward And/Or search in belief space
  - Node Expansion
    - OR branching
      - simulate effect of action execution
    - AND branching
      - simulate effect of observation
  - Nodes tagged as
    - success if contained in goal, or if descendent success
    - failure if no action possible or all descendants are failure due to loopbacks

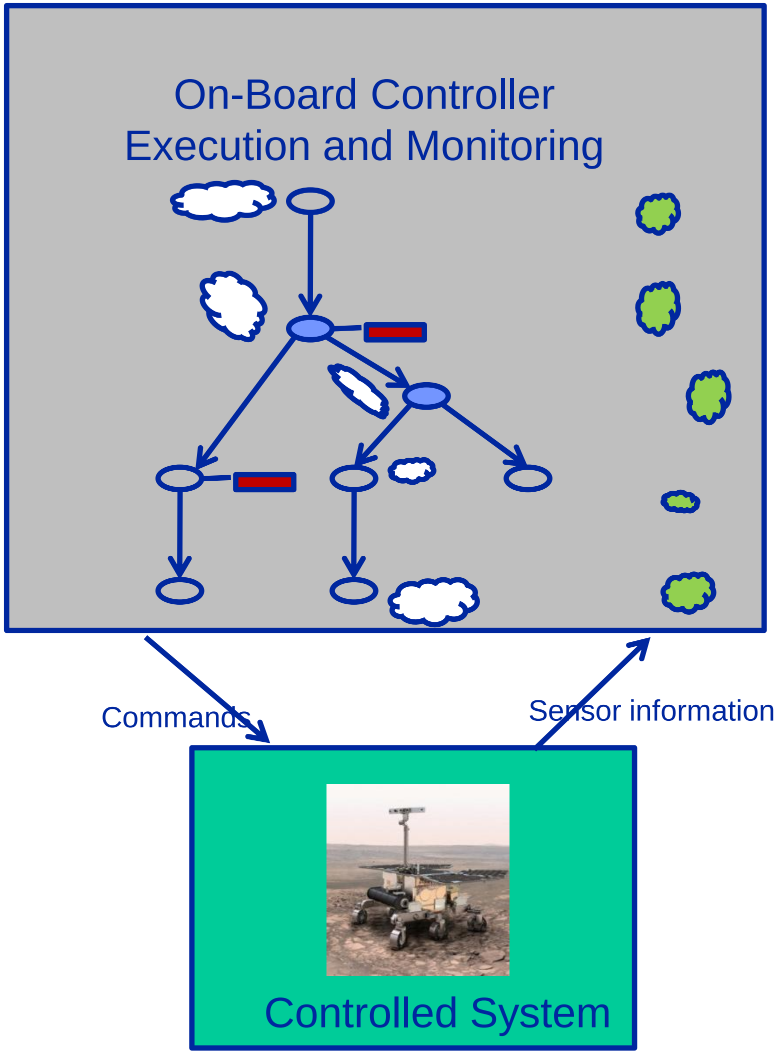
- To deal with assumptions
  - Beliefs pruned according to assumptions
  - Progress two “monitor-beliefs”
    - represent uncertainty w.r.t. status of assumption satisfaction
  - Prune monitor-beliefs using sensing, until no more uncertainty

- Heuristic search
  - General, domain-independent heuristic guidance used

- Resource consumption currently disregarded during planning
  - could be used to prune resource-inconsistent branches

Run-time execution and monitoring

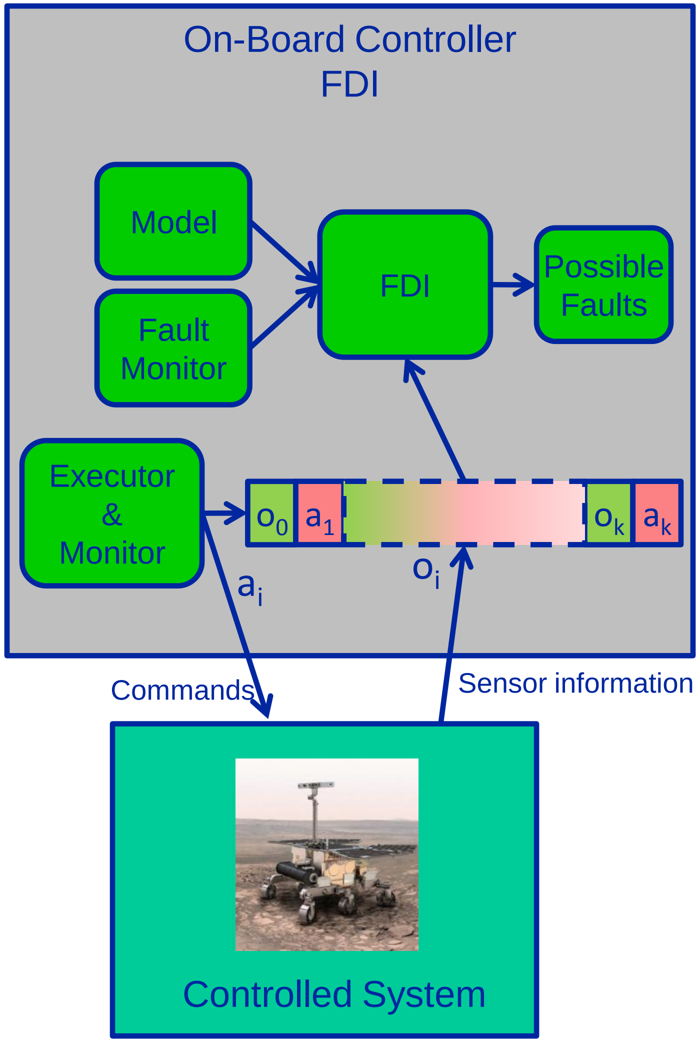
- Progress belief state and resources w.r.t. plan structure
  - Comparison of belief state read from sensors with:
    - » progressed belief state
    - » annotation in the plan
  - Comparison of expected resources w.r.t.
    - » resources from sensors
    - » minimal resource R_{min}



FONDAZIONE BRUNO KESSLER

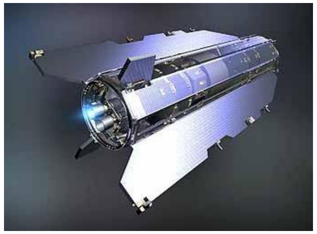
Fault Detection, Identification, Recovery

- Fault detection and identification via re-use of techniques developed in formal safety analysis for the extraction of fault-trees
- Record performed actions and observations while executing plan
  - Bounded History Window
- Construction of a monitor for fault variables
- Cross-product of monitor and Model of the plant
- Simulation of the History Window on the cross-product model
  - Accumulate reachable states of the cross-product
  - Project on fault monitor variables
  - Analyze the resulting set to extract the possible faults
  - For multiple faults, consider the one with highest probability
- Remark: FDIR not on-line



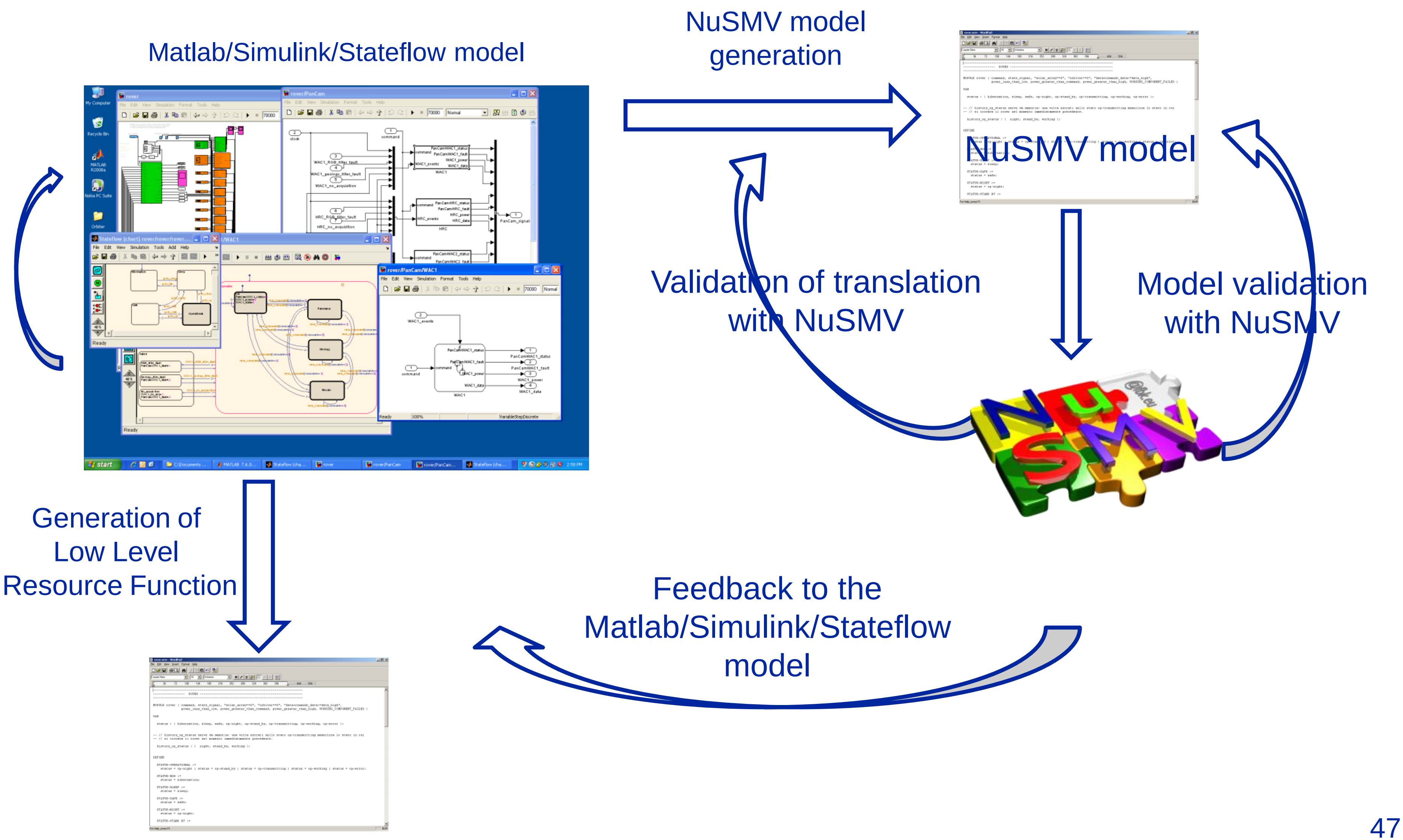
OMCARE: experimental evaluation

- Implemented framework within the NuSMV model checker using BDD techniques
- Integrated on a realistic spacecraft simulator
  - Including hw, sw, environment
- Case studies
  - Planetary rover
    - » Model taken from another running project developed in Thales-Alenia Space
  - Orbiting spacecraft
    - » Thales-Alenia Space in house simple model



- Characterization
  - Functional – on desktop PC under Linux
  - Embedded – on platforms RTEMS (LEON3, ERC32) and OSTRALES (ERC32)

Model: generation and validation

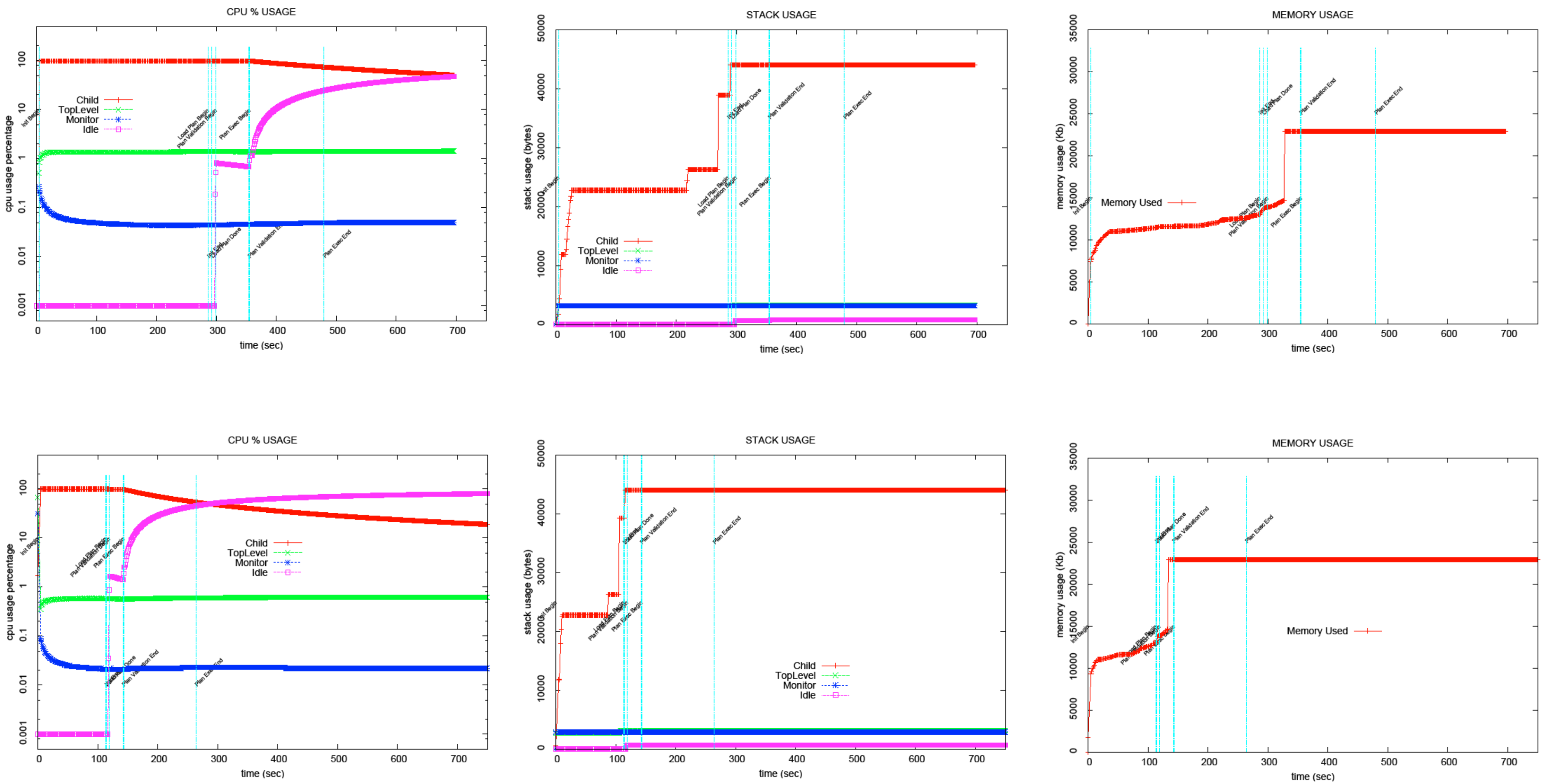


Embedded characterization

	ROVER				ORBITER	
	SMALL		FULL			
	ERC32	LEON3	ERC32	LEON3	ERC32	LEON3
Initialization	33	13	282	113	9	1
Plan loading	3	1	6	2	2	0.5
Plan validation	15	6.5	55	23	1	1
Plan execution	116	121	125	121	16	16
Plan generation	87	34	1349	540	6	2

Time in secs

ERC32, LEON3 (rover)



More info about OMC-ARE: http://es.fbk.eu/projects/esa_omc-are

See also IJCAI11 paper.

Lessons learnt

- *Unique formal framework* for all autonomy functionalities
- *Enables for formal validation* of the model using model checking techniques
- *Same framework* for on-board and on-ground reasoning
- Promising, non-trivial effort in technology transfer

Structure of the talk

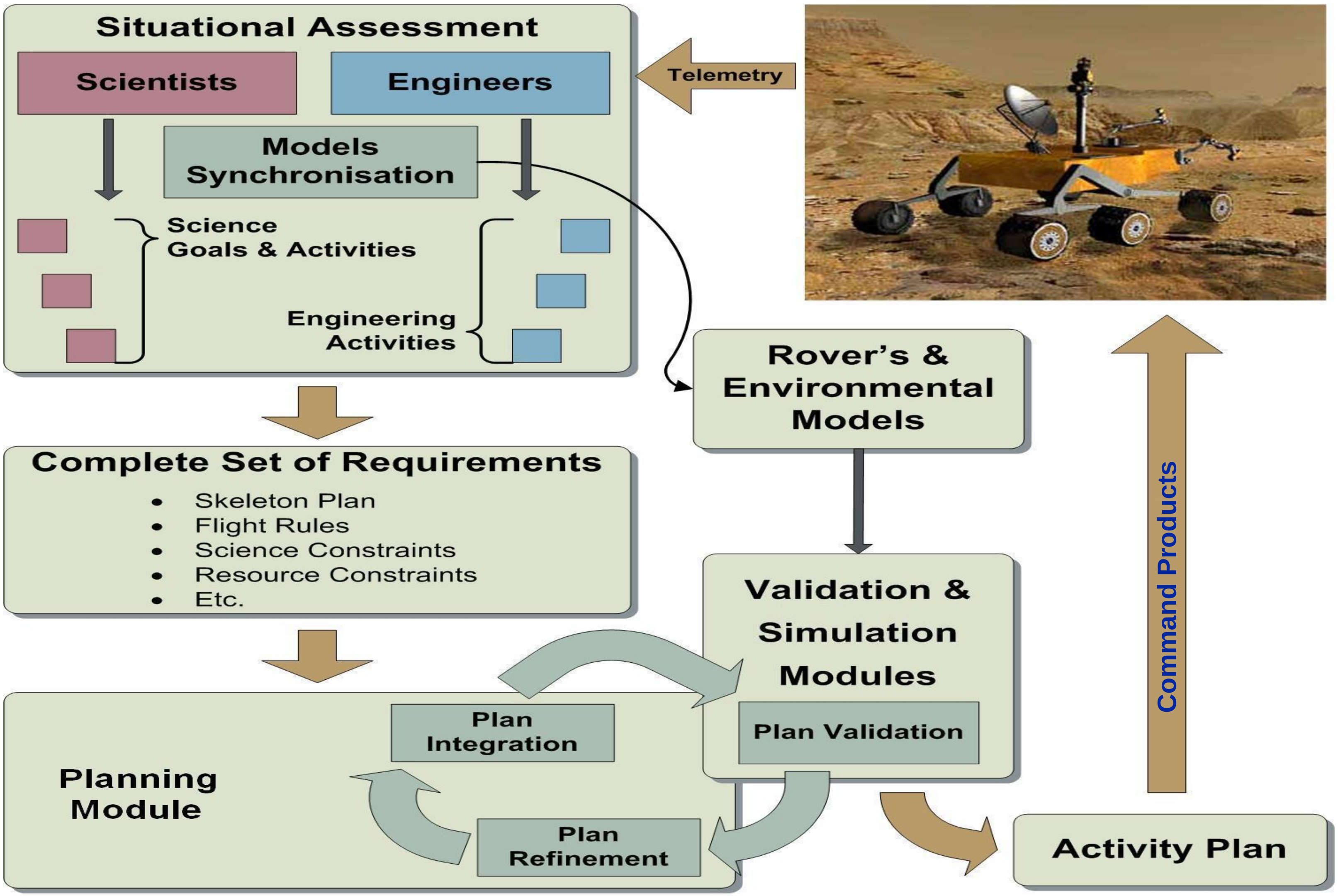
- Motivations
- Support for design activities
  - The COMPASS project
- Support for operation activities
  - Discrete case
    - » The OMCARE project
  - Continuous case
    - » The IRONCAP project
- Conclusions

IRONCAP

- *Preparing ESA for future robotics missions operations through the Investigation and Prototyping of Innovative Planning Operations Concepts for Rovers equipped with Autonomy Capabilities*

- Goals:
  - developing an operational concept for autonomous Rovers and define the processes and tools required for Rover ground control.
  - developing a prototype of a Rover planning and scheduling facility supporting the operational concept
  - demonstrating and evaluating the prototype in the context of two case studies

Workflow & Operations Planning Cycle



Functionalities

- Validation and Verification
  - Model V&V: does our domain model capture the expected behaviours
  - Plan V&V: does given plan achieve the expected conditions
- Planning and Scheduling
  - Find plan such that expected conditions are (always) met
    - Goal representation capabilities
    - Temporally extended goals
    - Resource-aware goals
    - Hard & soft goals
    - Hierarchical, mixed initiative goals
    - Constraints and assumptions
- Model Synchronization
  - Ensure consistency between on-ground reasoning and on-board

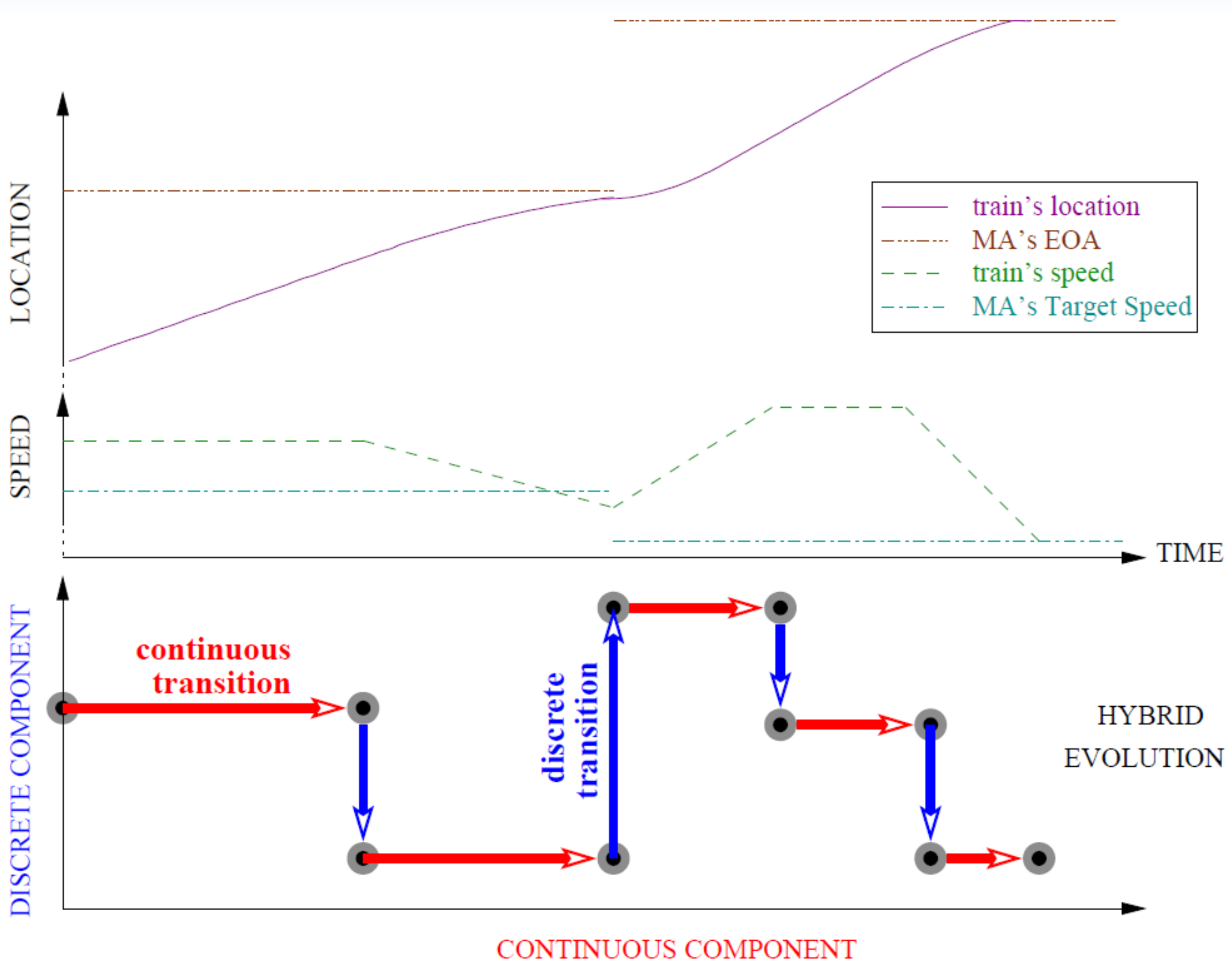
Expressiveness…

- We have a clean formalism to represent the controlled system and its environment
  - Nondeterministic action effects
  - Faults
  - Observations

- Missing ingredients:
  - Parallel actions
    - Start actuations in different subsystems
  - Time
    - Time taken by procedures
    - e.g. drilling, transmission, locomotion, power-up, …
  - Key issue: Resources
    - Power consumption, bandwidth, memory, …

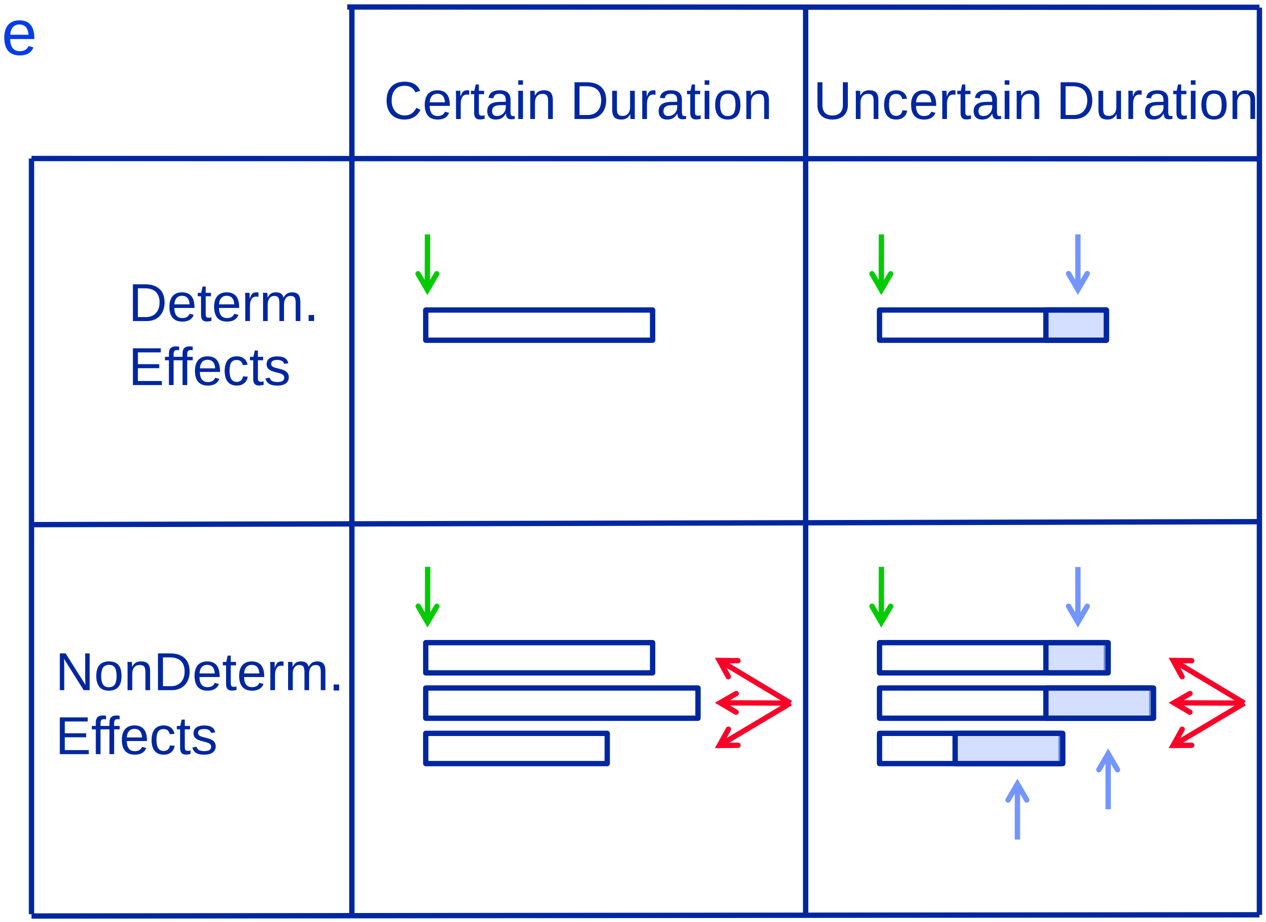
- Need for a richer formalism!

Hybrid evolution



Nondeterminism and uncertainty

- Nondeterminism
  - Discrete choice
- Uncertainty
  - Continuous
- Controllable
  - Start
- Uncontrollable
  - Effects
  - End



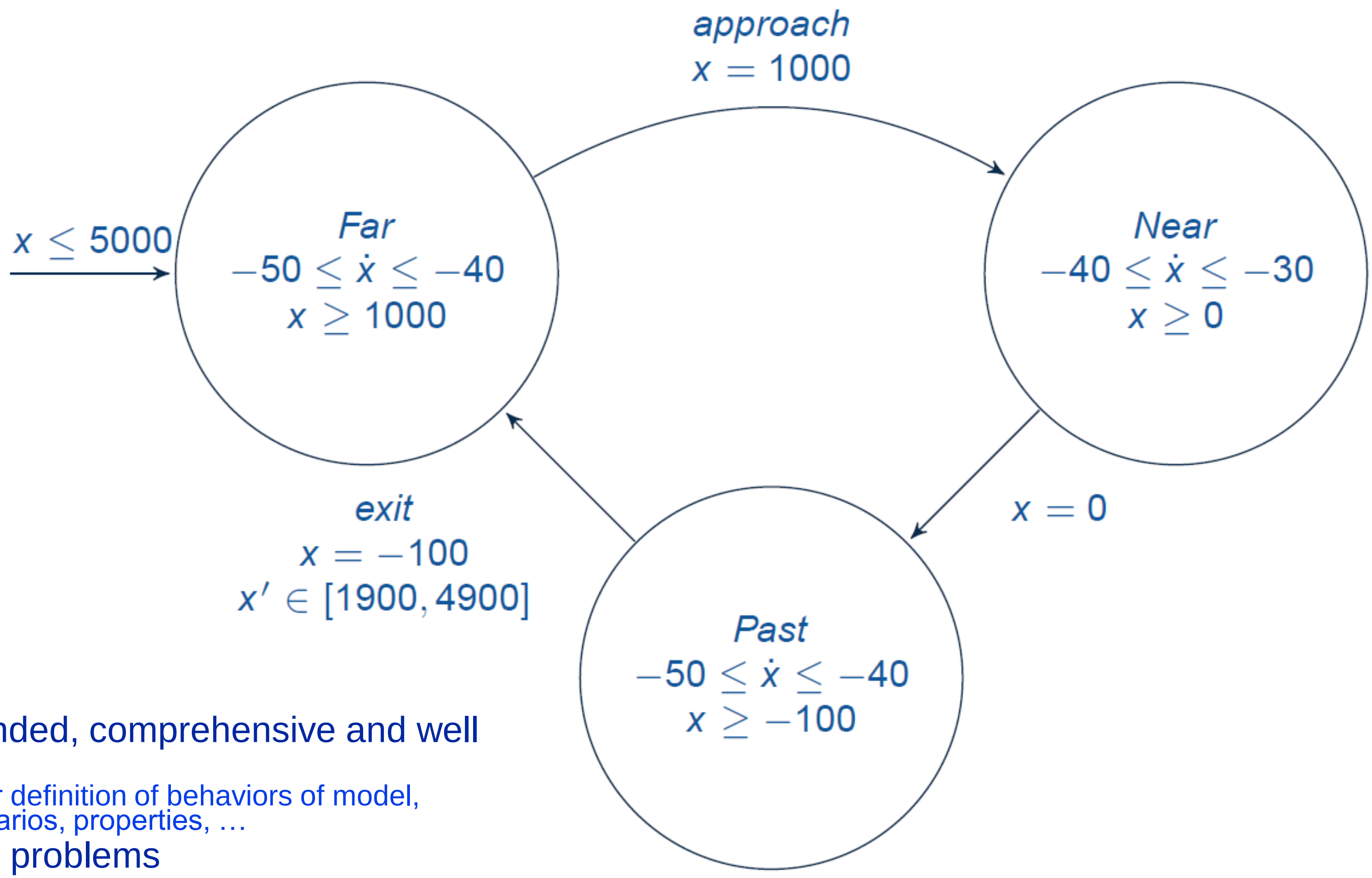
Structure of plans

- Sequences of actions
  - a_1 ; a_2 ; … ; a_n

- Time-triggered sequences of actions
  - @t_1 do a_1 ; @t_2 do a_2 ; … ; @t_n do a_n
- Time-triggered sequences of actions and checks
  - @t_1 do a_1 ; @$t_{1'}$ assert C_1 ;
    @t_2 do a_2 ; @$t_{2'}$ assert C_2 ;
    … ;
    @t_n do a_n
- Time-dependent sequences/conditionals
  - @t_1 if C_1 then do a_1 else do a_2 ;
    @t_2 assert C_2

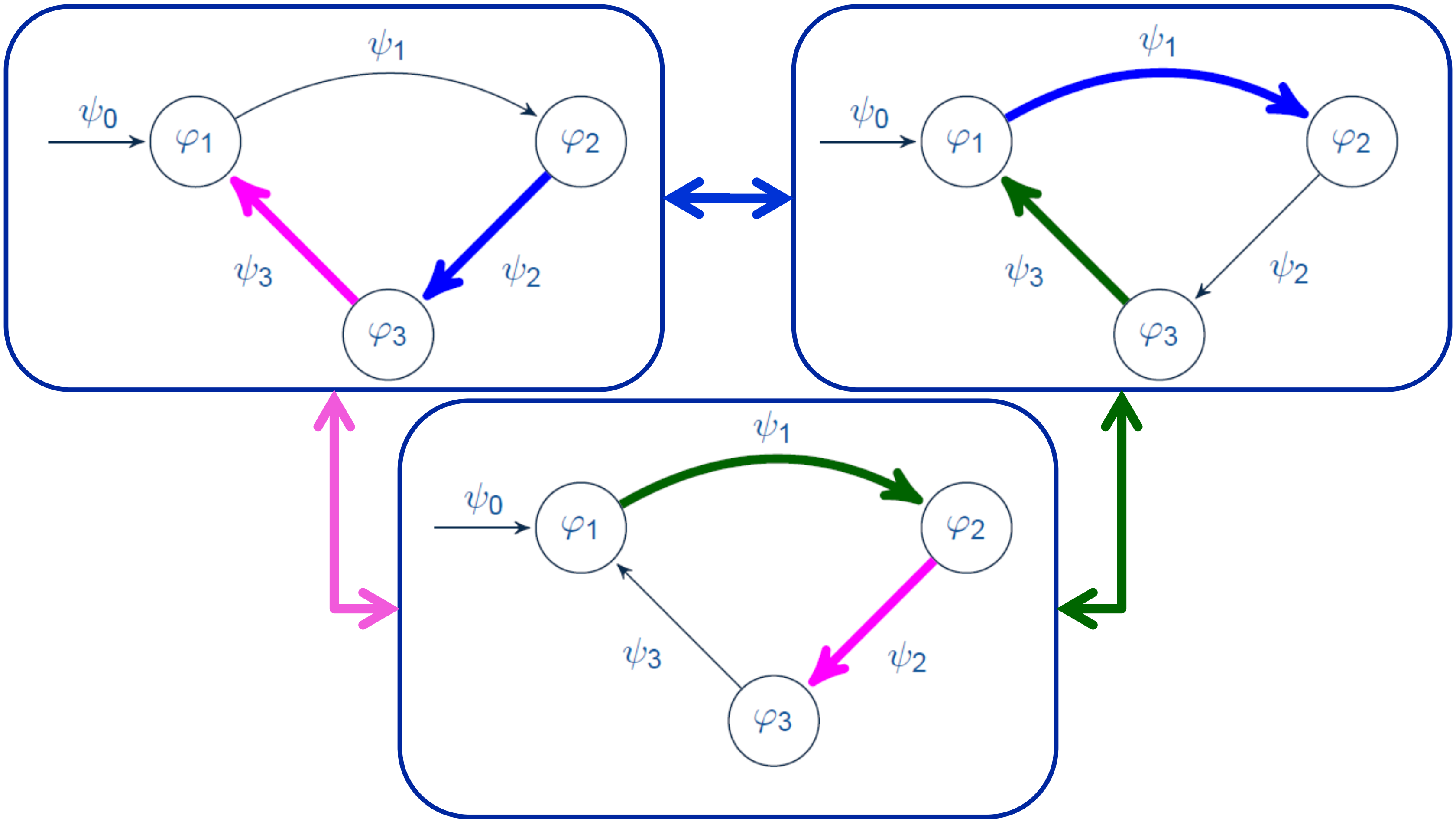
- Arbitrarily complex programming language…
- Possibly extended with embedded subgoal delegation…

The formalism: hybrid automata



- Well founded, comprehensive and well studied
  - Clear definition of behaviors of model, scenarios, properties, …
- Relevant problems
  - Model checking
  - Temporal problems (also with uncertainty)
  - Timed games

Networks of hybrid automata



SMT-based Model Checking for Hybrid Automata

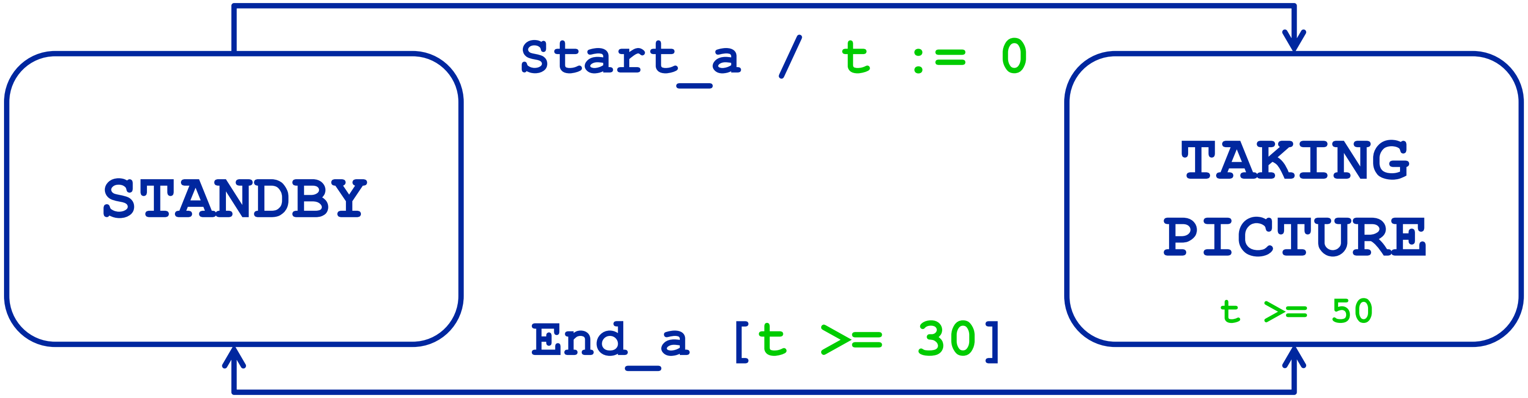
- Symbolic representation
  - I(X), R(X, X') are now first-order formulae
  - Boolean for discrete, real-valued for timing/continuous
- From SAT-based to SMT-based model checking
  - I(X), R(X, X') are now first-order formulae
  - bounded model checking, induction, abstraction/refinement, …
- The enabler: Satisfiability Modulo Theory
  - Richer language, decidable fragments of first order logic
  - E.g. theory of uninterpreted functions, linear integer arithmetics, ...
- SMT solvers
  - Tight integration of Boolean reasoning and constraint solving
  - SAT solver for boolean reasoning
  - theory solvers to interpret numerical constraints
- SMT community
  - Language and benchmarks: **`http://www.smt-lib.org`**
  - Yearly competition: **`http://www.smt-comp.org`**
  - MathSAT (our solver): **`http://mathsat.fbk.eu`**
- A MathSAT-based extension of NuSMV forthcoming

An example

```
Start_a -> s = STANDBY
Start_a -> next(s) = TAKING_PICTURE
Start_a -> next(t) = 0.0

s = TAKING_PICTURE -> t <= 50.0

End_a -> s = TAKING_PICTURE
End_a -> next(s) = TAKING_PICTURE
End_a -> t >= 30.0
```



Structure of the talk

- Motivations
- Support for design activities
  - The COMPASS project
- Support for operation activities
  - Discrete case
    - The OMCARE project
  - Continuous case
    - The IRONCAP project
- Conclusions

Conclusions

- A Symbolic Model Checking approach to autonomous systems
- A comprehensive formal modeling framework
  - Expressiveness of the model
    - Non-determinism, faults, partial observability, resources
  - Encompassing different autonomy functions
    - Requirements analysis, functional correctness, safety dependability assessment
    - Model validation, plan generation, plan validation, monitoring, execution and FDIR
- Strong support tools
  - On ground
    - Validation on realistic case studies
  - On board "not completely crazy"
    - Operational characterization within spacecraft simulators

Take-away messages

- Planning as the tip of the iceberg
  - Need to put planning into broader (lifecycle) perspective
    - Links to design phase and operation phase
- The role of design languages
  - Domain description languages come from design phase
  - Similar to tech transfer in formal verification
    - adapt method to already adopted language
    - no way to model rover with PDDL
    - but maybe we can extract PDDL from FSM's
- The role of symbolic representations
  - "Model everything as one gigantic automaton? I don't think so…"
  - Well studied composition primitives
  - Structure may also help partitioning verification

Take-away messages

- A model-based approach, models become critical
  - Need for model validation
    - Automatically constructed structural properties
    - Need for model validation
  - Model-to-model management
    - Proving equivalence after simplification
    - Different levels of abstraction (checking refinement)
    - Ground to board and back
- Mixed initiative, what-if?
  - Plan validation
    - Formal validation
    - Simulation-based validation
    - Their combination!
- Ground to on-board consistency
  - Model synchronization
    - Update conditions on ground after execution
    - Retrieve information from telemetry, re-execute and reconstruct (abduction needed?)
  - Model update
    - Revision of ground model based on inconsistencies wrt telemetry
      - E.g. faults detected, mis-estimated parameters, degradation due to use, …
    - Revision of on-board model

Open issues and future directions

- Synthesis of FDIR modules
  - The AutoGEF project
- Improving scalability of hybrid systems verification
  - Exploit structure of the problem
    - scenario-based validation
  - Tighten connection between planning and temporal reasoning
    - SMT-based scheduling
- Diagnosability checking and synthesis
  - Automated synthesis of sensors configurations that guarantee diagnosability
  - Generalize to the case of hybrid automata
- Towards validation of intelligence
  - Proof of correctness of the conceptual framework
  - Validation of the reasoning engine software
    - "translation validation" approach
    - independent checking of generated plan
  - See also “tool qualification problem” in FV

SAT 2012 in Trento, June 17-20

http://sat2012.fbk.eu/