PALS: Virtual Synchrony for Cyber-Physical Systems

Peter Ölveczky

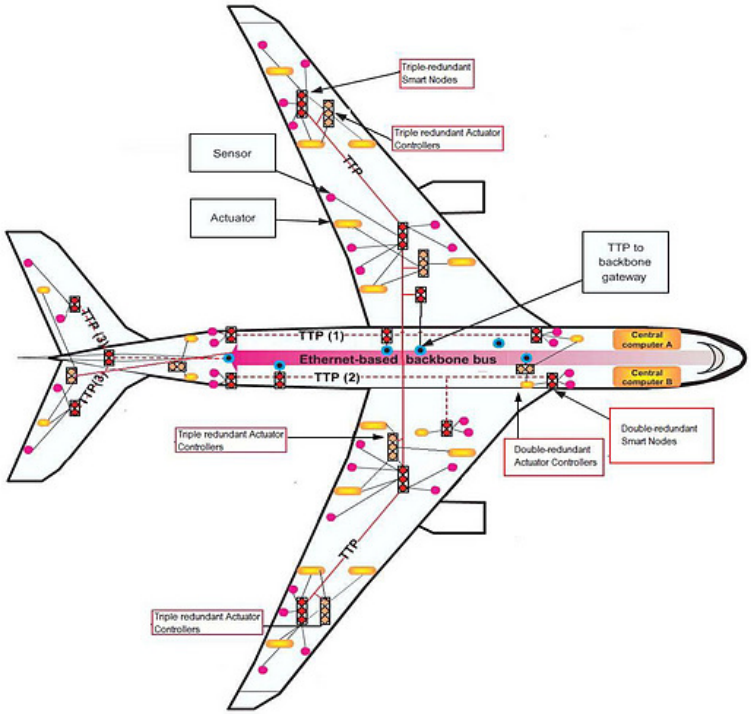
University of Oslo

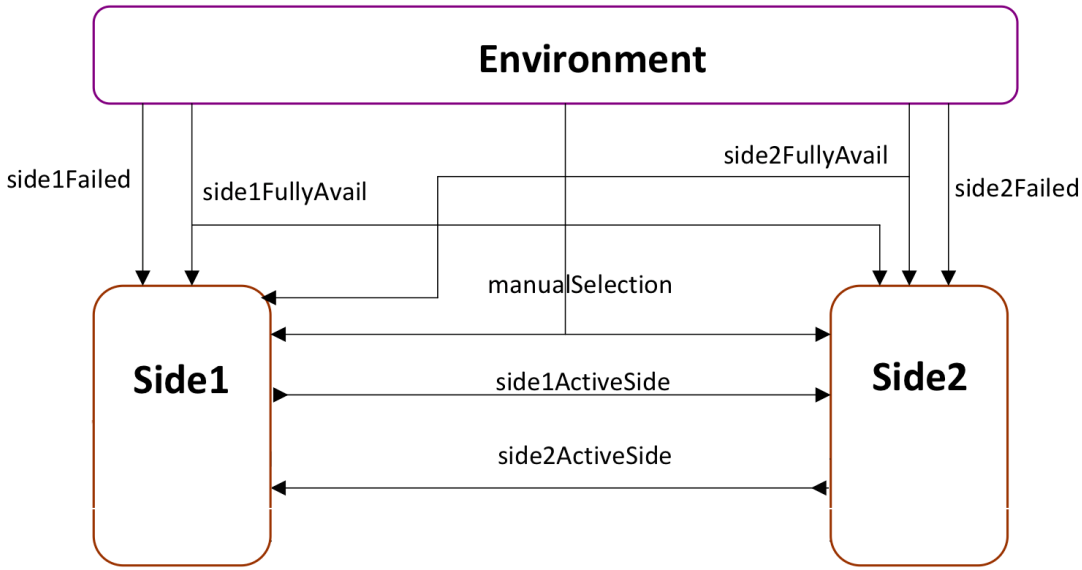
IFIP WG 1.3, Berlin, September 4, 2017

Based on work with Lui Sha and José Meseguer (UIUC); Kyungmin Bae (POSTECH); Steve Miller and Darren Cofer (Rockwell Collins); and others

Motivation: Which Cabinet is Active?

Rockwell Collins



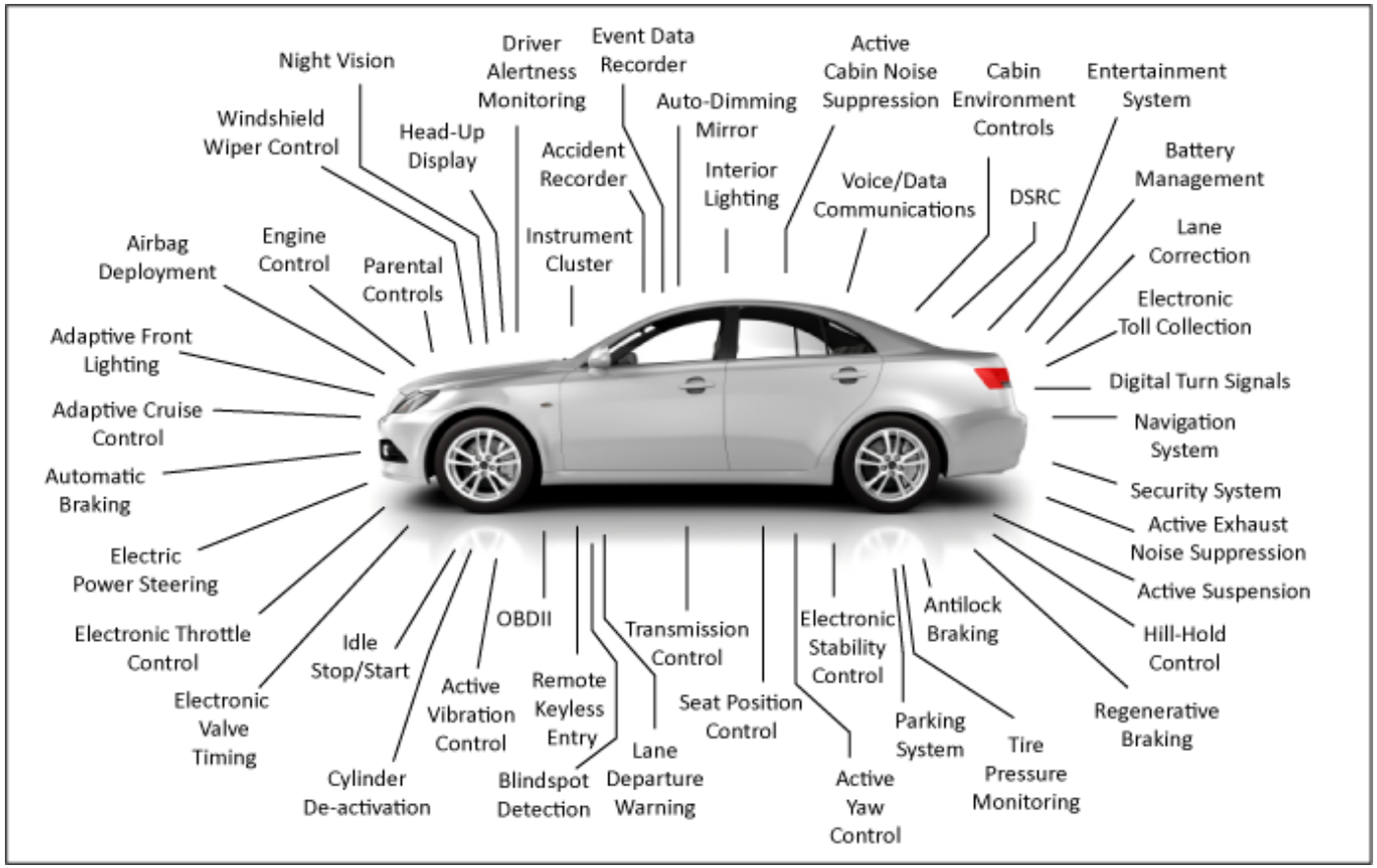


Motivation (II)

- Hard to design
  - race conditions, network delays, execution times, clock skews
- Hard to model check
  - state space explosion due to asynchrony
  - (impossible with explicit-state techniques?)

Virtually Synchronous CPSs

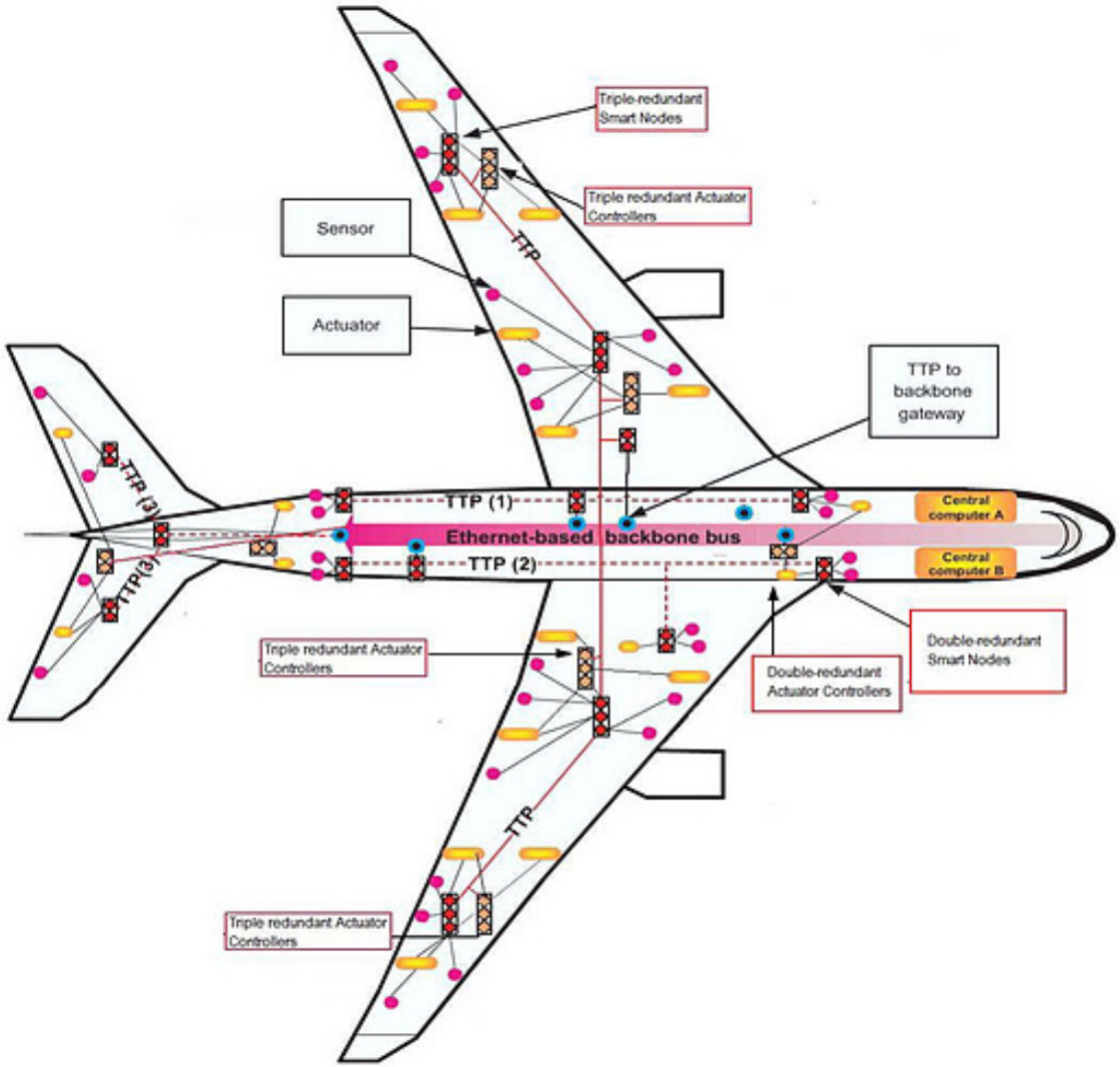
Many CPSs virtually synchronous



http://www.cvel.clemson.edu/auto/systems/auto-systems.html

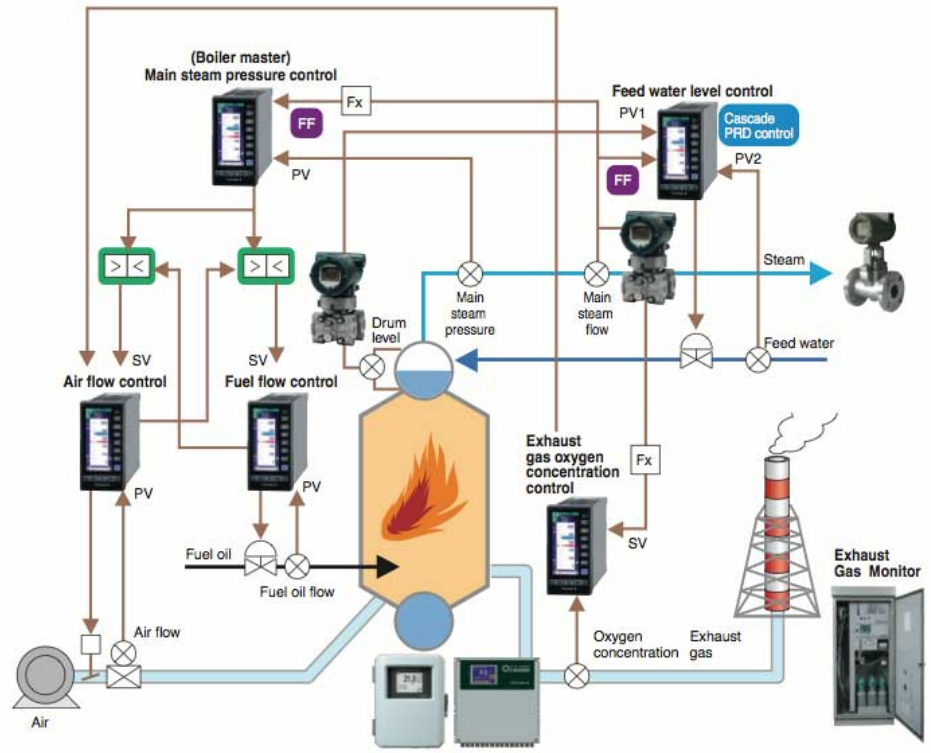
Virtually Synchronous CPSs

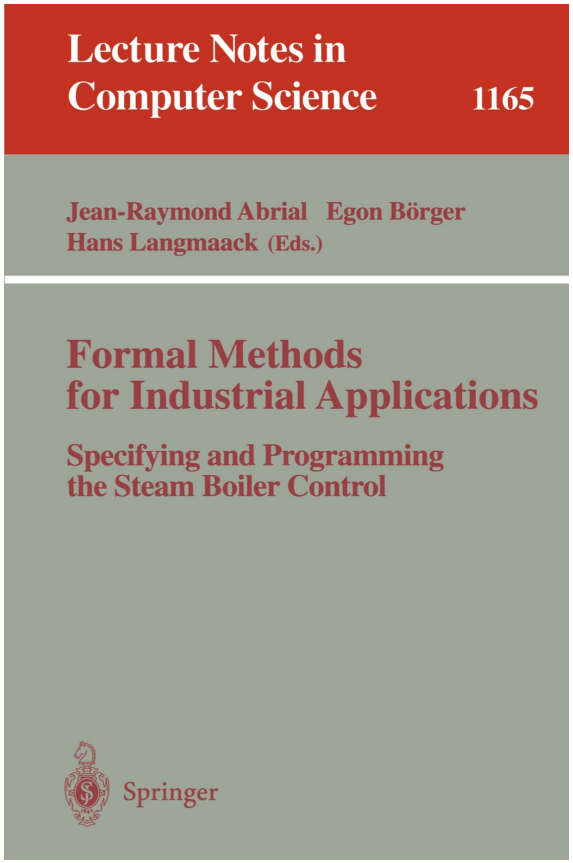
Many CPSs virtually synchronous



Virtually Synchronous CPSs

Many CPSs virtually synchronous





https://web-material3.yokogawa.com/image_8434.jpg

CPSs

- Time synchronization well understood (IEEE 1588, etc.)
- Bounded network delays

Formal Patterns [Meseguer, Sha, ...]

Formal patterns:

- Formalized and verified "design patterns"
- P a theory transformation

$$\langle \text{theory}, \text{params} \rangle \mapsto P(\text{theory}, \text{params})$$

PALS Formal Pattern [Cofer, Miller, Sha et al; Meseguer & Ölveczky]

- PALS: "physically asynchronous, logically synchronous"

  reduce design and verification of a virtually synchronous CPS to its synchronous design

PALS Formal Pattern [Cofer, Miller, Sha et al; Meseguer & Ölveczky]

- PALS: "physically asynchronous, logically synchronous"

  reduce design and verification of a virtually synchronous CPS to its synchronous design

- Transformation $(SD, \Gamma) \rightarrow PALS(SD, \Gamma)$

  SD: synchronous design

  Γ: bounds on network delay, execution time, clock skew

  $PALS(SD, \Gamma)$: corresponding distributed asynchronous design

PALS Formal Pattern [Cofer, Miller, Sha et al; Meseguer & Ölveczky]

- PALS: "physically asynchronous, logically synchronous"

  reduce design and verification of a virtually synchronous CPS to its synchronous design

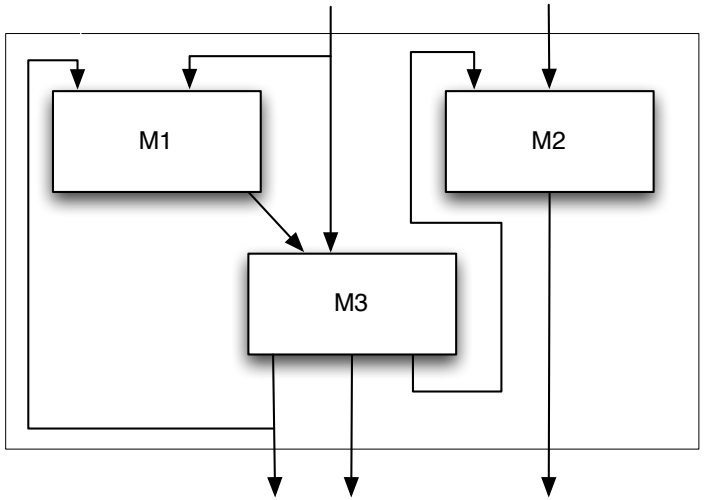
- Transformation $(SD, \Gamma) \rightarrow PALS(SD, \Gamma)$

- Correct by construction

$$SD \models \varphi \quad \text{if and only if} \quad PALS(SD, \Gamma) \models \varphi^*$$

PALS Details I: Synchronous Model

Ensemble *SD* of state machines



- all machines perform a transition in lockstep

PALS Details II: Asynchronous Model

Distributed implementation $PALS(SD, \Gamma)$ adds a "wrapper" around each state machine, with

- input buffer stores messages arrived during the round
- output buffer holds outgoing messages until they can be sent

Formalized in Real-Time Maude

PALS Details III: Some Assumptions

- External clock synchronization
  - difference between "local clock" time and "real" (global) clock time is always less than ϵ
- Local clock: $c_j : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$
  - monotonic and piecewise continuous
  - $|c_j(x) - x| < \epsilon$ for all "real" times x
- Time for (processing input + executing transition + generating output) $\in [\alpha_{min}, \alpha_{max}]$ with $0 \leq \alpha_{min} \leq \alpha_{max}$
- Message transmission time $\in [\mu_{min}, \mu_{max}]$ with $0 \leq \mu_{min} \leq \mu_{max}$

PALS Details IV: Optimal PALS Period

The smallest possible period T is

$$\mu_{max} + 2 \cdot \epsilon + \max(2 \cdot \epsilon - \mu_{min}, \alpha_{max})$$

PALS Details V: Correctness I

- A state in $PALS(SD, \Gamma)$ is stable iff all input buffers are full, all output buffers are empty, and there are no messages in transit
- The function *sync* maps stable states in $PALS(SD, \Gamma)$ to states in SD
- Can define Kripke structures (SD_{c_e}, L) for SD (with environment constraint c_e) and $(PALS(SD, \Gamma), L')$ in the expected way

PALS Details VI: Main Correctness Result

Theorem

Given a formula $\varphi \in CTL^*(AP)$, *and a state predicate* $stable \notin AP$ *characterizing stable states, there is a formula* $\varphi_{stable} \in CTL^* \setminus \{\bigcirc\}(AP \cup \{stable\})$ *defined as follows:*

$$
\begin{array}{rcl}
a_{stable} & = & a, \text{ for } a \in AP \\
(\neg\,\varphi)_{stable} & = & \neg\,(\varphi_{stable}) \\
(\varphi_1 \wedge \varphi_2)_{stable} & = & \varphi 1_{stable} \wedge \varphi 2_{stable} \\
(\varphi_1\ U\ \varphi_2)_{stable} & = & (stable \rightarrow \varphi 1_{stable})\ U\ (stable \wedge \varphi 2_{stable}) \\
(\bigcirc\,\varphi)_{stable} & = & stable\ U\ (\neg stable \wedge (\neg stable\ U\ (stable \wedge \varphi_{stable}))) \\
(\forall\,\varphi)_{stable} & = & \forall\,\varphi_{stable}
\end{array}
$$

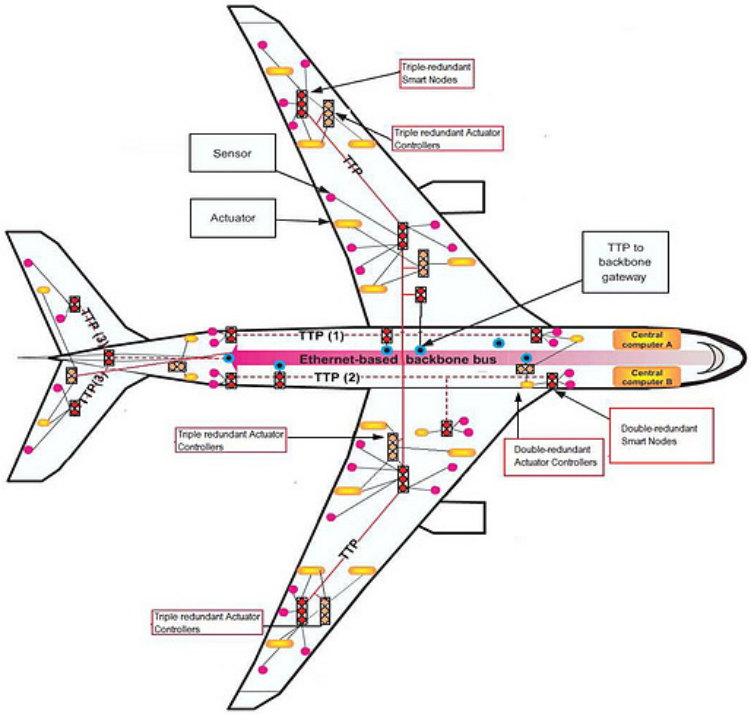
such that for each reachable stable state s *in* $PALS(SD, \Gamma)$ *we have*

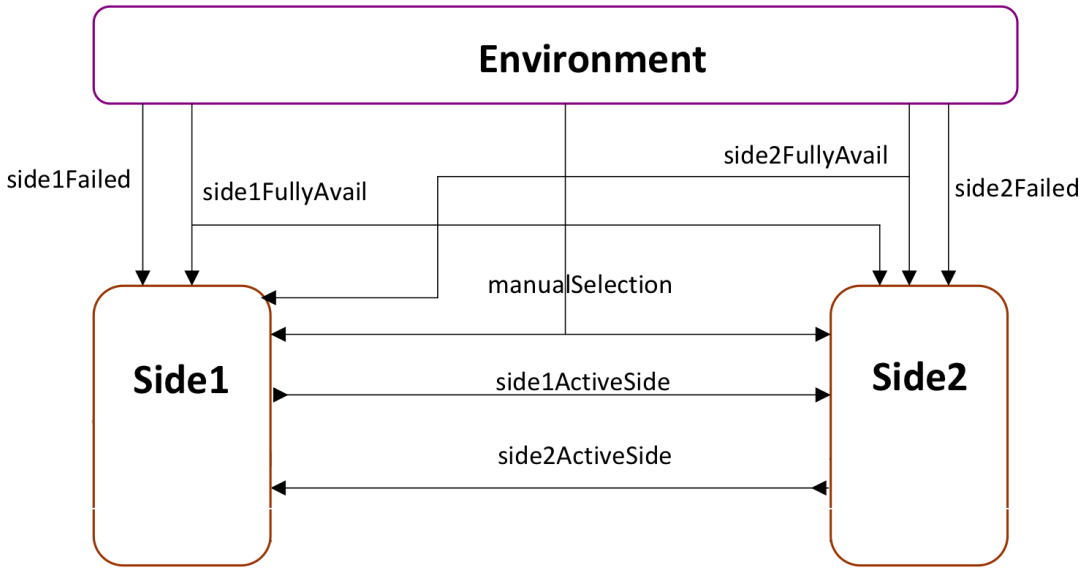
$$(PALS(SD, \Gamma), L'), s \models \varphi_{stable} \iff (SD_{c_e}, L), sync(s) \models \varphi,$$

where $L' : \mathcal{T}_{PALS(SD,\Gamma)_{\texttt{GlobalSystem}}} \rightarrow \mathcal{P}(AP \cup \{stable\})$ *is a labeling function satisfying* $L'(s) = L(sync(s)) \cup \{stable\}$ *when* s *is a stable state, and* $stable \notin L'(s)$ *otherwise.*

Case Study I: Which Cabinet is Active?

Rockwell Collins





Case Study II: Requirements of Active Standby

R_1: Both sides should agree on which side is active (provided neither side has failed, the availability of a side has not changed, and the pilot has not made a manual selection).

R_2: A side that is not fully available should not be the active side if the other side is fully available (again, provided neither side has failed, the availability of a side has not changed, and the pilot has not made a manual selection).

R_3: The pilot can always change the active side (except if a side is failed or the availability of a side has changed).

R_4: If a side is failed the other side should become active.

R_5: The active side should not change unless the availability of a side changes, the failed status of a side changes, or manual selection is selected by the pilot.

Case Study III: Analysis

Verified synchronous design using LTL model checking in Maude

- properties do not hold
- verified modified properties

Case Study IV: Comparison

Simplified asynchronous model in Real-Time Maude

- execution times 0; perfect clocks
  - no message delay
  - message delay 0 or 1

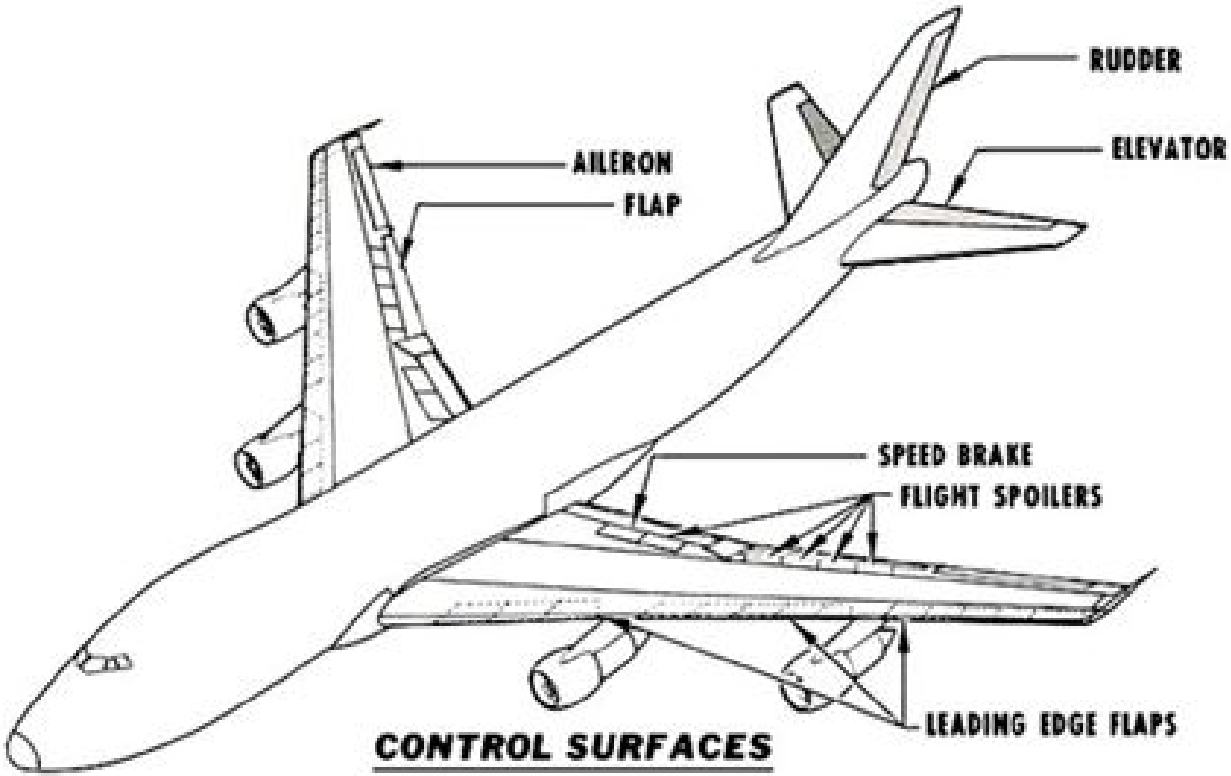
Model	Max.msg.dly	# states	ex.time
Synchr.	n/a	185	0.1 sec.
Asynchr.	0	3,047,832	1249 sec.
Asynchr.	1	aborted	

Multirate Systems [Bae, Meseguer, Ölveczky]

- Components may have different frequencies

Multirate Systems [Bae, Meseguer, Ölveczky]

- Components may have different frequencies
- Commercial airplane
  - aileron controllers 30-100 Hz
  - rudder controller 30-50 Hz
  - must synchronize to turn aircraft



Multirate PALS

Multirate PALS: extends PALS to multirate hierarchical control systems

- Controller period multiple of faster periods

Multirate PALS

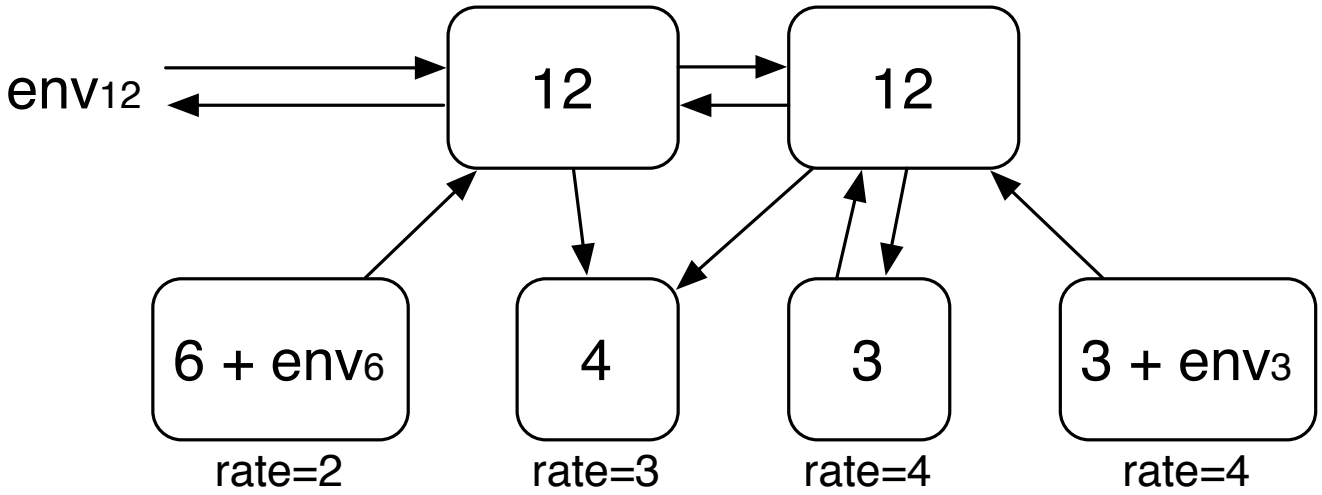
Multirate PALS: extends PALS to multirate hierarchical control systems

- Controller period multiple of faster periods
- Synchronous model:
  - all components must perform in lock-step
  - "slow down" fast components

Multirate PALS

Multirate PALS: extends PALS to multirate hierarchical control systems

- Controller period multiple of faster periods
- Synchronous model:
  - all components must perform in lock-step
  - "slow down" fast components



Details: Multirate PALS Synchronous Model

- Fast components perform k "internal transitions" in one step
  - reads/produces k-tuples of inputs/outputs
- Slow components read/produce single values

Details: Multirate PALS Synchronous Model

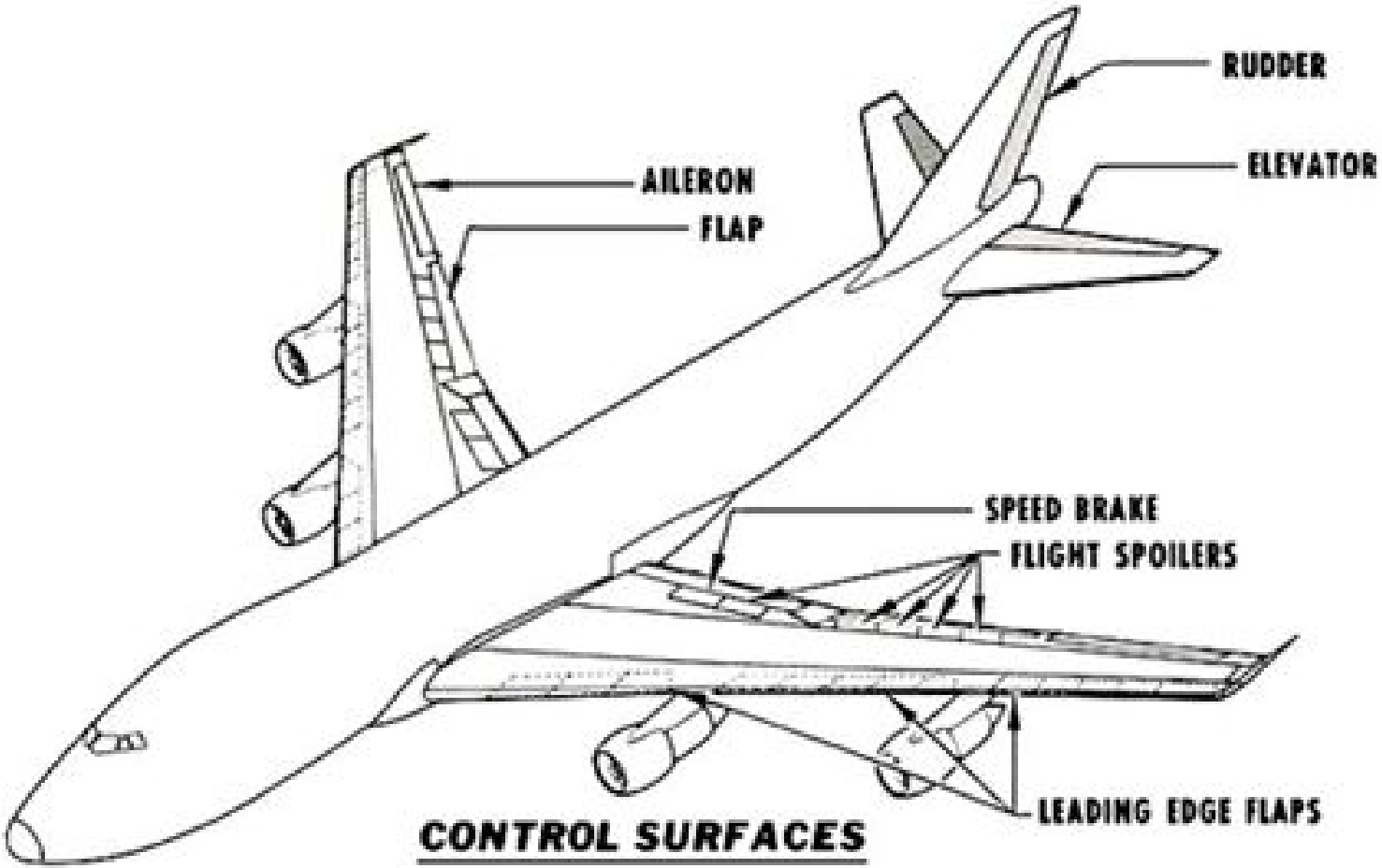
- Fast components perform k "internal transitions" in one step
  - reads/produces k-tuples of inputs/outputs
- Slow components read/produce single values
- Input adaptors transform k-tuples to/from single values

Multirate PALS II

Formalized multirate synchronous and asynchronous models

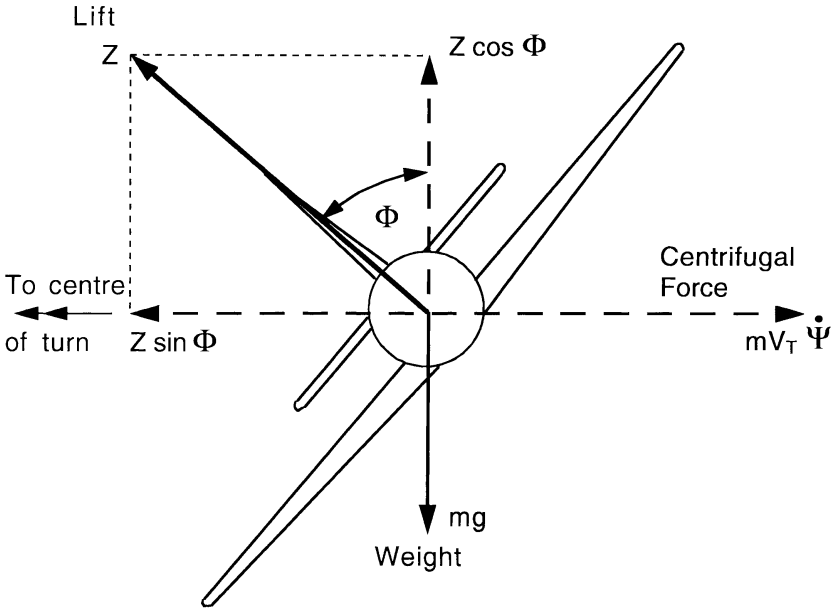
$$\text{synchronous design} \models \Phi$$
$$\text{iff}$$
$$(\text{“stable-state”})\ \text{asynchronous design} \models \Phi$$

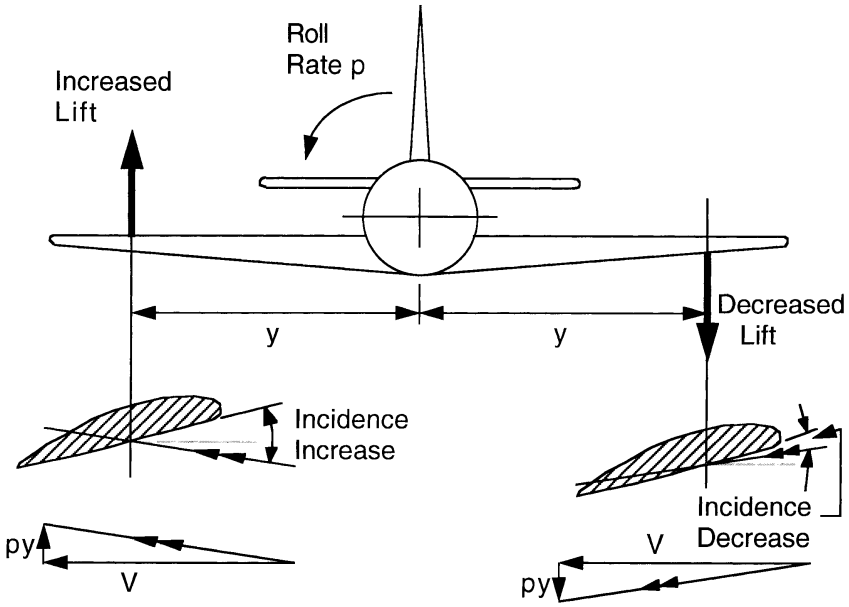
Case Study: Turning an Airplane [with J. Krisiloff]



Turning an Airplane (I)

- Move ailerons to roll airplane

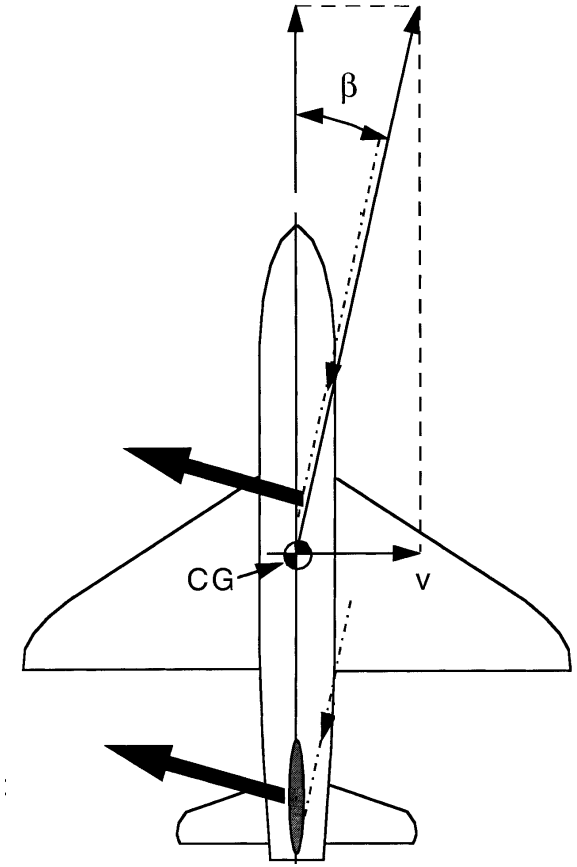




Turning an Airplane (II)

Rolling causes adverse yaw

- sideslip in wrong direction
- use rudder to avoid this



Case Study: Turning an Airplane (II)

Turning control algorithm:

Case Study: Turning an Airplane (II)

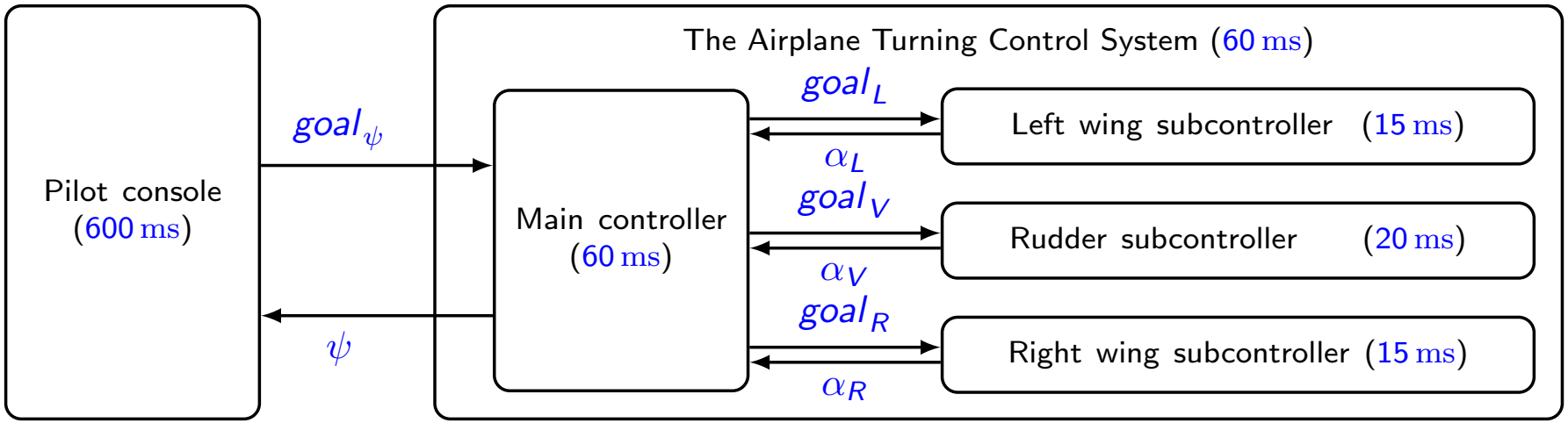
Turning control algorithm:

- Pilot selects direction

Case Study: Turning an Airplane (II)

Turning control algorithm:

- Pilot selects direction
- Controller moves ailerons and rudders to make optimal turn



Model Checking

- Pilot wants to turn plane $60°$
- Desired properties:
  - yaw angle always $< 1.0°$
  - goal direction reached within reasonable time
  - plane stable when goal direction reached

Analysis

- Real-Time Maude analysis: textbook algorithm does not achieve safe turn
- We designed and analyzed modified algorithm

Model Checking Performance (3000 ms bound)

- Synchronous model: 364 states
- Asynchronous model: 420,288 states
  - perfect clocks
  - no network delays

Model-Based Development with PALS [Bae, Ölveczky, Meseguer, Al-Nayeem]

Goal:

Make PALS design and verification methodology available to domain-specific modeling

Background: AADL

AADL: Industry standard for embedded systems modeling

- US Army, Honeywell, Airbus, Boeing, Dassault Aviation, EADS, ESA, Rockwell-Collins, Ford, Lockheed Martin, Raytheon, Toyota, U. S. Navy, ...
- OSATE: Eclipse plug-ins for AADL

Model-Based Development with PALS

Goal:

Use PALS design and verification methodology in domain-specific modeling

1. Model synchronous design *SD* in (Multirate) Synchronous AADL
2. Verify *SD* using SynchAADL2Maude OSATE plugin

Multirate Synchronous AADL Modeling Language

- Model synchronous designs
  - no need for hardware components

Multirate Synchronous AADL Modeling Language

- Model synchronous designs
  - no need for hardware components
- Language design choices:
  - use subset of AADL
  - constructs should have same meaning as in AADL
  - new property set `MR-SynchAADL` (rates, input adaptors, etc.)

Details: Multirate Synchronous AADL

- AADL subset:
  - hierarchical system, process, thread (and data) components
  - ports and connections
  - thread behavior in behavior annex
  - periodic dispatch
  - data ports
  - "delayed" connections
- `MR-SynchAADL` properties

Formal Semantics of Multirate Synchronous AADL

- Formal semantics in Real-Time Maude
  - simulation, reachability analysis
  - LTL and timed CTL model checking

The SynchAADL2Maude Tool

- OSATE/Eclipse plug-in for Multirate Synchronous AADL
- Checks if model valid Multirate Synchronous AADL model
- Real-Time Maude LTL model checking within OSATE
  - automatic synthesis of Real-Time Maude model
  - easy to define LTL formulas (XML, predefined propositions)
  - fairly intuitive counterexamples
- Predefined atomic propositions:

```
component name  |  boolean expression
```

Active standby verification in SynchAADL2Maude

Model	#States	Time
SynchAADL	203	0.6 s
Synch.	185	0.1 s
Asynch. (0)	3047832	1249 s
Asynch. (1)	n/a	n/a

Model Checking the Airplane Turn

Desired properties:

- yaw angle always $< 1.0°$
- goal direction (60) reached within reasonable time
- plane stable when goal direction reached

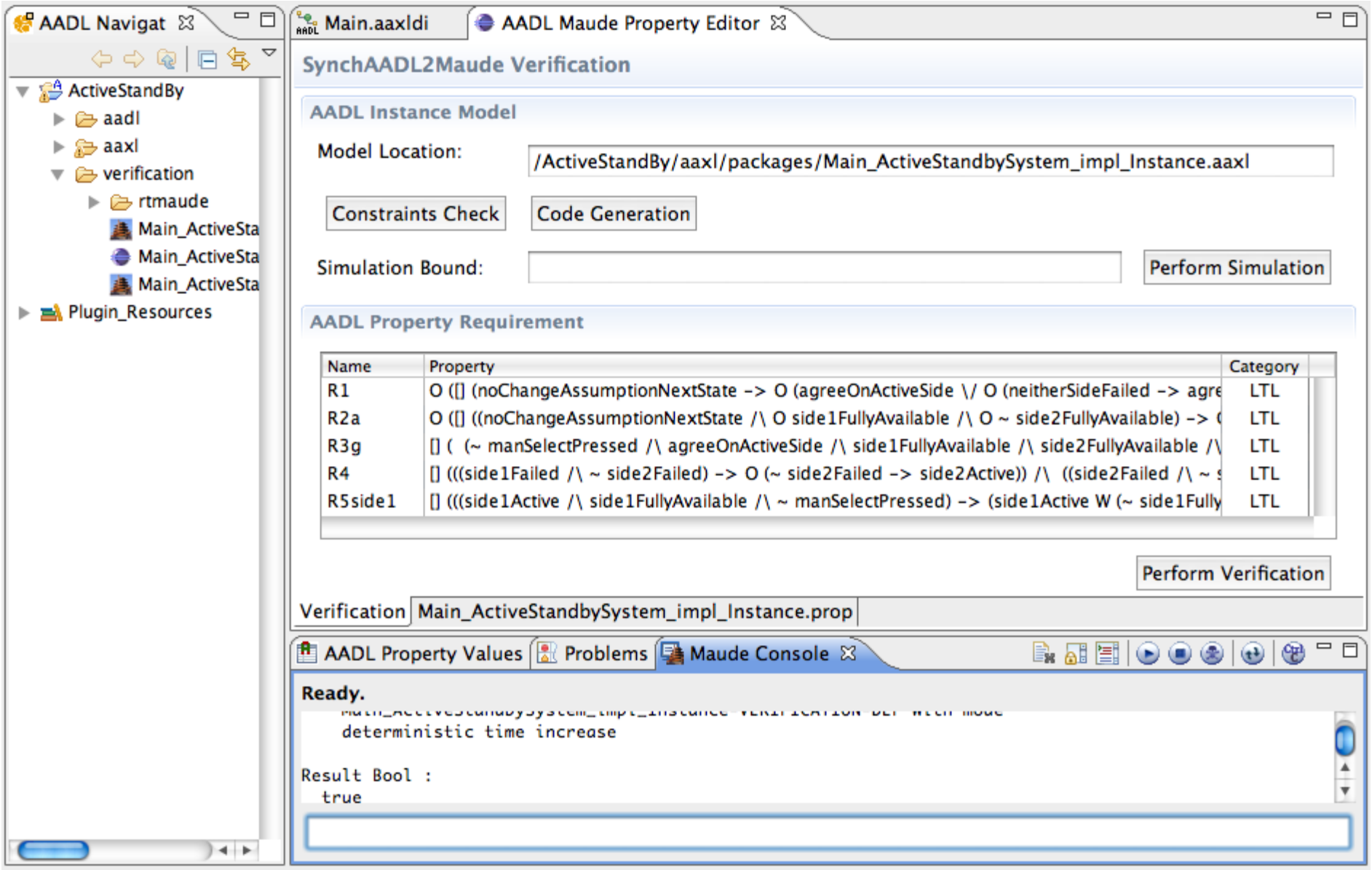
Model Checking the Airplane Turn

Desired properties:

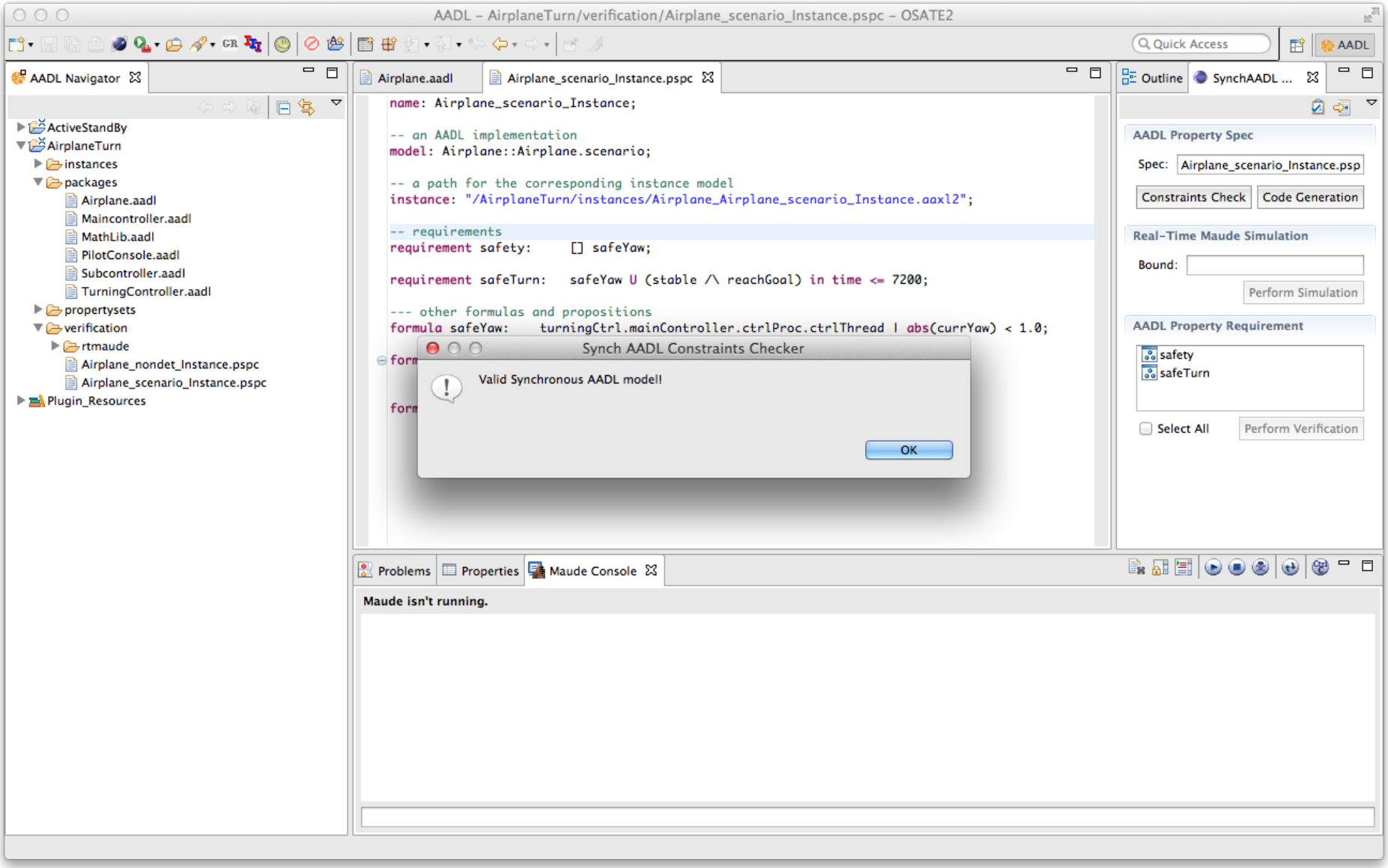
- yaw angle always $< 1.0°$
- goal direction (60) reached within reasonable time
- plane stable when goal direction reached

```
formula safeYaw:  turnCtrl.mainCtrl.ctrlProc.ctrlThread |
                      abs(currYaw) < 1.0;

requirement safety:  [] safeYaw;

requirement safeTurn:  safeYaw U (stable /\ reachGoal) in time <= 7200;

formula stable:          turnCtrl.mainCtrl.ctrlProc.ctrlThread |
                             abs(currRol) < 0.5 and abs(currYaw) < 0.5;

formula reachGoal:       turnCtrl | abs(curr_dr - 60.0) < 0.5;
```

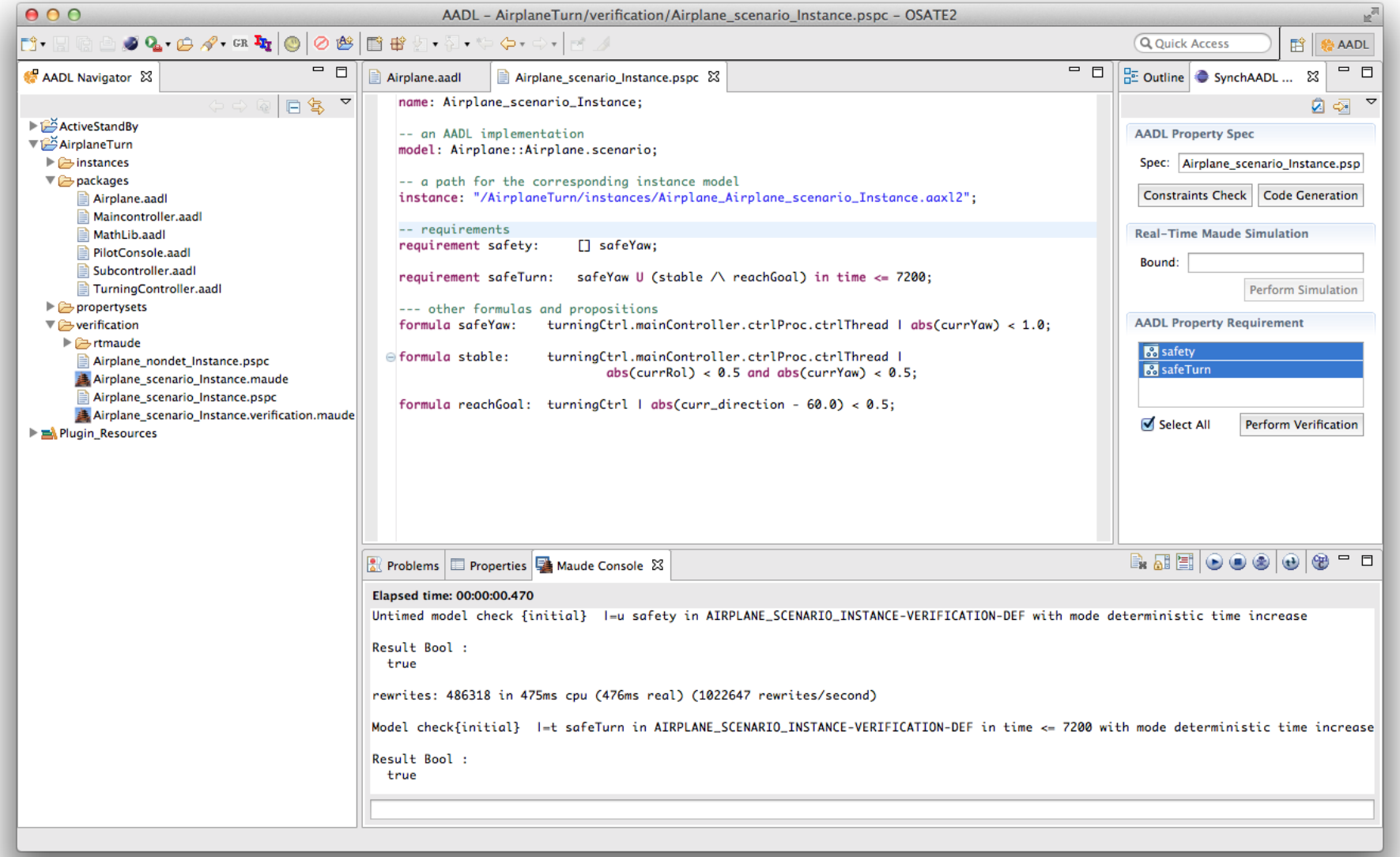
The SynchAADL2Maude Tool



The SynchAADL2Maude Tool



The SynchAADL2Maude Tool



What about Hybrid CPSs?

- PALS abstracts away the time an event takes place
  - cannot be abstracted away in hybrid systems
  - sampling continuous environment + commands to environment
    - ★ depends on local clocks

What about Hybrid CPSs?

- PALS abstracts away the time an event takes place
  - cannot be abstracted away in hybrid systems
  - sampling continuous environment + commands to environment
    - ⋆ depends on local clocks
- Nontrivial continuous behaviors
  - ODE
  - coupled environments

Hybrid PALS [Bae, Ölveczky, Kong, Gao, Clarke]

- Include time when sensing and actuating local environment
- Abstracts from
  - asynchronous communication
  - network delays
  - execution times
  - message buffering

Hybrid PALS [Bae, Ölveczky, Kong, Gao, Clarke]

- Include time when sensing and actuating local environment
- Abstracts from
  - asynchronous communication
  - network delays
  - execution times
  - message buffering
- Symbolically encode all possible local clocks
- Formal analysis problems encoded in SMT
  - satisfiability decidable up to given precision $\delta > 0$
- Case studies:
  - airplane turn
  - networked water tank controllers
  - networked thermostat controllers

Conclusions

- PALS reduces design and verification of distributed CPSs to designing and verifying underlying synchronous designs
  - abstracts away clock skews, network delays, execution times, asynchronous communication, buffering, timeouts, ...
  - enables explicit-state model checking

Conclusions

- PALS reduces design and verification of distributed CPSs to designing and verifying underlying synchronous designs
  - abstracts away clock skews, network delays, execution times, asynchronous communication, buffering, timeouts, ...
  - enables explicit-state model checking
- Makes model checking of distributed CPSs feasible

Conclusions

- PALS reduces design and verification of distributed CPSs to designing and verifying underlying synchronous designs
  - abstracts away clock skews, network delays, execution times, asynchronous communication, buffering, timeouts, ...
  - enables explicit-state model checking
- Makes model checking of distributed CPSs feasible
- Efficiency demonstrated on nontrivial avionics systems

Conclusions

- PALS reduces design and verification of distributed CPSs to designing and verifying underlying synchronous designs
  - abstracts away clock skews, network delays, execution times, asynchronous communication, buffering, timeouts, ...
  - enables explicit-state model checking
- Makes model checking of distributed CPSs feasible
- Efficiency demonstrated on nontrivial avionics systems
- Synchronous AADL: model and verify synchronous designs using AADL inside OSATE

Conclusions

- PALS reduces design and verification of distributed CPSs to designing and verifying underlying synchronous designs
  - abstracts away clock skews, network delays, execution times, asynchronous communication, buffering, timeouts, ...
  - enables explicit-state model checking
- Makes model checking of distributed CPSs feasible
- Efficiency demonstrated on nontrivial avionics systems
- Synchronous AADL: model and verify synchronous designs using AADL inside OSATE
- Extended to multi-rate and hybrid CPSs