

PALS: Virtual Synchrony for Cyber-Physical Systems

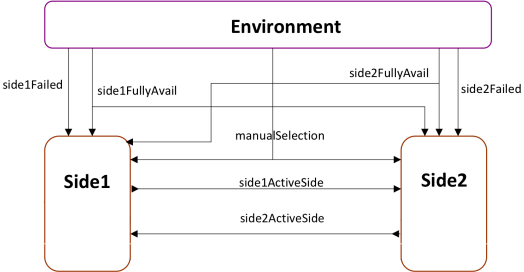
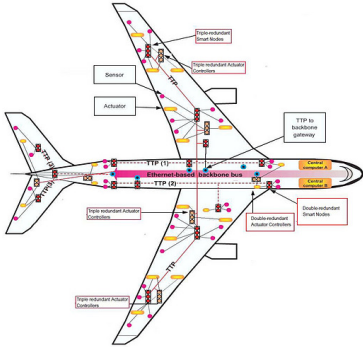
Peter Ölveczky

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?

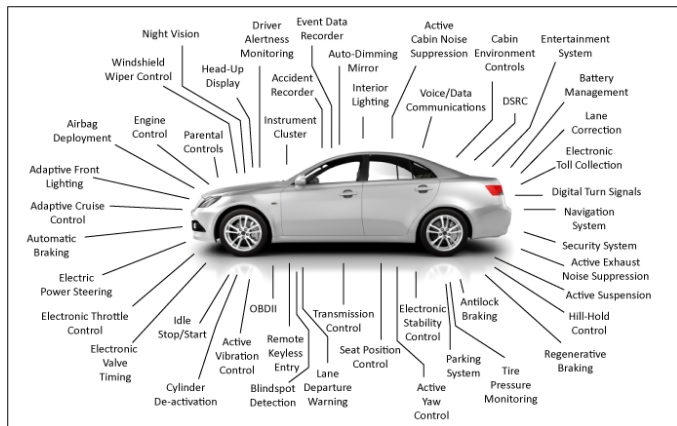


Motivation (II)

- 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

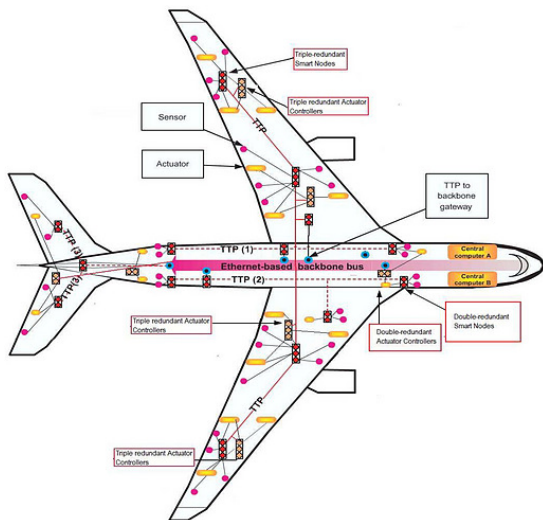
Many CPSs **virtually synchronous**



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

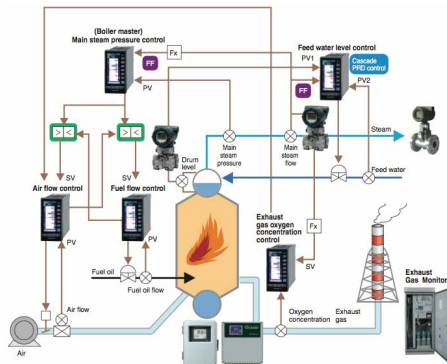
Virtually Synchronous CPSs

Many CPSs **virtually synchronous**



Virtually Synchronous CPSs

Many CPSs **virtually synchronous**



Lecture Notes in
Computer Science

1165

Jean-Raymond Abrial Egon Börger
Hans Langmaack (Eds.)

Formal Methods
for Industrial Applications

Specifying and Programming
the Steam Boiler Control



Springer

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

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

Formal patterns:

- Formalized and verified “design patterns”
- P a theory transformation

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

- **PALS**: “physically asynchronous, logically synchronous”

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

- **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**: “physically asynchronous, logically synchronous”

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

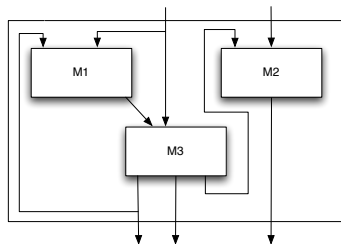
- Transformation $(SD, \Gamma) \rightarrow PALS(SD, \Gamma)$

- **Correct by construction**

$SD \models \varphi$ if and only if $PALS(SD, \Gamma) \models \varphi^*$

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:

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

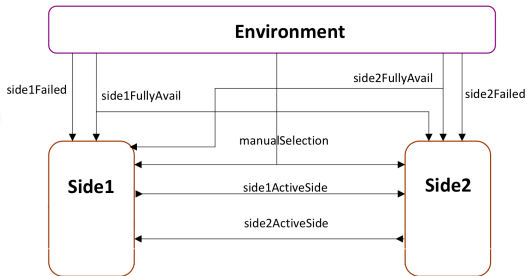
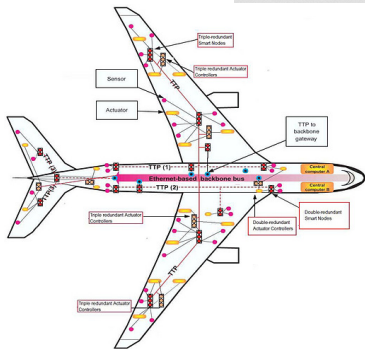
such that for each reachable stable state s in $PALS(SD, \Gamma)$ we have

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

where $L' : \mathcal{T}_{PALS(SD, \Gamma)_{GlobalSystem}} \rightarrow \mathcal{P}(AP \cup \{stable\})$ 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

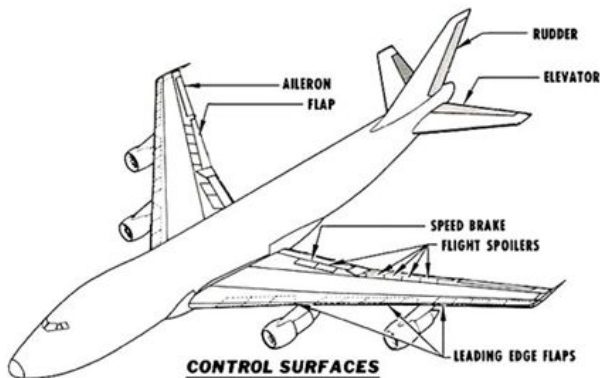
- execution times 0; perfect clocks
 - ▶ no message delay
 - ▶ message delay 0 or 1

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

- 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

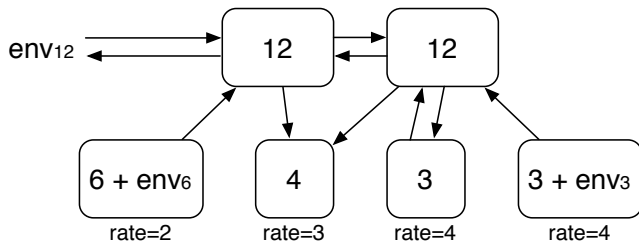
Multirate PALS: extends PALS to **multirate** hierarchical control systems

- 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

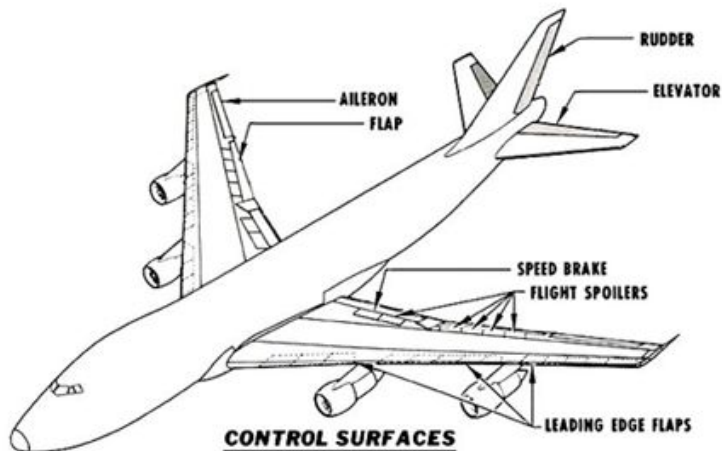
- **Fast** components perform k “internal transitions” in **one** step
 - ▶ reads/produces **k -tuples** of inputs/outputs
- **Slow** components read/produce **single** values
- **Input adaptors** transform k -tuples to/from single values

Multirate PALS II

Formalized multirate **synchronous** and **asynchronous** models

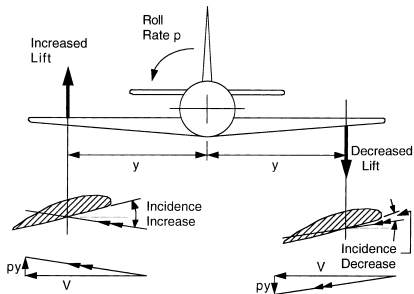
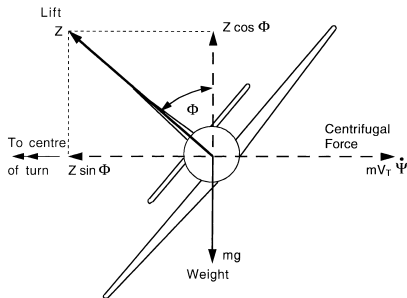
synchronous design $\models \Phi$
iff
("stable-state") asynchronous design $\models \Phi$

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



Turning an Airplane (I)

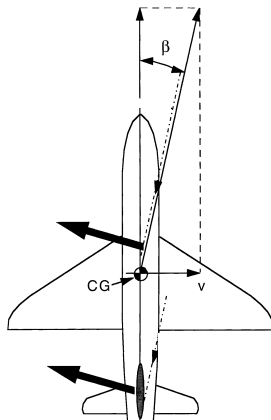
- Move ailerons to roll airplane



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)

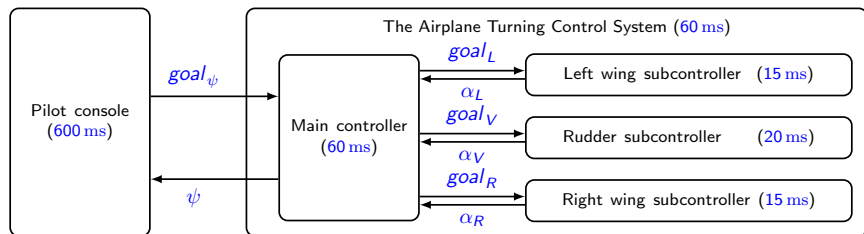
Turning control algorithm:

- 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 $< 1.0^\circ$
 - ▶ **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

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 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 $< 1.0^\circ$
- **goal direction** (60) reached within reasonable time
- plane **stable** when goal direction reached

Model Checking the Airplane Turn

Desired properties:

- **yaw** angle always $< 1.0^\circ$
- **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 screenshot shows the SynchAADL2Maude Verification tool interface. On the left is the AADL Navigator showing a project structure with folders like 'ActiveStandBy', 'aadl', 'aaxl', 'verification', and 'rtmaude', and files like 'Main_ActiveSta' and 'Plugin_Resources'. The main window is titled 'SynchAADL2Maude Verification' and contains several sections:

- AADL Instance Model:** Model Location is set to `/ActiveStandBy/aaxl/packages/Main_ActiveStandbySystem_impl_Instance.aaxl`. It includes buttons for 'Constraints Check', 'Code Generation', and 'Perform Simulation'.
- AADL Property Requirement:** A table listing properties with their names, logical formulas, and categories.
- Verification:** A dropdown menu is set to 'Main_ActiveStandbySystem_impl_Instance.prop' with a 'Perform Verification' button.
- AADL Property Values:** A section showing the result of the verification.

Name	Property	Category
R1	$O \ [] \ (\text{noChangeAssumptionNextState} \rightarrow O \ (\text{agreeOnActiveSide} \ \wedge \ O \ (\text{neitherSideFailed} \rightarrow \text{agreeOnActiveSide})))$	LTL
R2a	$O \ [] \ ((\text{noChangeAssumptionNextState} \ \wedge \ O \ \text{side1FullyAvailable} \ \wedge \ O \ \sim \ \text{side2FullyAvailable}) \rightarrow O \ (\text{side1FullyAvailable} \ \wedge \ \sim \ \text{side2FullyAvailable}))$	LTL
R3g	$\square \ (\ (\sim \ \text{manSelectPressed} \ \wedge \ \text{agreeOnActiveSide} \ \wedge \ \text{side1FullyAvailable} \ \wedge \ \text{side2FullyAvailable} \ \wedge \ \sim \ \text{manSelectPressed}) \rightarrow O \ (\text{side1FullyAvailable} \ \wedge \ \sim \ \text{side2FullyAvailable}))$	LTL
R4	$\square \ (((\text{side1Failed} \ \wedge \ \sim \ \text{side2Failed}) \rightarrow O \ (\sim \ \text{side2Failed} \rightarrow \text{side2Active})) \ \wedge \ ((\text{side2Failed} \ \wedge \ \sim \ \text{side1Failed}) \rightarrow O \ (\sim \ \text{side1Failed} \rightarrow \text{side1Active})))$	LTL
R5side1	$\square \ (((\text{side1Active} \ \wedge \ \text{side1FullyAvailable} \ \wedge \ \sim \ \text{manSelectPressed}) \rightarrow O \ (\text{side1Active} \ W \ (\sim \ \text{side1FullyAvailable} \ U \ \text{manSelectPressed})))$	LTL

The 'AADL Property Values' section shows the following text:

```
Ready.  
Main_ActiveStandbySystem_impl_Instance Verification: All properties  
deterministic time increase  
Result Bool :  
true
```

The SynchAADL2Maude Tool

The screenshot displays the SynchAADL2Maude tool interface. The main window shows the AADL Navigator on the left, the AADL model editor in the center, and the SynchAADL2Maude tool on the right. The AADL model editor contains the following code:

```
name: Airplane_scenario_Instance;
-- an AADL implementation
model: Airplane::Airplane_scenario;
-- a path for the corresponding instance model
instance: "/AirplaneTurn/instances/Airplane_Airplane_scenario_Instance.aaxl2";
-- requirements
requirement safety:  safeYaw;
requirement safeTurn: safeYaw U (stable ^ reachGoal) in time <= 7200;
--- other formulas and propositions
formula safeYaw: turningCtrl.mainController.ctrlProc.ctrlThread ! abs(currYaw) < 1.0;
```

The SynchAADL2Maude tool on the right has the following sections:

- AADL Property Spec:** Spec: Airplane_scenario_Instance.psp. Buttons: Constraints Check, Code Generation.
- Real-Time Maude Simulation:** Bound: [text box]. Button: Perform Simulation.
- AADL Property Requirement:** List: safety, safeTurn. Button: Select All. Button: Perform Verification.

A dialog box titled "Synch AADL Constraints Checker" is open in the center, displaying the message "Valid Synchronous AADL model!" with an "OK" button.

The bottom status bar shows "Maude isn't running." and a "Problems" tab.

The SynchAADL2Maude Tool

The screenshot displays the SynchAADL2Maude tool interface. The main window shows the AADL code for 'Airplane.aadl' and 'Airplane_scenario_Instance.pspc'. The code includes requirements for 'safeYaw' and 'safeTurn', and formulas for 'safeYaw', 'stable', and 'reachGoal'. The right sidebar contains controls for 'AADL Property Spec', 'Real-Time Maude Simulation', and 'AADL Property Requirement'. The bottom Maude Console shows the execution results of the model check.

```
name: Airplane_scenario_Instance;
-- an AADL implementation
model: Airplane::Airplane.scenario;

-- a path for the corresponding instance model
instance: "/AirplaneTurn/instances/Airplane_Airplane_scenario_Instance.aaxl2";

-- requirements
requirement safety:  safeYaw;

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

--- other formulas and propositions
formula safeYaw: turningCtrl.mainController.ctrlProc.ctrlThread | abs(currYaw) < 1.0;

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

formula reachGoal: turningCtrl | abs(curr_direction - 60.0) < 0.5;
```

Maude Console Output:

```
Elapsed time: 00:00:00.470
Untimed model check [initial] l-u safety in AIRPLANE_SCENARIO_INSTANCE-VERIFICATION-DEF with mode deterministic time increase

Result Bool :
true

rewrites: 486318 in 475ms cpu (476ms real) (1022647 rewrites/second)

Model check[initial] l-t safeTurn in AIRPLANE_SCENARIO_INSTANCE-VERIFICATION-DEF in time <= 7200 with mode deterministic time increase

Result Bool :
true
```

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

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

- 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