Formal Model Engineering of Synchronous CPS Designs in AADL

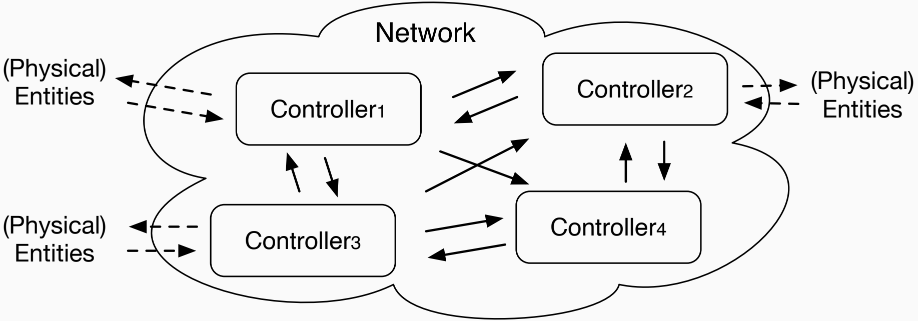
Kyungmin Bae[1] Peter Csaba Ölveczky[2]

[1]Pohang University of Science and Technology (POSTECH), Korea
[2]University of Oslo, Norway

2nd ADEPT workshop: AADL by its practitioners
June 16, 2023

Cyber-Physical Systems (CPSs)

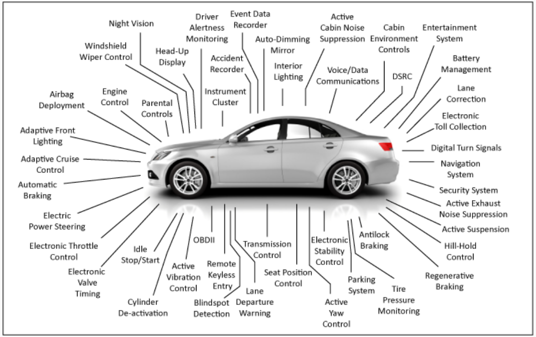
- Distributed controllers that interact with (physical) environments



- Many safety-critical applications

Cyber-Physical Systems (CPSs)

- Distributed controllers that interact with (physical) environments



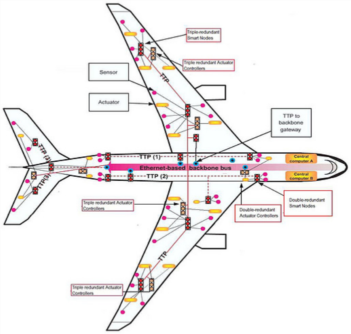
- Many safety-critical applications

---

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

Cyber-Physical Systems (CPSs)

- Distributed controllers that interact with (physical) environments



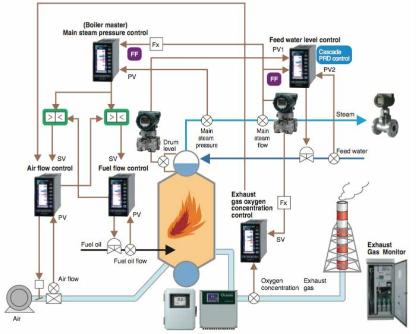
- Many safety-critical applications

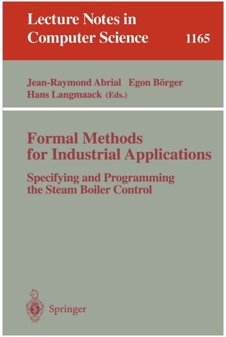
---

http://articles.sae.org/10234/

Cyber-Physical Systems (CPSs)

- Distributed controllers that interact with (physical) environments





- Many safety-critical applications

---

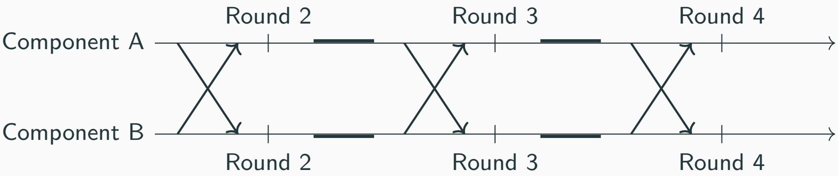
https://web-material3.yokogawa.com/image_8434.jpg

Formal Methods + Model-Based Development with AADL

- General models for many deployment scenarios
- Efficient formal analysis of such general models
- Safe deployment of the analyzed models

Many CPSs are Virtually Synchronous CPSs

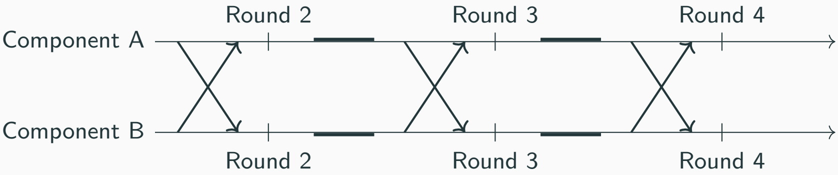
- Synchronous behavior & distributed realization: avionics, automotive, . . .



- Have be correct in many distributed settings
  - time synchronization (IEEE 1588, etc.), bounded network delays, . . .

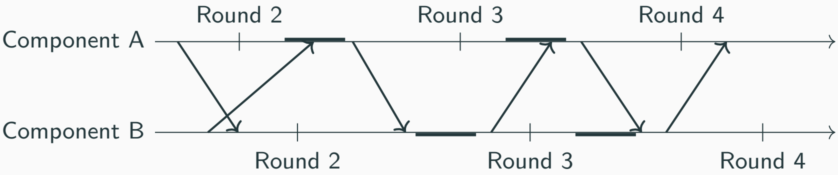
Designing and Verifying Virtually Synchronous CPSs

- Hard to design



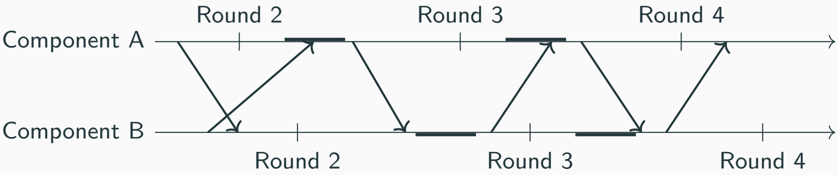
Designing and Verifying Virtually Synchronous CPSs

- Hard to design: network delays, execution times, clock skews, race conditions, . . .



Designing and Verifying Virtually Synchronous CPSs

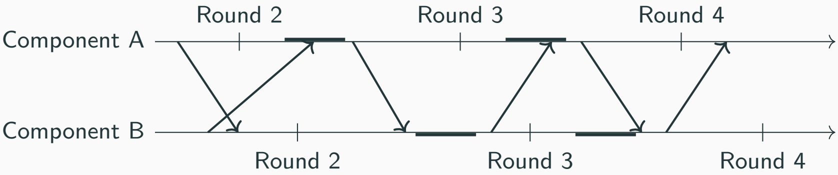
- Hard to design: network delays, execution times, clock skews, race conditions, ...



- Hard to verify
  - (discrete) state space explosion due to asynchrony (+ continuous dynamics)

Designing and Verifying Virtually Synchronous CPSs

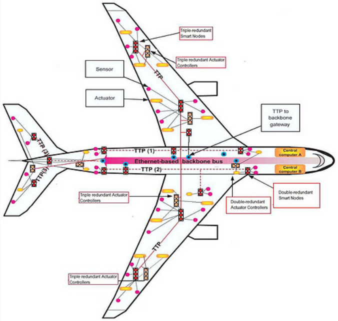
- Hard to design: network delays, execution times, clock skews, race conditions, ...

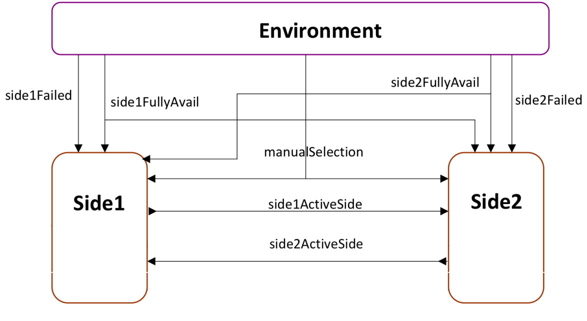


- Hard to verify
  - (discrete) state space explosion due to asynchrony (+ continuous dynamics)
- Hard to deploy correctly
  - small changes can cause bugs if only verified for specific deployment scenario

Example: Which Cabinet is Active?

Rockwell Collins





- $\geq$ 30 hours to model-check, (for specific network delays, execution times, . . .)

Long-Term Goal

Goal

Enable automated formal analysis for domain-specific modeling of virtually synchronous CPSs

- An easy-to-use modeling language for CPS developers
- A tool integrated with mature modeling environments
- A technique to reduce the design and verification complexity

Our Approach

1. Model synchronous design *SD* in the HybridSynchAADL modeling language
2. Verify *SD* using the HybridSynchAADL OSATE plugin
3. Obtain the corresponding asynchronous model using the Hybrid PALS synchronizer

Modeling Language

Language Design

- Goal
  - to abstractly capture many deployment scenarios
  - to model advanced control programs and continuous behaviors in AADL
- Design choice
  - use subset of AADL
  - leverage existing AADL constructs as much as possible

The HybridSynchAADL Modeling Language (1)

- Model synchronous designs with continuous dynamics
  - distributed controllers, continuous environments, sampling/actuation times, ...

The HybridSynchAADL Modeling Language (1)

- Model synchronous designs with continuous dynamics
  - distributed controllers, continuous environments, sampling/actuation times, ...
- Distributed controllers
  - in a subset of AADL: constructs have the same meaning as in AADL

The HybridSynchAADL Modeling Language (1)

- Model synchronous designs with continuous dynamics
  - distributed controllers, continuous environments, sampling/actuation times, ...
- Distributed controllers
  - in a subset of AADL: constructs have the same meaning as in AADL
- Continuous environments
  - continuous dynamics specified using continuous real functions or ODEs

The HybridSynchAADL Modeling Language (2)

- Extended with new AADL property set Hybrid_SynchAADL

```
property set Hybrid_SynchAADL is
    Synchronous: inherit aadlboolean applies to (system);
    isEnvironment: inherit aadlboolean applies to (system);
    ContinuousDynamics: aadlstring applies to (system);
    Max_Clock_Deviation: inherit Time applies to (system);
    Sampling_Time: inherit Time_Range applies to (system);
    Response_Time: inherit Time_Range applies to (system);
end Hybrid_SynchAADL;
```

- Minimize new syntactic extensions for ease of use by existing AADL users

Formal Semantics

Formal Semantics of HybridSynchAADL

- Formal semantics of HybridSynchAADL in Maude and SMT
  - Maude: a language and tool for specifying and analyzing distributed systems

Formal Semantics of HybridSynchAADL

- Formal semantics of HybridSynchAADL in Maude and SMT
  - Maude: a language and tool for specifying and analyzing distributed systems
- Discrete behaviors are specified in Maude
  - thread behaviors, behavior annex programs, synchronous communication, ...

Formal Semantics of HybridSynchAADL

- Formal semantics of HybridSynchAADL in Maude and SMT
  - Maude: a language and tool for specifying and analyzing distributed systems
- Discrete behaviors are specified in Maude
  - thread behaviors, behavior annex programs, synchronous communication, ...
- Continuous behaviors are encoded in SMT
  - continuous dynamics, sampling/actuation times, clock skews, ...

Formal Analysis of HybridSynchAADL

- Randomized simulation
    - randomly "samples" clock skews, sampling/actuation times, initial values, . . .

Formal Analysis of HybridSynchAADL

- Randomized simulation
  - randomly "samples" clock skews, sampling/actuation times, initial values, ...
- Symbolic reachability analysis
  - all possible continuous behaviors are encoded in SMT

Formal Analysis of HybridSynchAADL

- Randomized simulation
    - randomly "samples" clock skews, sampling/actuation times, initial values, ...
- Symbolic reachability analysis
    - all possible continuous behaviors are encoded in SMT
- Portfolio analysis
    - execute randomized simulation and symbolic reachability analysis in parallel

Experimental Evaluation

- Compare the performance of HybridSynchAADL's symbolic reachability analysis
  - with hybrid automata reachability analysis tools: HyComp, SpaceEx, Flow*, dReach

Experimental Evaluation

- Compare the performance of HybridSynchAADL's symbolic reachability analysis
  - with hybrid automata reachability analysis tools: HyComp, SpaceEx, Flow*, dReach
- Analysis invariant properties of synchronous designs up to bounds
  - two properties: $\mathit{Inv}_{\top}$ (which holds), and $\mathit{Inv}_{\bot}$ (which does not hold)

Experimental Evaluation

Model	Tool	$Inv_\top$ $N=2$ Time	B	$Inv_\top$ $N=3$ Time	B	$Inv_\top$ $N=4$ Time	B	$Inv_\bot$ $N=2$ Time	B	$Inv_\bot$ $N=3$ Time	B	$Inv_\bot$ $N=4$ Time	B
Rend (single)	HSADDL	2.0	5	3.9	5	5.8	5	2.4	3	4.2	3	5.9	3
	HyComp	0.8	5	4.0	5	17.2	5	8.9	3	11.5	3	192.6	3
	SpaceEx	8.0	5	2230.3	3	4.5	1	5.1	3	2676.6	3	T/O	-
	dReach	1382.7	3	107.1	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	3552.8	4	2725.5	2	1205.2	1	167.3	3	380.4	2	838.0	3
Form (single)	HSAADL	3.0	5	7.3	5	7.9	5	15.5	4	2.5	2	5.2	2
	HyComp	13.3	5	41.3	5	182.1	5	T/O	-	2.6	2	20.3	2
	SpaceEx	91.9	2	2.8	1	114.8	1	T/O	-	T/O	-	T/O	-
	dReach	139.0	1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	1464.7	2	873.4	1	T/O	-	T/O	-	45.3	1	291.3	2
Thermostat	HSAADL	2.7	5	4.7	5	7.8	5	7.6	5	15.3	5	10.7	4
	HyComp	1.6	5	8.5	5	37.9	5	2.6	5	15.5	5	43.1	4
	SpaceEx	2.3	5	696.4	3	34.5	1	2.2	5	T/O	-	T/O	-
	dReach	341.6	3	57.5	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	3196.4	5	1240.7	2	977.7	1	15.5	3	1718.1	4	T/O	-
Rend (double)	HSAADL	3.7	4	37.8	4	6.9	4	1.4	2	16.3	2	2.8	2
	SpaceEx	1147.6	3	81.1	1	T/O	-	15.2	2	T/O	-	T/O	-
	dReach	2156.2	3	274.3	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	232.5	2	230.1	1	T/O	-	2.2	2	25.4	2	2613.8	1

Timeout: 3,600 seconds

N: # components

B: # iterations

$Inv_\top$: largest B for which tool could analyze

$Inv_\bot$: smallest B where counterexample found

Experimental Evaluation

Model	Tool	$Inv_\top$						$Inv_\bot$					
		$N=2$		$N=3$		$N=4$		$N=2$		$N=3$		$N=4$	
		Time	B	Time	B	Time	B	Time	B	Time	B	Time	B
Rend (single)	HSADDL	2.0	5	3.9	5	5.8	5	2.4	3	4.2	3	5.9	3
	HyComp	0.8	5	4.0	5	17.2	5	8.9	3	11.5	3	192.6	3
	SpaceEx	8.0	5	2230.3	3	4.5	1	5.1	3	2676.6	3	T/O	-
	dReach	1382.7	3	107.1	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	3552.8	4	2725.5	2	1205.2	1	167.3	3	380.4	2	838.0	3
Form (single)	HSAADL	3.0	5	7.3	5	7.9	5	15.5	4	2.5	2	5.2	2
	HyComp	13.3	5	41.3	5	182.1	5	T/O	-	2.6	2	20.3	2
	SpaceEx	91.9	2	2.8	1	114.8	1	T/O	-	T/O	-	T/O	-
	dReach	139.0	1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	1464.7	2	873.4	1	T/O	-	T/O	-	45.3	1	291.3	2
Thermostat	HSAADL	2.7	5	4.7	5	7.8	5	7.6	5	15.3	5	10.7	4
	HyComp	1.6	5	8.5	5	37.9	5	2.6	5	15.5	5	43.1	4
	SpaceEx	2.3	5	696.4	3	34.5	1	2.2	5	T/O	-	T/O	-
	dReach	341.6	3	57.5	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	3196.4	5	1240.7	2	977.7	1	15.5	3	1718.1	4	T/O	-
Rend (double)	HSAADL	3.7	4	37.8	4	6.9	4	1.4	2	16.3	2	2.8	2
	SpaceEx	1147.6	3	81.1	1	T/O	-	15.2	2	T/O	-	T/O	-
	dReach	2156.2	3	274.3	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	232.5	2	230.1	1	T/O	-	2.2	2	25.4	2	2613.8	1

Timeout: 3,600 seconds

N: # components

B: # iterations

$Inv_\top$: largest B for which tool could analyze

$Inv_\bot$: smallest B where counterexample found

Experimental Evaluation

Model	Tool	$Inv_\top$						$Inv_\bot$					
		$N = 2$		$N = 3$		$N = 4$		$N = 2$		$N = 3$		$N = 4$	
		Time	B	Time	B	Time	B	Time	B	Time	B	Time	B
Rend (single)	HSADDL	2.0	5	3.9	5	5.8	5	2.4	3	4.2	3	5.9	3
	HyComp	0.8	5	4.0	5	17.2	5	8.9	3	11.5	3	192.6	3
	SpaceEx	8.0	5	2230.3	3	4.5	1	5.1	3	2676.6	3	T/O	-
	dReach	1382.7	3	107.1	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	3552.8	4	2725.5	2	1205.2	1	167.3	3	380.4	2	838.0	3
Form (single)	HSAADL	3.0	5	7.3	5	7.9	5	15.5	4	2.5	2	5.2	2
	HyComp	13.3	5	41.3	5	182.1	5	T/O	-	2.6	2	20.3	2
	SpaceEx	91.9	2	2.8	1	114.8	1	T/O	-	T/O	-	T/O	-
	dReach	139.0	1	T/O	-	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	1464.7	2	873.4	1	T/O	-	T/O	-	45.3	1	291.3	2
Thermostat	HSAADL	2.7	5	4.7	5	7.8	5	7.6	5	15.3	5	10.7	4
	HyComp	1.6	5	8.5	5	37.9	5	2.6	5	15.5	5	43.1	4
	SpaceEx	2.3	5	696.4	3	34.5	1	2.2	5	T/O	-	T/O	-
	dReach	341.6	3	57.5	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	3196.4	5	1240.7	2	977.7	1	15.5	3	1718.1	4	T/O	-
Rend (double)	HSAADL	3.7	4	37.8	4	6.9	4	1.4	2	16.3	2	2.8	2
	SpaceEx	1147.6	3	81.1	1	T/O	-	15.2	2	T/O	-	T/O	-
	dReach	2156.2	3	274.3	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	232.5	2	230.1	1	T/O	-	2.2	2	25.4	2	2613.8	1

Timeout: 3,600 seconds

N: # components

B: # iterations

$Inv_\top$: largest B for which tool could analyze

$Inv_\bot$: smallest B where counterexample found

Experimental Evaluation

Model	Tool	$Inv_{\top}$ $N=2$ Time	B	$Inv_{\top}$ $N=3$ Time	B	$Inv_{\top}$ $N=4$ Time	B	$Inv_{\bot}$ $N=2$ Time	B	$Inv_{\bot}$ $N=3$ Time	B	$Inv_{\bot}$ $N=4$ Time	B
Rend (single)	HSADDL	2.0	5	3.9	5	5.8	5	2.4	3	4.2	3	5.9	3
	HyComp	0.8	5	4.0	5	17.2	5	8.9	3	11.5	3	192.6	3
	SpaceEx	8.0	5	2230.3	3	4.5	1	5.1	3	2676.6	3	T/O	-
	dReach	1382.7	3	107.1	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	3552.8	4	2725.5	2	1205.2	1	167.3	3	380.4	2	838.0	3
Form (single)	HSAADL												
	HyComp												
	SpaceEx												
	dReach												
	Flow*												
Thermostat	HSAADL	2.7	5	4.7	5	7.8	5	7.6	5	15.3	5	10.7	4
	HyComp	1.6	5	8.5	5	37.9	5	2.6	5	15.5	5	43.1	4
	SpaceEx	2.3	5	696.4	3	34.5	1	2.2	5	T/O	-	T/O	-
	dReach	341.6	3	57.5	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	3196.4	5	1240.7	2	977.7	1	15.5	3	1718.1	4	T/O	-
Rend (double)	HSAADL	3.7	4	37.8	4	6.9	4	1.4	2	16.3	2	2.8	2
	SpaceEx	1147.6	3	81.1	1	T/O	-	15.2	2	T/O	-	T/O	-
	dReach	2156.2	3	274.3	1	T/O	-	T/O	-	T/O	-	T/O	-
	Flow*	232.5	2	230.1	1	T/O	-	2.2	2	25.4	2	2613.8	1

Timeout: 3,600 seconds

N: # components

...ons

...B for which

...d analyze

$Inv_{\bot}$: smallest B where counterexample found

From Experimental Results

HybridSynchAADL is effective for analyzing models with both complex control programs and continuous behaviors

Complexity Reduction

Reducing Design/Verification Complexity Using Synchronizers

Formal design patterns (or synchronizers) for CPSs

1. Design and verify synchronous system *SD*
    - abstract from communication, network delays, clock skews, execution times, . . .

Reducing Design/Verification Complexity Using Synchronizers

Formal design patterns (or synchronizers) for CPSs

1. Design and verify synchronous system SD
   - abstract from communication, network delays, clock skews, execution times, . . .
2. Obtain correct-by-construction distributed system $\mathcal{A}(SD, \Gamma)$
   - provided bounds Γ on network delay, execution time, and clock skew

Reducing Design/Verification Complexity Using Synchronizers

Formal design patterns (or synchronizers) for CPSs

1. Design and verify synchronous system SD
   - abstract from communication, network delays, clock skews, execution times, . . .
2. Obtain correct-by-construction distributed system $\mathcal{A}(SD, \Gamma)$
   - provided bounds Γ on network delay, execution time, and clock skew

- Examples: TTA, LTTA, PALS, HybridPALS, MSYNC, . . .

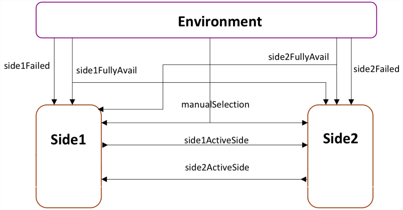
Examples: Synch vs. Async

- Analyzed both synchronous and simplified asynchronous models in Maude
  - no execution times, no clock skews, no message delays

Examples: Synch vs. Async

- Analyzed both synchronous and simplified asynchronous models in Maude
  - no execution times, no clock skews, no message delays

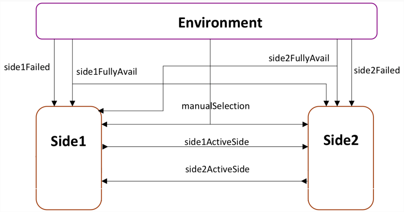
- Which Cabinet is Active?
  - sync. model: 185 states
  - async. model: 3,047,832 states



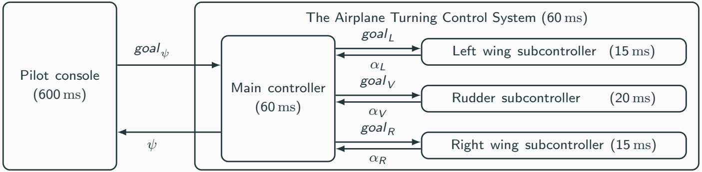
Examples: Synch vs. Async

- Analyzed both synchronous and simplified asynchronous models in Maude
  - no execution times, no clock skews, no message delays

- Which Cabinet is Active?
  - sync. model: 185 states
  - async. model: 3,047,832 states



- Turning an Airplane
  - sync. model: 364 states
  - async. model: 420,288 states



Hybrid PALS

- PALS and TTA: abstract away time when an event takes place
  - not possible in hybrid systems
  - sensing/actuating time of continuous environment depends on local clocks

Hybrid PALS

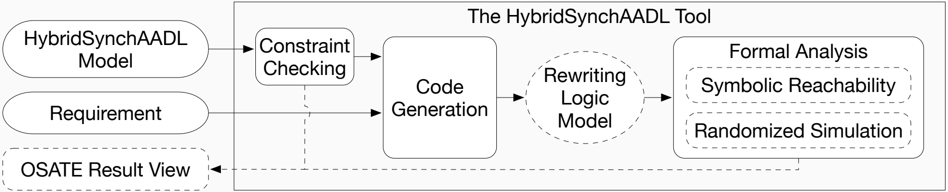
- PALS and TTA: abstract away time when an event takes place
  - not possible in hybrid systems
  - sensing/actuating time of continuous environment depends on local clocks
- Hybrid PALS: include time when sensing/actuating local environment
  - abstract from asynchronous communication, network delays, execution times, . . .
  - symbolically encode all possible local clocks

Examples: Synch vs. Async

Model	N	B	Synchronous Models: |Sample| = 1 Time	|Sample| = 1 #State	|Sample| = 2 Time	|Sample| = 2 #State	|Sample| = 3 Time	|Sample| = 3 #State	Asynchronous Models: |Sample| = 1 Time	|Sample| = 1 #State	|Sample| = 2 Time	|Sample| = 2 #State	|Sample| = 3 Time	|Sample| = 3 #State
Rend (single)	2	1	0.01	0.03	0.02	0.1	0.1	0.2	6.0	10.7	53.5	90.5	251.4	393.0
		2	0.01	0.05	0.2	0.6	1.2	4.1	9.7	19.4	73.1	135.2	317.5	528.8
		3	0.02	0.07	2.6	8.9	100.7	297.9	15.0	31.1	107.0	208.1	447.7	769.5
	3	1	0.01	0.04	0.1	0.2	0.2	0.5	970.4	939.7	T/O	-	T/O	-
Form (single)	2	1	0.01	0.03	0.05	0.1	0.2	0.3	7,937.9	3,888.4	T/O	-	T/O	-
	3	1	0.02	0.05	0.1	0.3	0.5	0.9	T/O	-	T/O	-	T/O	-
Rend (double)	2	1	0.01	0.02	0.03	0.1	0.1	0.1	145.1	188.2	1,557.6	1,500.6	15,348.1	6,339.2
		2	0.02	0.04	0.1	0.2	0.6	8.6	826.3	1,121.8	10,200.0	5,495.6	T/O	-
		3	0.03	0.07	0.9	1.7	12.9	24.8	2,773.4	2,764.0	T/O	-	T/O	-
	3	1	0.01	0.03	0.1	0.1	0.2	0.3	T/O	-	T/O	-	T/O	-
Form (double)	2	1	0.02	0.03	0.1	0.1	0.1	0.1	T/O	-	T/O	-	T/O	-
	3	1	0.03	0.04	0.1	0.2	0.4	0.4	T/O	-	T/O	-	T/O	-

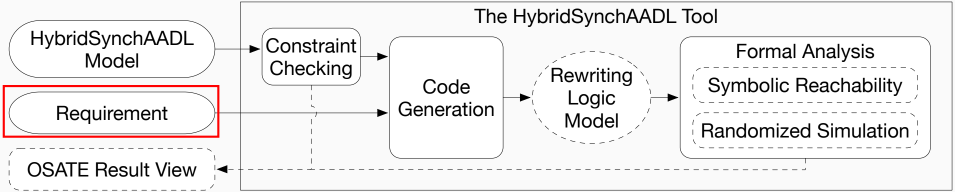
Tool and Case Study

The HybridSynchAADL Tool



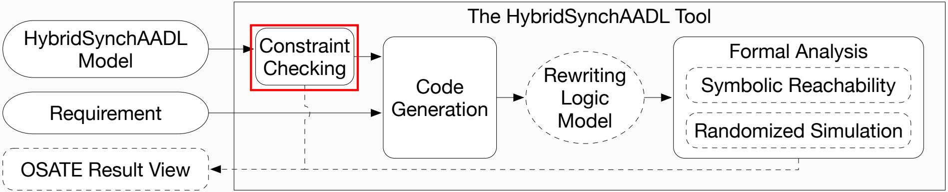
- OSATE plug-in

The HybridSynchAADL Tool



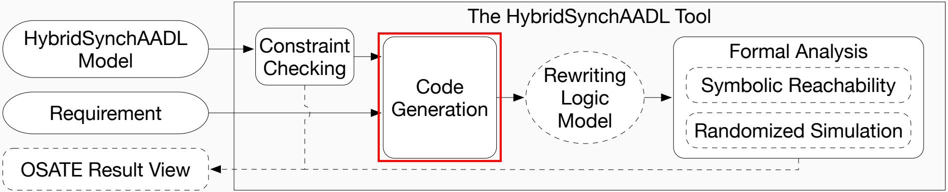
- OSATE plug-in
- Provide an intuitive language to specify properties of models

The HybridSynchAADL Tool



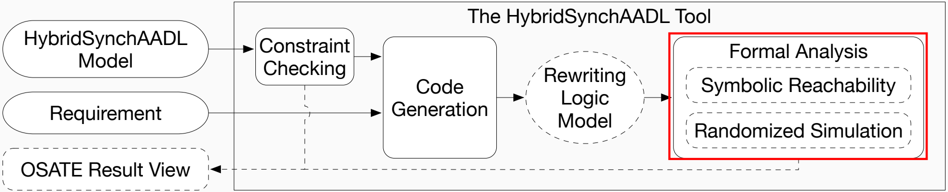
- OSATE plug-in
- Provide an intuitive language to specify properties of models
- Check if a given model is a valid HybridSynchAADL model

The HybridSynchAADL Tool



- OSATE plug-in
- Provide an intuitive language to specify properties of models
- Check if a given model is a valid HybridSynchAADL model
- Use OSATE's code generation facilities to synthesize a Maude model

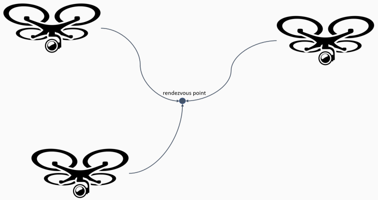
The HybridSynchAADL Tool

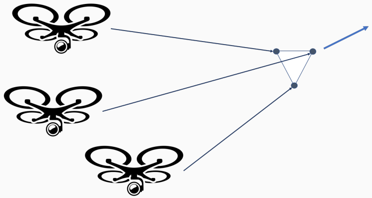


- OSATE plug-in
- Provide an intuitive language to specify properties of models
- Check if a given model is a valid HybridSynchAADL model
- Use OSATE's code generation facilities to synthesize a Maude model
- Invoke Maude and Yices2 to perform formal analysis

Case Study: Collaborating Autonomous Drones

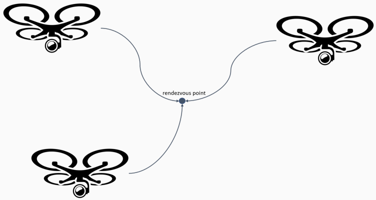
- Collaborate to achieve common goals (e.g., rendezvous, formation, . . .)

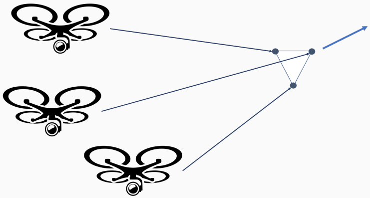




Case Study: Collaborating Autonomous Drones

- Collaborate to achieve common goals (e.g., rendezvous, formation, . . .)





- Continuous dynamics of each drone

$$\dot{\vec{x}} = \vec{v} \qquad \text{(position } \vec{x}, \text{ velocity } \vec{v}\text{)}$$

Case Study: Collaborating Autonomous Drones

- Collaborate to achieve common goals (e.g., rendezvous, formation, ...)

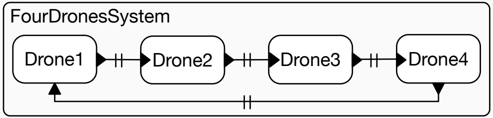
- Continuous dynamics of each drone

$$\dot{\vec{x}} = \vec{v} \qquad \text{(position } \vec{x}\text{, velocity } \vec{v}\text{)}$$

- Controller of each drone
  - determines the velocity $\vec{v}$ according to the status of the other drones

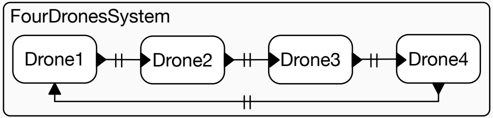
Example: Rendezvous of Four Distributed Drones

- Each drone communicates with two other drones

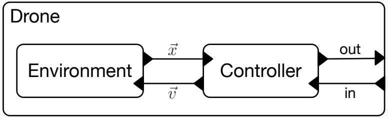


Example: Rendezvous of Four Distributed Drones

- Each drone communicates with two other drones



- A drone component consists of an environment and its controller



Example: System Architecture in HybridSynchAADL (1)

- A top-level system component (a subset of AADL)

```
system implementation FourDronesSystem.impl
  subcomponents
    drones: system Drone::Drone.impl;    dr2: system Drone::Drone.impl;
    dr3: system Drone::Drone.impl;    dr4: system Drone::Drone.impl;
  connections
    C1: port dr1.oX -> dr2.iX;        C2: port dr1.oY -> dr2.iY;
    C3: port dr2.oX -> dr3.iX;        C4: port dr2.oY -> dr3.iY;
    C5: port dr3.oX -> dr4.iX;        C6: port dr3.oY -> dr4.iY;
    C7: port dr4.oX -> dr1.iX;        C8: port dr4.oY -> dr1.iY;
  properties
    Timing => Delayed applies to C1, C2, C3, C4, C5, C6, C7, C8;
    Period => 100ms;
    Hybrid_SynchAADL::Synchronous => true;
    Hybrid_SynchAADL::Max_Clock_Deviation => 10ms;
end FourDrones.impl;
```

Example: System Architecture in HybridSynchAADL (1)

- A top-level system component (a subset of AADL)

```
system implementation FourDronesSystem.impl
  subcomponents
    drones: system Drone::Drone.impl;    dr2: system Drone::Drone.impl;
    dr3: system Drone::Drone.impl;    dr4: system Drone::Drone.impl;
  connections
    C1: port dr1.oX -> dr2.iX;        C2: port dr1.oY -> dr2.iY;
    C3: port dr2.oX -> dr3.iX;        C4: port dr2.oY -> dr3.iY;
    C5: port dr3.oX -> dr4.iX;        C6: port dr3.oY -> dr4.iY;
    C7: port dr4.oX -> dr1.iX;        C8: port dr4.oY -> dr1.iY;
  properties
    Timing => Delayed applies to C1, C2, C3, C4, C5, C6, C7, C8;
    Period => 100ms;
    Hybrid_SynchAADL::Synchronous => true;
    Hybrid_SynchAADL::Max_Clock_Deviation => 10ms;
end FourDrones.impl;
```

drone components

Example: System Architecture in HybridSynchAADL (1)

- A top-level system component (a subset of AADL)

```
system implementation FourDronesSystem.impl
  subcomponents
    drones: system Drone::Drone.impl;    dr2: system Drone::Drone.impl;
    dr3: system Drone::Drone.impl;    dr4: system Drone::Drone.impl;
  connections
    C1: port dr1.oX -> dr2.iX;        C2: port dr1.oY -> dr2.iY;
    C3: port dr2.oX -> dr3.iX;        C4: port dr2.oY -> dr3.iY;
    C5: port dr3.oX -> dr4.iX;        C6: port dr3.oY -> dr4.iY;
    C7: port dr4.oX -> dr1.iX;        C8: port dr4.oY -> dr1.iY;
  properties
    Timing => Delayed applies to C1, C2, C3, C4, C5, C6, C7, C8;
    Period => 100ms;
    Hybrid_SynchAADL::Synchronous => true;
    Hybrid_SynchAADL::Max_Clock_Deviation => 10ms;
end FourDrones.impl;
```

network connections

Example: System Architecture in HybridSynchAADL (1)

- A top-level system component (a subset of AADL)

```
system implementation FourDronesSystem.impl
  subcomponents
    drones: system Drone::Drone.impl;    dr2: system Drone::Drone.impl;
    dr3: system Drone::Drone.impl;    dr4: system Drone::Drone.impl;
  connections
    C1: port dr1.oX -> dr2.iX;        C2: port dr1.oY -> dr2.iY;
    C3: port dr2.oX -> dr3.iX;        C4: port dr2.oY -> dr3.iY;
    C5: port dr3.oX -> dr4.iX;        C6: port dr3.oY -> dr4.iY;
    C7: port dr4.oX -> dr1.iX;        C8: port dr4.oY -> dr1.iY;
  properties
    Timing => Delayed applies to C1, C2, C3, C4, C5, C6, C7, C8;
    Period => 100ms;
    Hybrid_SynchAADL::Synchronous => true;
    Hybrid_SynchAADL::Max_Clock_Deviation => 10ms;
end FourDrones.impl;
```

HybridSynchAADL annotations

Example: System Architecture in HybridSynchAADL (2)

- A drone component (a subset of AADL)

```
system Drone
  features
    iX: in data port Base_Types::Float;          iY: in data port Base_Types::Float;
    oX: out data port Base_Types::Float;         oY: out data port Base_Types::Float;
end Drone;

system implementation Drone.impl
  subcomponents
    ctl: system DroneControl::DroneControl.impl;
    env: system Environment::Environment.impl;
  connections
    C1: port ctl.oX -> oX;      C2: port ctl.oY -> oY;         C3: port iX -> ctl.iX;
    C4: port iY -> ctl.iY;      C5: port ctl.vX -> env.vX;      C6: port ctl.vY -> env.vY;
    C7: port env.cX -> ctl.cX;  C8: port env.cY -> ctl.cY;
  properties
    Hybrid_SynchAADL::Sampling_Time => 2ms .. 4ms;
    Hybrid_SynchAADL::Response_Time => 6ms .. 9ms;
end Drone.impl;
```

Example: System Architecture in HybridSynchAADL (2)

- A drone component (a subset of AADL)

```
system Drone
  features
    iX: in data port Base_Types::Float;         iY: in data port Base_Types::Float;
    oX: out data port Base_Types::Float;        oY: out data port Base_Types::Float;
end Drone;

system implementation Drone.impl
  subcomponents
    ctl: system DroneControl::DroneControl.impl;
    env: system Environment::Environment.impl;
  connections
    C1: port ctl.oX -> oX;      C2: port ctl.oY -> oY;        C3: port iX -> ctl.iX;
    C4: port iY -> ctl.iY;      C5: port ctl.vX -> env.vX;      C6: port ctl.vY -> env.vY;
    C7: port env.cX -> ctl.cX;  C8: port env.cY -> ctl.cY;
  properties
    Hybrid_SynchAADL::Sampling_Time => 2ms .. 4ms;
    Hybrid_SynchAADL::Response_Time => 6ms .. 9ms;
end Drone.impl;
```

Example: System Architecture in HybridSynchAADL (2)

- A drone component (a subset of AADL)

```
system Drone
  features
    iX: in data port Base_Types::Float;          iY: in data port Base_Types::Float;
    oX: out data port Base_Types::Float;         oY: out data port Base_Types::Float;
end Drone;

system implementation Drone.impl
  subcomponents
    ctl: system DroneControl::DroneControl.impl;
    env: system Environment::Environment.impl;
  connections
    C1: port ctl.oX -> oX;     C2: port ctl.oY -> oY;         C3: port iX -> ctl.iX;
    C4: port iY -> ctl.iY;     C5: port ctl.vX -> env.vX;     C6: port ctl.vY -> env.vY;
    C7: port env.cX -> ctl.cX; C8: port env.cY -> ctl.cY;
  properties
    Hybrid_SynchAADL::Sampling_Time => 2ms .. 4ms;
    Hybrid_SynchAADL::Response_Time => 6ms .. 9ms;
end Drone.impl;
```

Example: System Architecture in HybridSynchAADL (2)

- A drone component (a subset of AADL)

```
system Drone
  features
    iX: in data port Base_Types::Float;         iY: in data port Base_Types::Float;
    oX: out data port Base_Types::Float;        oY: out data port Base_Types::Float;
end Drone;

system implementation Drone.impl
  subcomponents
    ctl: system DroneControl::DroneControl.impl;
    env: system Environment::Environment.impl;
  connections
    C1: port ctl.oX -> oX;      C2: port ctl.oY -> oY;        C3: port iX -> ctl.iX;
    C4: port iY -> ctl.iY;      C5: port ctl.vX -> env.vX;     C6: port ctl.vY -> env.vY;
    C7: port env.cX -> ctl.cX;  C8: port env.cY -> ctl.cY;
  properties
    Hybrid_SynchAADL::Sampling_Time => 2ms .. 4ms;
    Hybrid_SynchAADL::Response_Time => 6ms .. 9ms;
end Drone.impl;
```

HybridSynchAADL annotations for environment interactions

Example: Discrete Controller in HybridSynchAADL

- A thread component for a drone controller (AADL's Behavior Annex)

```
thread implementation DroneControlThread.impl
  subcomponents
    cls: data Base_Types::Boolean;
  annex behavior_specification {**
    variables
      nx: Base_Types::Float;    ny: Base_Types::Float;
    states
      s1: initial complete state;       s2, s3: state;
    transitions
      s1 -[on dispatch]-> s2;
      s2 -[abs(cX - iX) < 0.1 and abs(cY - iY) < 0.1]-> s3 {
        vX := 0;  vY := 0;  cls := true };
      s2 -[otherwise]-> s3 {
        nx := -1 * (cX - iX);  ny := -1 * (cY - iY);
        ...
       };
      s3 -[]-> s1 { oX := cX; oY := cY }; **};
end DroneControlThread.impl;
```

Example: Environment Component in HybridSynchAADL

- An environment component

```
system Environment
  features
    cX: out data port Base_Types::Float;        cY: out data port Base_Types::Float;
    vX: in data port Base_Types::Float;         vY: in data port Base_Types::Float;
  properties
    Hybrid_SynchAADL::isEnvironment => true;
end Environment;

system implementation Environment.impl
  subcomponents
    x: data Base_Types::Float;          velx: data Base_Types::Float;
    y: data Base_Types::Float;          vely: data Base_Types::Float;
  connections
    C1: port x -> cX;   C2: port y -> cY;       C3: port vX -> velx;    C4: port vY -> vely;
  properties
    Hybrid_SynchAADL::ContinuousDynamics =>
      "x(t) = 0.001 * velx * t + x(0);
       y(t) = 0.001 * vely * t + y(0);";
end Environment.impl;
```

Example: Environment Component in HybridSynchAADL

- An environment component

```
system Environment
  features
    cX: out data port Base_Types::Float;        cY: out data port Base_Types::Float;
    vX: in data port Base_Types::Float;         vY: in data port Base_Types::Float;
  properties
    Hybrid_SynchAADL::isEnvironment => true;
end Environment;

system implementation Environment.impl
  subcomponents
    x: data Base_Types::Float;          velx: data Base_Types::Float;
    y: data Base_Types::Float;          vely: data Base_Types::Float;
  connections
    C1: port x -> cX;   C2: port y -> cY;       C3: port vX -> velx;    C4: port vY -> vely;
  properties
    Hybrid_SynchAADL::ContinuousDynamics =>
      "x(t) = 0.001 * velx * t + x(0);
       y(t) = 0.001 * vely * t + y(0);";
end Environment.impl;
```

HybridSynchAADL annotation

HybridSynchAADL annotation for continuous dynamics

Example: Specifying Properties in HybridSynchAADL

- Two properties of `FourDronesSystem`
  - safety: drones do not collide
  - rendezvous: all drones can eventually gather together

Example: Specifying Properties in HybridSynchAADL

- Two properties of `FourDronesSystem`
  - safety: drones do not collide
  - rendezvous: all drones can eventually gather together

```
invariant [safety]: ?initial ==> not ?collision in time 500;
reachability [rendezvous]: ?initial ==> ?gather in time 500;
```

Example: Specifying Properties in HybridSynchAADL

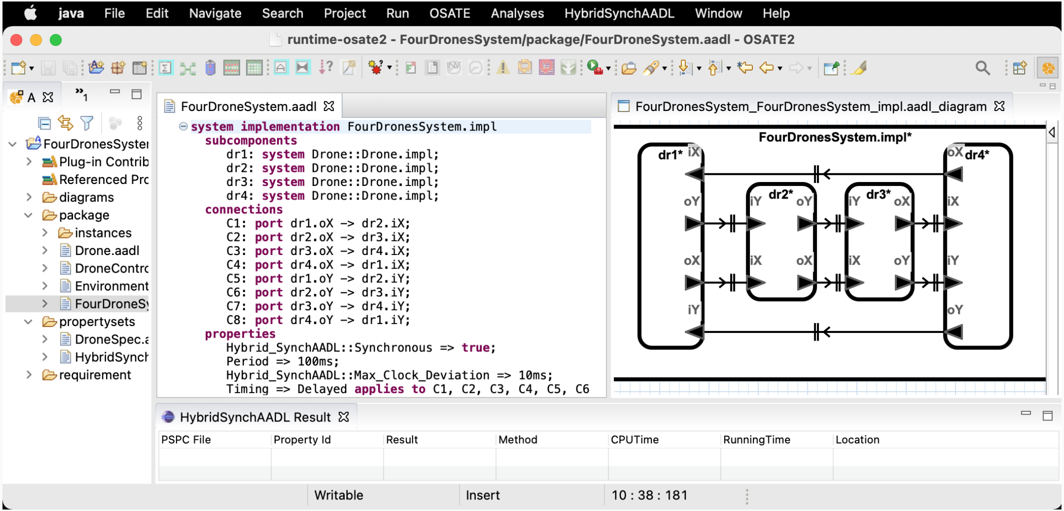
- Two properties of `FourDronesSystem`
  - safety: drones do not collide
  - rendezvous: all drones can eventually gather together

```
invariant [safety]: ?initial ==> not ?collision in time 500;
reachability [rendezvous]: ?initial ==> ?gather in time 500;
```

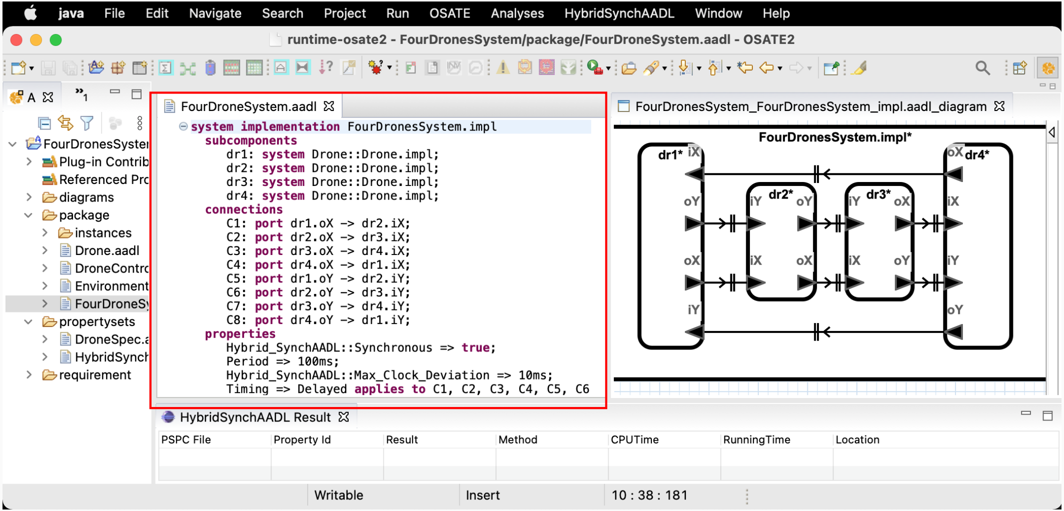
- Propositions as AADL Boolean expressions, e.g.,

```
proposition [initial] :
  abs(dr1.env.x - 1.1) < 0.01 and abs(dr1.env.y - 1.5) < 0.01 and
  abs(dr2.env.x + 1.5) < 0.01 and abs(dr2.env.y + 1.1) < 0.01 and
  abs(dr3.env.x - 1.5) < 0.01 and abs(dr3.env.y - 1.1) < 0.01 and
  abs(dr4.env.x + 1.1) < 0.01 and abs(dr4.env.y + 1.5) < 0.01;
```

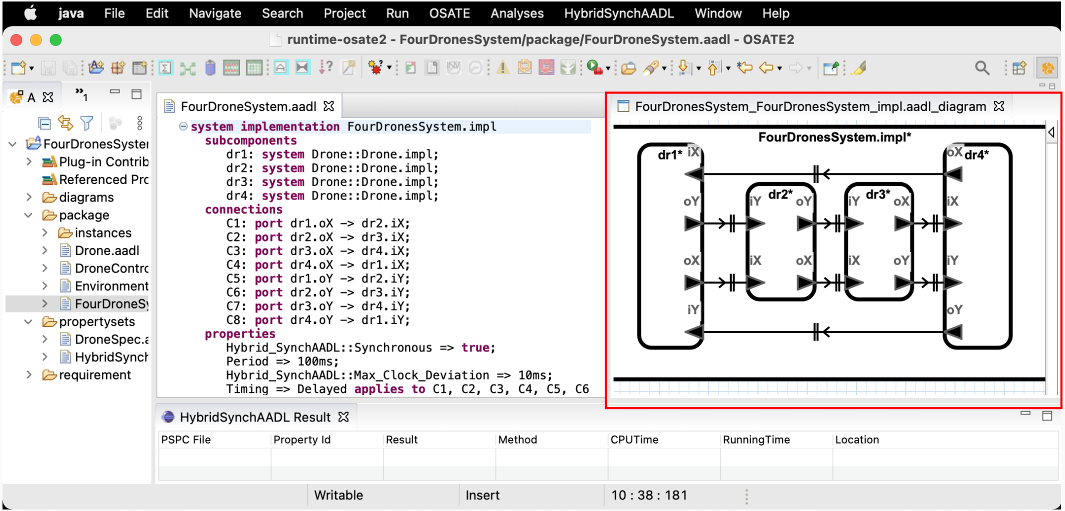
The HybridSynchAADL Tool: Example



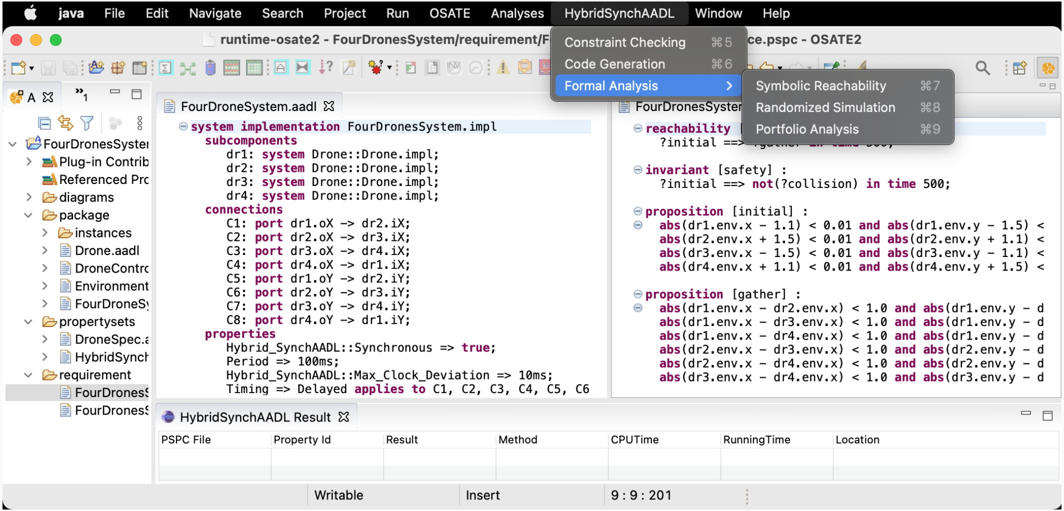
The HybridSynchAADL Tool: Example



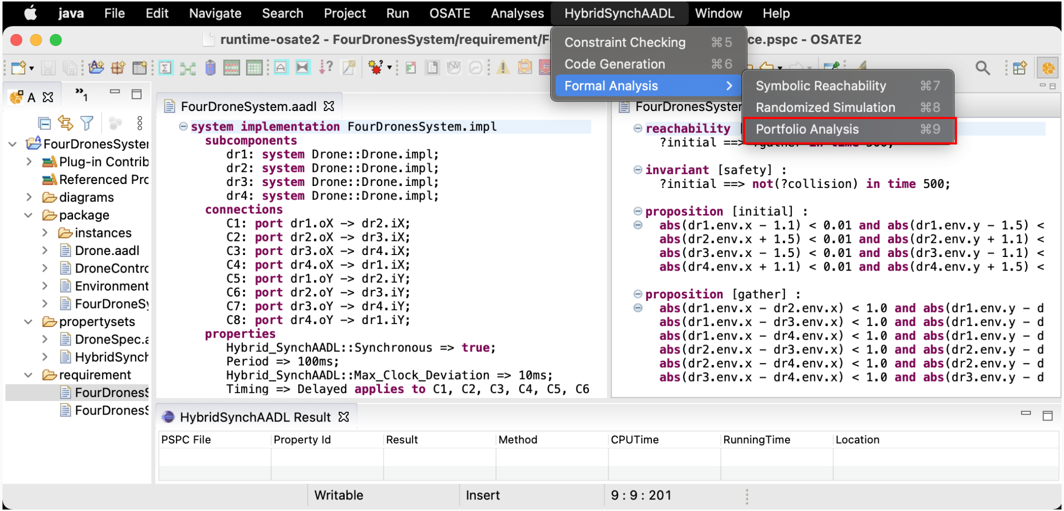
The HybridSynchAADL Tool: Example



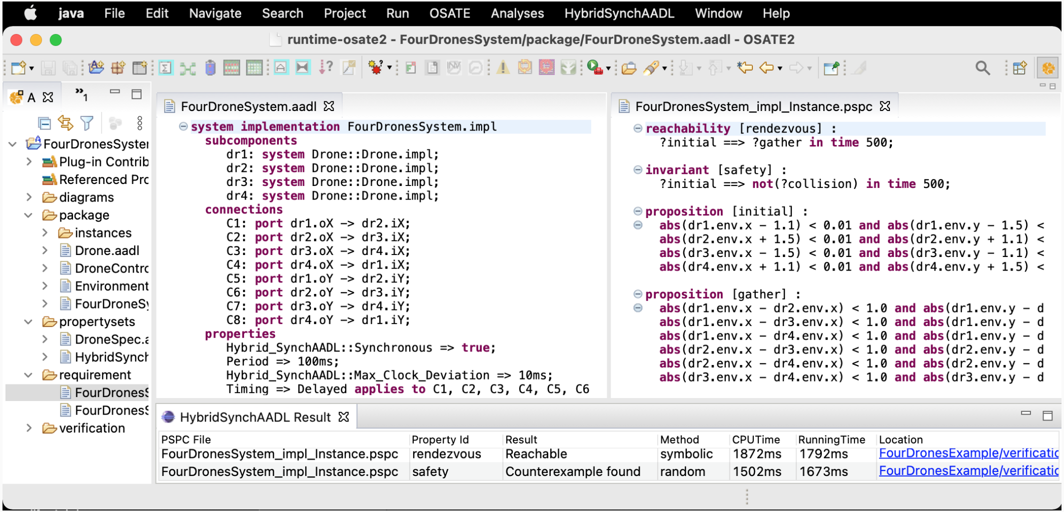
The HybridSynchAADL Tool: Example



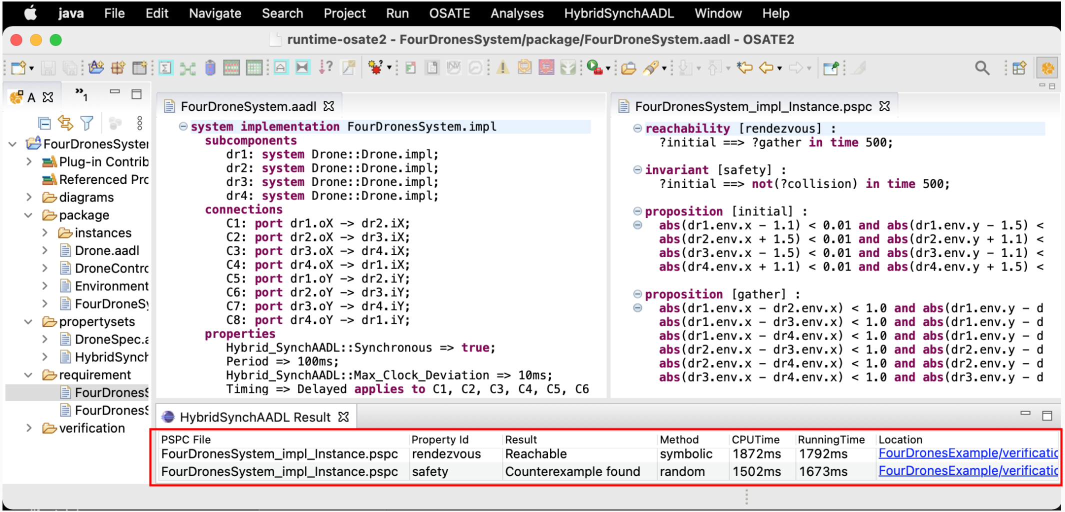
The HybridSynchAADL Tool: Example



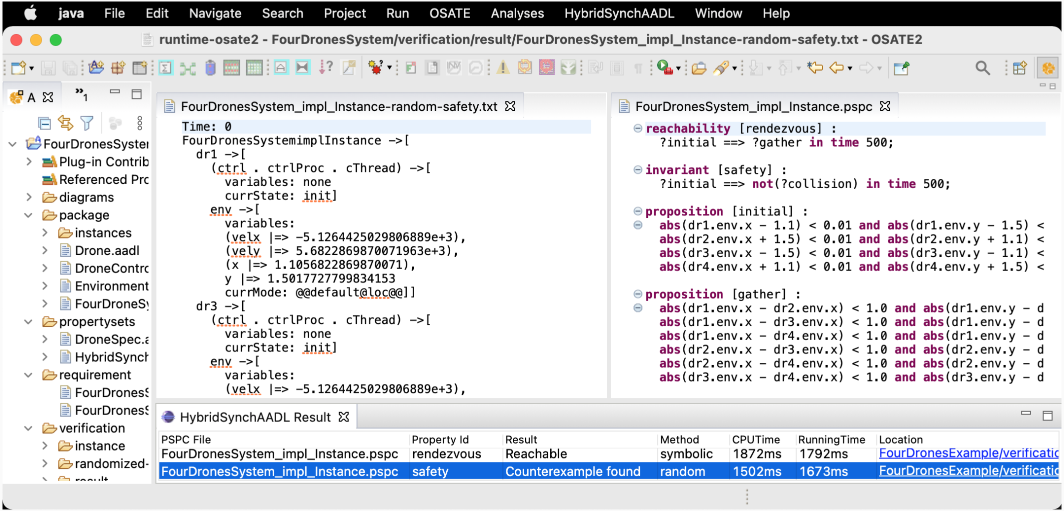
The HybridSynchAADL Tool: Example



The HybridSynchAADL Tool: Example



The HybridSynchAADL Tool: Example



Summary

Goal

Enable automated formal analysis for domain-specific modeling of virtually synchronous CPSs

Summary

Goal

Enable automated formal analysis for domain-specific modeling of virtually synchronous CPSs

- HybridSynchAADL modeling language
  - models synchronous designs in AADL
  - easy-to-use for CPS developers

Summary

Goal

Enable automated formal analysis for domain-specific modeling of virtually synchronous CPSs

- HybridSynchAADL modeling language
  - models synchronous designs in AADL
  - easy-to-use for CPS developers
- HybridSynchAADL tool
  - design and automatic formal analysis inside OSATE
  - symbolic reachability analysis using Maude and SMT

Summary

Goal

Enable automated formal analysis for domain-specific modeling of virtually synchronous CPSs

- HybridSynchAADL modeling language
  - models synchronous designs in AADL
  - easy-to-use for CPS developers
- HybridSynchAADL tool
  - design and automatic formal analysis inside OSATE
  - symbolic reachability analysis using Maude and SMT
- Hybrid PALS
  - reduces the design and verification complexity
  - synchronizer for virtually synchronous CPSs with continuous dynamics

Thank you!