HYBRIDSYNCHAADL: Modeling and Formal Analysis of Virtually Synchronous CPSs in AADL

Jaehun Lee¹, Sharon Kim¹, Kyungmin Bae¹(✉) ,
and Peter Csaba Ölveczky²

¹ Pohang University of Science and Technology, Pohang, Korea
kmbae@postech.ac.kr

² University of Oslo, Oslo, Norway

Abstract. We present the HYBRIDSYNCHAADL modeling language and formal analysis tool for virtually synchronous cyber-physical systems with complex control programs, continuous behaviors, bounded clock skews, network delays, and execution times. We leverage the Hybrid PALS equivalence, so that it is sufficient to model and verify the simpler underlying synchronous designs. We define the HYBRIDSYNCHAADL language as a sublanguage of the avionics modeling standard AADL for modeling such designs in AADL, and demonstrate the effectiveness of HYBRIDSYNCHAADL on a number of applications.

1 Introduction

Many cyber-physical systems (CPSs) are *virtually synchronous* networks of hybrid components with continuous behaviors combined with sophisticated controllers. They should logically behave as if they were synchronous—in each iteration of the system, all components, in lockstep, read inputs and perform transitions which generate outputs for the next iteration—but have to be realized in a distributed setting, with clock skews and message passing communication. Examples of such CPSs include avionics and automotive systems [34, 42], networked medical devices [5, 30], and other distributed control systems such as the steam-boiler benchmark [1], where the underlying infrastructure often guarantees bounds on clock skews, network delays, and local execution times.

The uptake of automated formal analysis of such CPSs is challenging, since:

1. The combination of large “discrete” state spaces, caused by interleavings due to asynchronous communication, and continuous behaviors, taking into account clock skews, network delays, and sampling/actuation times (based on imprecise clocks) makes direct automatic model checking analysis infeasible.
2. To enable formal analysis to a large user base, the modeling language for such CPSs, with complex control programs, should be well-known for CPS developers, and should be integrated into mature modeling environments.

To confront these challenges, we present in this paper the HYBRIDSYNCHAADL modeling language and analysis tool, which address them as follows:

1. To dramatically reduce both the modeling complexity and the state space caused by asynchronous communication, we use the Hybrid PALS equivalence [8], which says that the underlying synchronous design—where all components execute in lockstep, and there is no asynchronous message passing—satisfies the same properties as the asynchronous distributed system.
2. The HYBRIDSYNCHAADL modeling language is a subset of the avionics modeling standard AADL [22] and its behavioral annex to model control programs, and captures a synchronous subset of AADL with continuous behaviors. We have also integrated modeling and formal analysis of HYBRIDSYNCHAADL models into the OSATE modeling environment for AADL.

Providing formal semantics and analysis for HYBRIDSYNCHAADL, with its expressive control program formalism, continuous behaviors, and clock skews, and having to cover all possible continuous behaviors based on imprecise clocks, is challenging. We combine Maude [19] and the SMT solver Yices [21] to provide such a semantics, as well as symbolic reachability analysis of bounded invariant properties. To make the analysis feasible, our tool also implements a state-space reduction method that merges symbolic states for Maude-with-SMT to significantly improve the performance of symbolic reachability analysis. We illustrate the use of the HYBRIDSYNCHAADL language and tool—and compare its effectiveness with other state-of-the-art CPS analysis tools—on a number of hybrid CPS applications, including distributed drones that communicate to reach the “same” location, or fly in formation, without crashing into each other.

Our tool extends the SynchAADL tool [7, 9, 10] for distributed *real-time* systems without continuous behaviors, where the time when an event takes place can be abstracted away, so there is no need to consider clock skews, and any (sufficiently expressive) explicit-state model checker can be applied. In contrast, HYBRIDSYNCHAADL must model continuous behaviors and clock skews, and must analyze all possible behaviors based on when the continuous components are sampled and actuated, which depend on the imprecise local clocks. The tool is available at <https://hybridsynchaadl.github.io>.

2 Preliminaries

PALS and Hybrid PALS. When the infrastructure guarantees bounds Γ on clock skews, network delays, and execution times, the PALS pattern [4, 36] reduces the problems of designing and verifying virtually synchronous distributed real-time systems to the much simpler problems of designing and verifying their underlying synchronous designs: Given a synchronous system design SD , bounds Γ , and a period p of each round, the PALS transformation gives the asynchronous distributed real-time system $PALS(SD, \Gamma, p)$, which is stuttering bisimilar to SD .

The synchronous design SD is formalized as the synchronous composition of state machines with input and output ports [36]. In each iteration, all machines simultaneously perform a transition, which includes reading inputs, changing the local state, and generating outputs (for the next iteration).

Hybrid PALS [8] extends PALS to virtually synchronous CPSs with physical environments that exhibit continuous behaviors. The *physical environment* E_M of a machine M has real-valued parameters $\vec{x} = (x_1, \dots, x_l)$. The continuous behaviors of $\vec{x}$ are modeled by ordinary differential equations (ODEs) that specify different *trajectories* on $\vec{x}$. E_M also defines *which* trajectory the environment follows, as a function of the last *control command* received by E_M .

The local clock of a machine M can be seen as a function $c_M : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$, where $c_M(t)$ is the value of the local clock at time t , with $\forall t \in \mathbb{R}_{\geq 0}, |c_M(t) - t| < \epsilon$ for $\epsilon > 0$ the maximal clock skew [36]. In its i th iteration, a controller M samples the values of its environment at time $c_M(i \cdot p) + t_s$, where t_s is the *sampling time*, and then executes a transition. As a result, the new control command is received by the environment at time $c_M(i \cdot p) + t_a$, where t_a is the *actuating time*.

AADL. The *Architecture Analysis & Design Language* (AADL) [22] is an industrial modeling standard used in avionics, aerospace, automotive, medical devices, and robotics to describe an embedded real-time system. In AADL, a component *type* specifies the component's *interface* (e.g., ports) and *properties* (e.g., periods), and a component *implementation* specifies its internal structure as a set of *subcomponents* and a set of *connections* linking their ports. An AADL construct may have *properties* describing its parameters, declared in *property sets*. The OSATE modeling environment provides a set of Eclipse plug-ins for AADL.

An AADL model describes a system of hardware and software components. Software components include *threads* that model the application software and *data* components representing data types. *System* components are the top-level components. A port is a *data* port, an *event* port, or an *event data* port. A component can have different *modes* and mode-specific property values, sub-components, etc. Mode transitions are triggered by events.

Thread behavior is modeled as a guarded transition system with local variables using AADL's *Behavior Annex* [23]. When a thread is activated, transitions are applied until a *complete* state is reached. The *dispatch protocol* determines when a thread is executed. A *periodic* thread is activated at fixed time intervals.

Maude with SMT. Maude [19] is a language and tool for formally specifying and analyzing distributed systems in rewriting logic. System states are specified as elements of algebraic data types, and transitions are specified using rewrite rules. In addition to its explicit-state analysis methods for concrete states, Maude provides SMT solving and *symbolic reachability analysis* for *constrained terms* $\phi \parallel t$, which symbolically represent all instances of the term $t(x_1, \dots, x_n)$ satisfying the constraint $\phi(x_1, \dots, x_n)$ [40], using connections to Yices2 [21] and CVC4 [14].

3 The HYBRIDSYNCHAADL Modeling Language

This section presents the HYBRIDSYNCHAADL language for modeling virtually synchronous CPSs in AADL. HYBRIDSYNCHAADL can specify environments with continuous dynamics, synchronous designs of distributed controllers, and

nontrivial interactions between controllers and environments with respect to imprecise local clocks and sampling and actuation times.

The HYBRIDSYNCHAADL language is a subset of AADL extended with property set `Hybrid_SynchAADL`. We use a subset of AADL without changing the meaning of AADL constructs or adding a new annex—the subset has the same meaning for synchronous models and distributed implementations—so that AADL experts can easily develop and understand HYBRIDSYNCHAADL models.

```
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;
```

Environment Components. An *environment component* models real-valued state variables that continuously change over time. State variables are specified using data subcomponents of type `Base_Types::Float`. Each environment component declares the property `Hybrid_SynchAADL::isEnvironment => true`.

An environment component can have different *modes* to specify different continuous behaviors (trajectories). A controller command may change the mode of the environment or the value of a variable. The continuous dynamics in each mode is specified using either ODEs or continuous real functions as follows:

```
Hybrid_SynchAADL::ContinuousDynamics =>
  "dynamics1" in modes (mode1), ..., "dynamicsn" in modes (moden);
```

In HYBRIDSYNCHAADL, a set of ODEs over n variables $x_1, \dots, x_n$, say, $\frac{dx_i}{dt} = e_i(x_1, \dots, x_n)$ for $i = 1, \dots, n$, is written as a semicolon-separated string:

```
d/dt(x1) = e1(x1, ..., xn); ... ; d/dt(xn) = en(x1, ..., xn);
```

If a closed-form solution of ODEs is known, we can directly specify concrete continuous functions, which are parameterized by a time parameter t and the initial values $x_1(0), \dots, x_n(0)$ of the variables $x_1, \dots, x_n$:

```
x1(t) = e1(t, x1(0), ..., xn(0)); ... ; xn(t) = en(t, x1(0), ..., xn(0));
```

An environment component interacts with discrete controllers by sending its state values, and by receiving actuator commands that may update state variables or trigger mode (and hence trajectory) changes. This behavior is specified in HYBRIDSYNCHAADL using *connections between ports and data subcomponents*. A connection from a data subcomponent d inside the environment to an output data port o declares that the value of d is “sampled” by a controller. A connection from an environment’s input port i to d declares that a controller command arrived at i updates the value of the data subcomponent d .

Controller Components. Discrete controllers are usual software components in the Synchronous AADL subset [7, 9]. A controller component is specified using the behavioral and structural subset of AADL: hierarchical system, process, thread components, and thread behaviors defined by the Behavior Annex [23].

A controller receives the state of the environment at some *sampling time*, and sends a controller command to the environment at some *actuation time*. Sampling and actuation take place according to the local clock of the controller.

```
Hybrid_SynchAADL::Max_Clock_Deviation => time;
Hybrid_SynchAADL::Sampling_Time => lower bound .. upper bound;
Hybrid_SynchAADL::Response_Time => lower bound .. upper bound;
```

The top-level system component declares the following properties to state that the entire model is a synchronous design with a period T :

```
Hybrid_SynchAADL::Synchronous => true;      Period => T;
```

Communication. In HYBRIDSYNCHAADL, connections are constrained for synchronous behaviors: no connection is allowed between environments, or between environments and the enclosing system components.

All (non-actuator) outputs of controller components generated in an iteration are available to the receiving *controller* components at the beginning of the *next* iteration. As explained in [7, 9], *delayed connections between data ports* meet this requirement. Therefore, two controller components can be connected only by data ports with delayed connections: `Timing => Delayed`.

Interactions between a controller and an environment occur *instantaneously* at the sampling and actuating times of the controller. Because an environment does not “actively” send data for sampling, every output port of an environment must be a *data* port, whereas its input ports could be of any kind.

4 The HYBRIDSYNCHAADL Tool

This section introduces the HYBRIDSYNCHAADL tool supporting the modeling and formal analysis of HYBRIDSYNCHAADL models. The tool is an OSATE plugin which: (i) provides an intuitive language to specify properties of models, (ii) synthesizes a rewriting logic model from a HYBRIDSYNCHAADL model, and (iii) performs various formal analyses using Maude combined with SMT solving.

Specifying Properties. The tool’s *property specification language* allows the user to specify time-bounded invariant and reachability properties as propositional formulas whose atomic propositions are AADL Boolean expressions.

A “named” atomic proposition can be declared in HYBRIDSYNCHAADL as follows, where each identifier is fully qualified with its component path:

```
proposition [id]: AADL Boolean Expression
```

Fig. 1. Interface of the HYBRIDSYNCHAADL tool.

The following *named invariant property* holds if, for every (initial) state satisfying the initial condition φ_{init} , all states reachable within the time bound τ_{bound} satisfy the invariant condition φ_{inv} .

```
invariant [name]:  $\varphi_{init} \implies \varphi_{inv}$  in time  $\tau_{bound}$ 
```

A *reachability property* (the dual of an invariant) holds if a state satisfying φ_{goal} is reachable from some state satisfying φ_{init} within the time bound τ_{bound} .

```
reachability [name]:  $\varphi_{init} \implies \varphi_{goal}$  in time  $\tau_{bound}$ 
```

Tool Interface. The tool first statically checks whether a given model is a valid model that satisfies the syntactic constraints of HYBRIDSYNCHAADL.

HYBRIDSYNCHAADL provides two analysis methods. *Symbolic reachability analysis* can verify that all possible behaviors satisfy a given requirement;¹ if not, a counterexample is generated. *Randomized simulation* repeatedly executes the model until a counterexample is found, by randomly choosing concrete sampling and actuating times, nondeterministic transitions, etc.

Our tool also provides *portfolio analysis* that combines symbolic reachability analysis and randomized simulation. HYBRIDSYNCHAADL runs both methods in parallel using multithreading, and displays the result of the analysis that terminates first. Symbolic analysis can guarantee the absence of a counterexample, whereas randomized simulation is effective for finding “obvious” bugs.

Figure 1 shows the interface of our tool that is fully integrated into OSATE. The left editor shows the code of `FourDronesSystem` in Sect. 5, the bottom right editor shows its graphical representation, and the top right editor shows two properties in the property specification language. The HYBRIDSYNCHAADL menu contains three items for constraint checking, code generation, and formal analysis. The **Portfolio Analysis** item has already been clicked, and the **Result** view at the bottom displays the analysis results in a readable format.

¹ Symbolic analysis currently only supports polynomial continuous dynamics, since the Yices2 SMT solver does not support general classes of ODEs.

Tool Implementation. We have (in our report [33]) developed a Maude-with-SMT semantics for HYBRIDSYNCHAADL that formalizes our modeling language and implements our tool’s analysis commands. Maude is suitable to capture the expressive control program language, the hierarchical structure of systems, and communication. Symbolic rewriting with SMT allows us to analyze infinite states and all possible behaviors caused by sampling and actuation times with imprecise clocks, where continuous dynamics can be encoded in SMT [18, 26].

Nontrivial control programs with many conditional branches and guarded transitions typically involve a large number of symbolic states; to reduce the number of results of executing one iteration of the system, we have implemented a *state merging* optimization technique [11] that merges two symbolic states into one using disjunction and generalization. As shown in the report [33], this state merging dramatically improves the performance of symbolic analysis and makes the formal analysis feasible for such distributed hybrid systems.

The HYBRIDSYNCHAADL tool uses OSATE’s code generation facilities to synthesize a Maude model from the HYBRIDSYNCHAADL model. It then invokes Maude and an SMT solver to check whether the model satisfies given invariant and reachability requirements. Our tool is implemented in around 6,200 lines of Maude code and around 8,600 lines of Java and Xtend code.

5 Case Study: Collaborating Autonomous Drones

This section shows how virtually synchronous CPSs for controlling distributed drones—which collaborate to achieve common goals, such as rendezvous and formation control—can be modeled and analyzed in HYBRIDSYNCHAADL.

Rendezvous of Multiple Drones. Consider N drones, where vectors $\vec{x}_i$ and $\vec{v}_i$, for $1 \leq i \leq N$, denote the position and velocity of the i -th drone. The continuous dynamics of the i -th drone is specified by the ordinary differential equation $\dot{\vec{x}}_i = \vec{v}_i$. The controller samples the drone’s position and velocity, and gives the new velocity value to the environment as a control command. The goal of rendezvous is for all drones to arrive near a common location simultaneously.

Figure 2 shows the AADL architecture of our rendezvous model for four drones. Each drone is connected to two other drones to exchange positions. A drone component consists of an environment (with the drone’s position and velocity) and its controller. Figure 3 shows the implementation of the top-level component, a Drone system component, an Environment system component, and a thread component for a drone controller in HYBRIDSYNCHAADL.

In each round, the controller obtains the position $\vec{x}$ from its environment at its sampling time. The position of the connected drone was sent in the previous round. The controller determines a new velocity to synchronize its movement with the other drones using a distributed consensus algorithm [39]. The environment changes its position according to the velocity indicated by its controller, where the new velocity $\vec{v}$ becomes effective at its actuation time.

Fig. 2. The AADL architecture of four drones (left), and a drone component (right).

```

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

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;

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;
    y: data Base_Types::Float;
    velx: 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;

thread DroneControlThread
  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;
    cX: in data port Base_Types::Float;
    cY: in data port Base_Types::Float;
    vX: out data port Base_Types::Float;
    vY: out data port Base_Types::Float;
  properties
    Dispatch_Protocol => Periodic;
end DroneControlThread;

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);
      if (nx > 0.3) vX := 2.5
      elsif (nx > 0.15)
        if (cls) vX := 1.5 else vX := 0.0 end if
      else vX := -2.5 end if;
      if (ny > 0.3) vY := 2.5
      elsif (ny > 0.15)
        if (cls) vY := 1.5 else vY := 0.0 end if
      else vY := -2.5 end if; cls := false };
    s3 -[]-> s1 { oX := cX; oY := cY }; **};
end DroneControlThread.impl;

```

Fig. 3. A HYBRIDSYNCHAADL model for four distributed drones.

Verification. We analyze the following properties up to bound 500 ms using HYBRIDSYNCHAADL portfolio analysis: (i) drones do not collide (**safety**), and (ii) all drones can eventually gather together (**rendezvous**).

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

We define three propositions: **initial**, defining the range of initial positions of the four drones **dr1**, **dr2**, **dr3**, and **dr4**; **collision**, where two drones collide if the (horizontal and vertical) distance between them is less than 0.1; and **gather**, indicating that the distance between each pair of drones is less than 1. For example, **collision** and **initial** are defined as follows.

```
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;

proposition [collision] :
  (abs(dr1.env.x - dr2.env.x) < 0.1 and abs(dr1.env.y - dr2.env.y) < 0.1) or
  ...
  (abs(dr3.env.x - dr4.env.x) < 0.1 and abs(dr3.env.y - dr4.env.y) < 0.1);
```

The analysis result is shown in the **Result** view at the bottom of Fig. 1. There is a witness for **rendezvous**, obtained by symbolic reachability analysis in 1.7 seconds. A counterexample for **safety** is found by randomized simulation in 1.5 seconds, since **initial** does not constrain the speed of the drones. In [33], we add initial velocity constraints, and verify that **safety** holds up to the time bound by symbolic reachability analysis in 15 minutes.

6 Experimental Evaluation

We compare the performance of HYBRIDSYNCHAADL’s symbolic analysis with four reachability analysis tools for hybrid automata, HyComp [18], SpaceEx [24], Flow* [17], and dReach [31], on models of rendezvous and formation control for distributed drones, and on networked thermostats (adapted from [6, 29]). We use *simplified* models with less complex control; otherwise, most of the other tools time out (see [33] for results on more complex models). We use two invariant properties for each model: $Inv_{\top}$, which holds, and $Inv_{\perp}$, which does not hold.

To use the other tools, we have “encoded” the *synchronous designs* of the HYBRIDSYNCHAADL models as networks of hybrid automata. Each component is modeled as a hybrid automaton with three modes: starting a new round, sampling, and controller transition/actuation. The behavior of a controller is encoded as single jumps. We use flat hybrid automata (obtained by HYST [12]) for Flow* and dReach, which do not support networks of hybrid automata.

We measure the execution times for analyzing the invariant properties *up to* bound 500 ms, with a timeout of 60 minutes. For HYBRIDSYNCHAADL, we use

Table 1. HYBRIDSYNCHAADL vs. HyComp, SpaceEx, dReach, and Flow*.

Model	Tool	$Inv_{\top}$			$Inv_{\perp}$		
		$N = 2$	$N = 3$	$N = 4$	$N = 2$	$N = 3$	$N = 4$
		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

a specialized version of Maude with Yices 2.6 for polynomial arithmetic [44]. For SpaceEx, we use PHAVer for linear dynamics, and STC for nonlinear polynomial dynamics. For Flow*, we use adaptive steps, and TM orders 1 (for single) and 2 (for double). We use the default precision for dReach, and BMC for HyComp. We have run all experiments on Intel Xeon 2.8GHz with 256 GB memory.

The results are summarized in Table 1, as execution times (seconds) over time bounds ($B \cdot 100$ ms), with N the number of components. The results for “Rend (double)” (rendezvous with double-integrator dynamics, where control input is given by acceleration instead of velocity) do not include HyComp, which does not support nonlinear polynomial dynamics. For $Inv_{\top}$, Table 1 shows the largest time bound for which the tool could prove the absence of counterexamples. Often, *tools timed out when trying to verify that $Inv_{\top}$ holds up to time bound 500*.² For $Inv_{\perp}$, the table shows the smallest bound for which the tool found counterexamples.³ As seen, HYBRIDSYNCHAADL outperforms the other tools in most cases, in particular for complex models with larger N .

² E.g., for “Rend (single)” with $N = 4$, HYBRIDSYNCHAADL needs 5.8 seconds for $B = 5$, whereas SpaceEx needs 4.5 seconds for $B = 1$ and timed out for $B > 1$.

³ Flow* occasionally found (spurious) counterexamples at smaller bounds, because of over-approximation by the Taylor model flowpipe construction.

7 Related Work

Our tool can model check virtually synchronous CPSs with both complex control programs and continuous behaviors (and imprecise local clocks, etc.), whereas most formal tools are strong at analyzing either discrete or continuous behaviors. The latter includes analysis tools for hybrid automata [13, 18, 24], which do not deal well with the “discrete complexity” (e.g., control programs) of CPSs.

The *Hybrid Annex* [2, 3] allows specifying continuous behaviors in AADL, but message delays, clock skews, etc., are not taken into account. Controllers are defined in Hybrid CSP instead of in AADL’s convenient Behavioral Annex. Another hybrid annex is proposed in [38], and an AADL sublanguage, AADL+, in [35]. None of these languages support automated formal correctness analysis.

Hybrid PALS models with simple controllers are encoded as logical formulas and analyzed by dReal in [8]. However, there is no tool support, and so CPSs must be *manually* modeled as SMT formulas in [8]. In contrast, we provide a tool for modeling Hybrid PALS models using a well-known modeling standard.

Our work is also related to *almost-synchronous* systems, including approximate synchrony [20], quasi-synchrony [15, 16, 28, 32], GALS [27, 37], time-triggered architectures [41, 43], etc. Our method makes it possible to verify such systems *with continuous behaviors*, which are typically not considered in related work.

8 Concluding Remarks

We have presented the HYBRIDSYNCHAADL modeling language and formal analysis tool for modeling and analyzing the *synchronous designs*—and, by the Hybrid PALS equivalence, also the corresponding *asynchronous distributed real-time system* with bounded clock skews, network delays, and execution times—of virtually synchronous networks of hybrid systems with potentially complex control programs in the modeling standard AADL. Our tool provides randomized simulation and symbolic reachability analysis, and is fully integrated into the OSATE modeling environment for AADL. We have shown that in most cases, HYBRIDSYNCHAADL’s symbolic analysis outperforms state-of-the-art hybrid systems reachability analysis tools on a number of distributed hybrid systems.

Currently, HYBRIDSYNCHAADL’s symbolic analysis is restricted to systems with (nonlinear) polynomial continuous dynamics, because the underlying SMT solver, Yices2, cannot deal with general classes of ODEs. We should therefore integrate Maude with ODE solvers such as dReal [25] and Flow* [17].

Acknowledgments. This work was supported in part by the National Research Foundation of Korea (NRF) grants funded by the Korea government (MSIT) (No. 2019R1C1C1002386 and No. 2017M3C4A7068175).

References

1. Abrial, J.-R., Börger, E., Langmaack, H. (eds.): Formal Methods for Industrial Applications. LNCS, vol. 1165. Springer, Heidelberg (1996). <https://doi.org/10.1007/BFb0027227>
2. Ahmad, E., Dong, Y., Wang, S., Zhan, N., Zou, L.: Adding formal meanings to AADL with Hybrid Annex. In: Lanese, I., Madelaine, E. (eds.) FACS 2014. LNCS, vol. 8997, pp. 228–247. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15317-9_15
3. Ahmad, E., Larson, B.R., Barrett, S.C., Zhan, N., Dong, Y.: Hybrid Annex: An AADL extension for continuous behavior and cyber-physical interaction modeling. In: Proceedings of ACM SIGAda HILT 2014. ACM (2014)
4. Al-Nayeem, A., Sun, M., Qiu, X., Sha, L., Miller, S.P., Cofer, D.D.: A formal architecture pattern for real-time distributed systems. In: Proceedings of RTSS. IEEE (2009)
5. Arney, D., Jetley, R., Jones, P., Lee, I., Sokolsky, O.: Formal methods based development of a PCA infusion pump reference model: Generic infusion pump (GIP) project. In: Proceedings of HCMDSS-MDPnP 2007. IEEE (2007)
6. Bae, K., Gao, S.: Modular SMT-based analysis of nonlinear hybrid systems. In: Proceedings of FMCAD, pp. 180–187. IEEE (2017)
7. Bae, K., Ölveczky, P.C., Al-Nayeem, A., Meseguer, J.: Synchronous AADL and its formal analysis in Real-Time Maude. In: Qin, S., Qiu, Z. (eds.) ICFEM 2011. LNCS, vol. 6991, pp. 651–667. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24559-6_43
8. Bae, K., Ölveczky, P.C., Kong, S., Gao, S., Clarke, E.M.: SMT-based analysis of virtually synchronous distributed hybrid systems. In: Proceedings of HSCC. ACM (2016)
9. Bae, K., Ölveczky, P.C., Meseguer, J.: Definition, semantics, and analysis of Multirate Synchronous AADL. In: Jones, C., Pihlajasaari, P., Sun, J. (eds.) FM 2014. LNCS, vol. 8442, pp. 94–109. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06410-9_7
10. Bae, K., Ölveczky, P.C., Meseguer, J., Al-Nayeem, A.: The SynchAADL2Maude tool. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 59–62. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28872-2_4
11. Bae, K., Rocha, C.: Symbolic state space reduction with guarded terms for rewriting modulo SMT. *Sci. Comput. Program.* **178**, 20–42 (2019)
12. Bak, S., Bogomolov, S., Johnson, T.T.: HYST: A source transformation and translation tool for hybrid automaton models. In: Proceedings of HSCC 2015. ACM (2015)
13. Bak, S., Duggirala, P.S.: Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In: Proceedings of HSCC 2017. ACM (2017)
14. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_14
15. Baudart, G., Bourke, T., Pouzet, M.: Soundness of the quasi-synchronous abstraction. In: Proceedings of FMCAD, pp. 9–16. IEEE (2016)
16. Caspi, P., Mazuet, C., Paligot, N.R.: About the design of distributed control systems: The quasi-synchronous approach. In: Voges, U. (ed.) SAFECOMP 2001. LNCS, vol. 2187, pp. 215–226. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45416-0_21

17. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_18
18. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: HyCOMP: An SMT-based model checker for hybrid systems. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 52–67. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_4
19. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007). <https://doi.org/10.1007/978-3-540-71999-1>
20. Desai, A., Seshia, S.A., Qadeer, S., Broman, D., Eidson, J.C.: Approximate synchrony: An abstraction for distributed almost-synchronous systems. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9207, pp. 429–448. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21668-3_25
21. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_49
22. Feiler, P.H., Gluch, D.P.: Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis and Design Language. Addison-Wesley (2012)
23. França, R., Bodeveix, J.P., Filali, M., Rolland, J.F., Chemouil, D., Thomas, D.: The AADL Behaviour Annex - experiments and roadmap. In: ICECCS. IEEE (2007)
24. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_30
25. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 208–214. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_14
26. Gao, S., Kong, S., Clarke, E.M.: Satisfiability modulo ODEs. In: Proceedings of FMCAD. IEEE (2013)
27. Girault, A., Ménier, C.: Automatic production of globally asynchronous locally synchronous systems. In: Sangiovanni-Vincentelli, A., Sifakis, J. (eds.) EMSOFT 2002. LNCS, vol. 2491, pp. 266–281. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45828-X_20
28. Halbwachs, N., Mandel, L.: Simulation and verification of asynchronous systems by means of a synchronous model. In: Proceedings of ACSD 2006. IEEE (2006)
29. Henzinger, T.: The theory of hybrid automata. In: Inan, M.K., Kurshan, R.P. (eds.) Verification of Digital and Hybrid Systems. NATO ASI Series, vol. 170, pp. 265–292. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-642-59615-5_13
30. Kim, C., Sun, M., Mohan, S., Yun, H., Sha, L., Abdelzaher, T.F.: A framework for the safe interoperability of medical devices in the presence of network failures. In: Proceedings of ICCPS 2010. ACM (2010)
31. Kong, S., Gao, S., Chen, W., Clarke, E.: dReach: δ -reachability analysis for hybrid systems. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 200–205. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_15
32. Larrieu, R., Shankar, N.: A framework for high-assurance quasi-synchronous systems. In: Proceedings of MEMOCODE 2014. IEEE (2014)

33. Lee, J., Kim, S., Bae, K., Ölveczky, P.C.: HybridSynchAADL: Modeling and formal analysis of virtually synchronous CPSs in AADL. manuscript, January 2021. <https://hybridsynchaadl.github.io/docs/techrep.pdf>
34. Leen, G., Heffernan, D., Dunne, A.: Digital networks in the automotive vehicle. *Comput. Control Eng. J.* **10**(6), 257–266 (1999)
35. Liu, J., Li, T., Ding, Z., Qian, Y., Sun, H., He, J.: AADL+: A simulation-based methodology for cyber-physical systems. *Front. Comput. Sci.* **13**(3), 516–538 (2018). <https://doi.org/10.1007/s11704-018-7039-7>
36. Meseguer, J., Ölveczky, P.C.: Formalization and correctness of the PALS architectural pattern for distributed real-time systems. *Theor. Comput. Sci.* **451**, 5–27 (2012)
37. Potop-Butucaru, D., Caillaud, B.: Correct-by-construction asynchronous implementation of modular synchronous specifications. *Fundam. Inform.* **78**(1), 131–159 (2007)
38. Qian, Y., Liu, J., Chen, X.: Hybrid AADL: A sublanguage extension to AADL. In: *Proceedings of MEMOCODE 2014*. ACM (2013)
39. Ren, W., Beard, R.W.: *Distributed Consensus in Multi-vehicle Cooperative Control*. Springer, London (2008). <https://doi.org/10.1007/978-1-84800-015-5>
40. Rocha, C., Meseguer, J., Muñoz, C.: Rewriting modulo SMT and open system analysis. *J. Log. Algebraic Methods Program.* **86**(1), 269–297 (2017)
41. Rushby, J.: Systematic formal verification for fault-tolerant time-triggered algorithms. *IEEE Trans. Software Eng.* **25**(5), 651–660 (1999)
42. Steiner, W., Bauer, G., Hall, B., Paulitsch, M., Varadarajan, S.: TTEthernet dataflow concept. In: *2009 Eighth IEEE International Symposium on Network Computing and Applications*, pp. 319–322. IEEE (2009)
43. Tripakis, S., Pinello, C., Benveniste, A., Sangiovanni-Vincent, A., Caspi, P., Di Natale, M.: Implementing synchronous models on loosely time triggered architectures. *IEEE Trans. Comput.* **57**(10), 1300–1314 (2008)
44. Yu, G., Bae, K.: Maude-SE: A tight integration of Maude and SMT solvers. In: *Proceedings of International Workshop on Rewriting Logic and Its Applications* (2020)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.