Formal Specification and Verification of Distributed Cyber-Physical Systems

Kyungmin Bae

University of Illinois at Urbana-Champaign

December 31, 2013

1Collaboration with: Peter Ölveczky (University of Oslo), José Meseguer, Abdullah Al-Nayeem, Joshua Krisiloff, and Lui Sha (UIUC), and Steve Miller and Darren Cofer (Rockwell-Collins)
Distributed Cyber-Physical Systems (DCPS)

- collection of components that control physical entities
- complex interaction of embedded systems and real-time control
- e.g., avionics, automotive, medical devices, ...
- **safety-critical** systems
- asynchronous communications
- hard real-time constraints
- often **virtually synchronous**
  - in each period, read input, perform transition, and produce output
Challenges (1)

- Hard to design correctly
  - race conditions
  - clock skews
  - network delays and execution times

- No fault found
  - hard to duplicate a (reported) failure
  - due to distributed nature
Challenges (2)

- Model checking
  - examines all possible behaviors from the initial states
  - provides a counterexample

- Hard to model check
  - real-time
  - state space explosion due to asynchrony
Our Approach

1. **Multirate PALS**
   - reduces design and verification of a DCPS to its synchronous version

2. **Multirate Synchronous AADL**
   - makes PALS available in avionics modeling standard AADL
   - formal semantics in (real-time) rewriting logic

3. **The MR-SynchAADL tool**
   - Eclipse plug-in for Multirate Synchronous AADL
Outline

1. Multirate PALS
2. Multirate Synchronous AADL
3. Case Study: Turning an Airplane
Multirate PALS (1)

PALS: physically asynchronous logically synchronous

*Reduces design/verification of DRTS to its synchronous version*

- Relies on asynchronous bounded delay (ABD) network infrastructure
- Assumes underlying clock synchronization (IEEE 1588, etc.)
Multirate PALS (2)

PALS: physically asynchronous logically synchronous

Reduces design/verification of DRTS to its synchronous version

- Multirate PALS gives a transformation $(\mathcal{E}, T, \Gamma) \rightarrow \mathcal{MA}(\mathcal{E}, T, \Gamma)$
  - $\mathcal{E}$: multi-rate synchronous design
  - $T$: a rate function
  - $\Gamma$: bounds on network delay, execution time, and clock skew
  - $\mathcal{MA}(\mathcal{E}, T, \Gamma)$: the corresponding distributed asynchronous design

Correct by construction

$E|_\phi = \phi$ if and only if $MA(E, T, \Gamma)|_\phi = \phi$

Verified formal architectural pattern: verification effort amortized over many systems!
PALS: physically asynchronous logically synchronous

Reduces design/verification of DRTS to its synchronous version

- Multirate PALS gives a transformation \((\mathcal{E}, T, \Gamma) \rightarrow \mathcal{MA}(\mathcal{E}, T, \Gamma)\)
- Correct by construction
  \(\mathcal{E} \models \varphi \iff \mathcal{MA}(\mathcal{E}, T, \Gamma) \models \varphi\)
- Verified formal architectural pattern
  - verification effort amortized over many systems!
Synchronous Model (1)

Synchronous composition of typed state machines

Controller periods multiple of faster periods

All components must perform in lock-step
  - “slow down” fast components by performing \( k \) (\( = \) rate) transitions
  - input adaptors transform \( k \)-tuples to/from single values
Synchronous Model (2)

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

![Diagram of synchronous model with fast components and input adaptors]

- Transformations/“formal patterns” define synchronous model
  - “$k$-step decelerated machine”
  - “input adaptor closure machine”
Asynchronous Model (1)

Add “wrappers” around each machine

- input buffers, output buffers, timers

- **optimal** PALS period: \( \mu_{\text{max}} + 2 \cdot \epsilon + \max(2 \cdot \epsilon - \mu_{\text{min}}, \alpha_{\text{max}}) \)
  - clock skew \( \epsilon \), execution time \( \alpha_{\text{max}} \), and network delay \( \mu_{\text{min}}, \mu_{\text{max}} \)
Components perform at different rates

Assumption: adaptors ignore inputs not received in time
Correctness of Multirate PALS

- **Stable states** of asynchronous models
  - virtually synchronized states
  - PALS wrapper: all input buffers full, all output buffers empty
- Correct by construction

\[
\text{synchronous design } \models \varphi \iff \text{ (“stable-state”) asynchronous design } \models \varphi
\]
Integrated modular avionics example

- Which of two cabinets is active?
- Non-active side monitors active sides, failures, and pilot toggle

By Steven Miller and Darren Cofer at Rockwell-Collins
Case Study: Active Standby (2)

System Requirements

$R_1$: 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: Active Standby (3)

- Comparison with the simplest possible asynchronous model

<table>
<thead>
<tr>
<th>Model</th>
<th>#States</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synch.</td>
<td>185</td>
<td>0.1 s</td>
</tr>
<tr>
<td>Asynch. (0)</td>
<td>3047832</td>
<td>1249 s</td>
</tr>
<tr>
<td>Asynch. (1)</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

- $10!$ different message reception ordering in each round
Outline

1. Multirate PALS
2. Multirate Synchronous AADL
3. Case Study: Turning an Airplane
Goal

Make Multirate PALS methodology and formal verification available to domain-specific modeling
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, ...
- Avionics, aerospace, medical devices, robotic, ...
- OSATE: Eclipse plug-ins for AADL
Goal:

*Make Multirate PALS methodology and formal verification available to domain-specific modeling*

1. Model synchronous design $\mathcal{E}$ in Multirate Synchronous AADL
2. Verify $\mathcal{E}$ using MR-SynchAADL OSATE plugin
Multirate Synchronous AADL (2)

- **Subset of AADL** to model synchronous PALS designs
  - identifies AADL models that can be considered as *synchronous*
  - extended with **new annotations**: property set `MR_SynchAADL`
  - provides **predefined** input adaptors

- **Focus on behavioral and structuring subset of AADL**
  - abstract from hardware and memory, etc.,
AADL constructs in subset have the **same meaning** as before

- easy to use for AADL modeler
- **same behaviors** as in AADL, **without** the intermediate states introduced by asynchrony

- **Formalized** in real-time rewriting logic
  - **Real-Time Maude**: formal analysis tool for real-time systems
OSATE/Eclipse plug-in for Synchronous AADL

Real-Time Maude model checking within OSATE
- checks if given model is valid Multirate Synchronous AADL model
- automatic synthesis of Real-Time Maude model

Requirement specification language
- easy to define system requirements as temporal formulas
Modeling tools | Rewriting logic | Model checking (Real-Time Maude)
---|---|---
System design $M$ | $R_M$ | LTL
time-bounded LTL
Timed CTL
Metric LTL
Property specification $spec$ | Logic formula $\varphi_{spec}$ | ...
The MR-SynchAADL Tool (3)

- MR-SynchAADL window in OSATE

```
name: Airplane_scenario_Instanc;
   -- an AADL implementation
model: Airplane::Airplane.scenario;
   -- a path for the corresponding instance model
instance: "AirplaneTurn/instances/Airplane_Airplane_scenario_Instanc.aaxl2";
   -- requirements
requirement safety: [] safeYaw;
requirement safeTurn: safeYaw U (stable \ reachGoal) in time <= 7200;
   --- other formulas and propositions
formula safeYaw: turningCtrl.mainController.ctrlProc.ctrl1Thread \ abs(currYaw) < 1.0;
formula stable: turningCtrl.mainController.ctrlProc.ctrl1Thread \ abs(currRoll) < 0.5 and abs(currYaw) < 0.5;
formula reachGoal: turningCtrl \ abs(curr_direction - 60.0) < 0.5;
```

Ready.
Untimed model check {initial} 1-w safety in AIRPLANE_SCENARIO_INSTANCE-VERIFICATION-DEF with mode deterministic time increase
Result Bool: true

rewrites: 486318 in 502ms cpu (507ms real) (967240 rewrites/second)
Model check{initial} 1-t safeTurn in AIRPLANE_SCENARIO_INSTANCE-VERIFICATION-DEF in time <= 7200 with mode deterministic time increase
Result Bool: true
Outline

1. Multirate PALS

2. Multirate Synchronous AADL

3. Case Study: Turning an Airplane
Problem: Design a Controller for Turning an Airplane

- aileron controllers (e.g. 67 Hz) and rudder controllers (e.g. 50 Hz)
- controller must ensure synchronization for turning aircraft
Move ailerons to roll airplane for a turn

Turning rate \( d\psi = \left(\frac{g}{v}\right) \times \tan \phi \) (roll angle \( \phi \))
Rolling causes adverse yaw

- sideslip in wrong direction
- use rudder to avoid this
- yaw angle $\beta$ should always be 0

Fig. 3.21 Lateral forces.
Side force derivative due to sideslip velocity $Y_v$. The change in sideslip velocity, $v$, during disturbance changes the incidence angle, $\beta$, of the aircraft's velocity vector, $V_T$, (or relative wind) to the vertical surfaces of the aircraft comprising the fin and fuselage sides (see Figure 3.21). The change in incidence angle $\frac{v}{V_T}$ results in a sideways lifting force being generated by these surfaces. The net side force from the fuselage and fin combined is equal to $Y_v v$ where $Y_v$ is the sideforce derivative due to the sideslip velocity.

Yawing moment derivative due to sideslip velocity $N_v$. The side force on the fin due to the incidence, $\beta$, results from the sideslip velocity, $v$, creating a yawing moment about the CG which tends to align the aircraft with the relative wind in a similar manner to a weathercock (refer to Figure 3.21).

The main function of the fin is to provide this directional stability (often referred to as weathercock stability). This yawing moment is proportional to the sideslip velocity and is dependent on the dynamic pressure, fin area, fin lift coefficient and the fin moment arm, the latter being the distance between the aerodynamic centre of the fin and the yaw axis through the CG. However, the aerodynamic lateral forces acting on the fuselage during side-slipping also produce a yawing moment which opposes the yawing moment due to the fin and so is destabilising. The net yawing moment due to sideslip is thus dependent on the combined contribution of the fin and fuselage. The fin area and moment arm, known as the fin volume, is thus sized to provide...
Roll angle ($\phi$) and yaw angle ($\beta$):

\[
d\phi^2 = \frac{(\text{Lift Right} - \text{Lift Left})}{(\text{Weight} \times \text{Length of Wing})} \tag{1}
\]

\[
d\beta^2 = \text{Drag Ratio} \times \frac{(\text{Lift Right} - \text{Lift Left})}{(\text{Weight} \times \text{Length Wing})} + \frac{\text{Lift Vertical}}{(\text{Weight} \times \text{Length of Aircraft})} \tag{2}
\]

where

\[
\text{Lift} = \text{Lift constant} \times \text{Angle} \tag{3}
\]
Architecture of the Airplane Turning Control System

The Airplane Turning Control System (60ms, rate = 10)
Main Controller + Sensors (60ms, rate = 1)
Left-wing Sub-controller (15ms, rate = 4)
Rudder Sub-controller (20ms, rate = 3)
Right-wing Sub-controller (15ms, rate = 4)

Pilot Console (600ms, rate = 1)
Multirate Synchronous AADL Model (1)

```plaintext
system TurningController -- "interface" of the turning controller
  features
    pilot_goal: in data port Base_Types::Float {MR_SynchAADL::InputAdaptor => "use in first iteration";};
    curr_dr: out data port Base_Types::Float;
end TurningController;

system implementation TurningController.impl
  subcomponents
    mainCtrl: system Maincontroller.impl;
    rudderCtrl: system Subcontroller.impl;
    leftCtrl: system Subcontroller.impl;
    rightCtrl: system Subcontroller.impl;
  connections
    port leftCtrl.curr_angle -> mainCtrl.left_angle {Timing => Delayed;};
    port rightCtrl.curr_angle -> mainCtrl.right_angle {Timing => Delayed;};
    port rudderCtrl.curr_angle -> mainCtrl.rudder_angle {Timing => Delayed;};
    port mainCtrl.left_goal -> leftCtrl.goal_angle {Timing => Delayed;};
    port mainCtrl.right_goal -> rightCtrl.goal_angle {Timing => Delayed;};
    port mainCtrl.rudder_goal -> rudderCtrl.goal_angle {Timing => Delayed;};
    port pilot_goal -> mainCtrl.goal_angle;
    port mainCtrl.curr_dr -> curr_dr;
  properties
    Period => 60 ms;
    Period => 15 ms applies to leftCtrl, rightCtrl;
    Period => 20 ms applies to rudderCtrl;
    Data_Model::Initial_Value => ("1.0") applies to leftCtrl.ctrlProc.ctrlThread.diffAngle, rightCtrl.ctrlProc.ctrlThread.diffAngle;
    Data_Model::Initial_Value => ("0.5") applies to rudderCtrl.ctrlProc.ctrlThread.diffAngle;
... 
end TurningController.impl;
```
system Subcontroller -- "interface" of a device controller
  features
    goal_angle: in data port Base_Types::Float
      {MR_SynchAADL::InputAdaptor => "use in first iteration"};
    curr_angle: out data port Base_Types::Float;
end Subcontroller;

thread implementation SubcontrollerThread.impl
  subcomponents
    currAngle : data Base_Types::Float
      {Data_Model::Initial_Value => ("0.0");};
    goalAngle : data Base_Types::Float
      {Data_Model::Initial_Value => ("0.0");};
    diffAngle : data Base_Types::Float;
  annex behavior_specification {**
    states
      init: initial complete state; move, update: state;
    transitions
      init -[on dispatch]-> move;

      move -[abs(goalAngle - currAngle) > diffAngle]-> update {
        if (goalAngle - currAngle >= 0) currAngle := currAngle + diffAngle
        else currAngle := currAngle - diffAngle end if
      };

      move -[otherwise]-> update {currAngle := goal_angle};

      update -[ ]-> init {
        if (goal_angle'fresh) goalAngle := goal_angle end if; curr_angle := currAngle};
    **};
end SubcontrollerThread.impl;
Key properties:

- reach **desired direction** (+ no roll or yaw) in reasonable time
- **yaw angle** always close to 0 during turn

In the requirement specification language:

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

formula safeYaw:
    turnCtrl.mainCtrl.ctrlProc.ctrlThread | abs(currYaw) < 1.0;
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;
```
### Model checking with different pilot behaviors

<table>
<thead>
<tr>
<th>Model</th>
<th>Env.</th>
<th>$T \leq 600\text{ ms}$ states</th>
<th>$T \leq 1,800\text{ ms}$ states</th>
<th>$T \leq 3,000\text{ ms}$ states</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>time</td>
<td>time(s)</td>
<td>time(s)</td>
</tr>
<tr>
<td>Sync.</td>
<td>Det.</td>
<td>2 0.14</td>
<td>4 0.16</td>
<td>6 1.18</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4 0.16</td>
<td>28 0.33</td>
<td>202 1.55</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6 0.16</td>
<td>116 0.89</td>
<td>2,091 14.86</td>
</tr>
<tr>
<td>Async.</td>
<td>Det.</td>
<td>6,327 0.76</td>
<td>28,071 2.98</td>
<td>50,139 50.14</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>17,469 2.26</td>
<td>381,213 73.13</td>
<td>2,547,423 2,884.81</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>28,611 3.01</td>
<td>1,634,211 938.79</td>
<td>- &gt; 10 hours</td>
</tr>
</tbody>
</table>
**Concluding Remarks**

- **Multirate PALS**
  - reduces design and verification of DRTS to its synchronous version

- **Multirate Synchronous AADL**
  - modeling synchronous designs in AADL

- **MR-SynchAADL**
  - simulation and model checking in OSATE
K. Bae, P. C. Ölveczky, and J. Meseguer.
Definition, Semantics, and Analysis of Multirate Synchronous AADL.
Submitted to a conference, http://formal.cs.illinois.edu/kbae/MR-SynchAADL

K. Bae, J. Krisiloff, J. Meseguer, and P. C. Ölveczky.
Designing and Verifying Distributed Cyber-Physical Systems using Multirate PALS: An Airplane Turning Control System Case Study.
Science of Computer Programming (2014), to appear,
http://formal.cs.illinois.edu/kbae/airplane

K. Bae, J. Meseguer, and P. C. Ölveczky.
Formal patterns for multirate distributed real-time systems.
Science of Computer Programming (2013), in press,
http://dx.doi.org/10.1016/j.scico.2013.09.010

K. Bae, P. C. Ölveczky, J. Meseguer, and A. Al-Nayeem.
The SynchAADL2Maude tool.

K. Bae, P. C. Ölveczky, A. Al-Nayeem, and J. Meseguer.
Synchronous AADL and its formal analysis in Real-Time Maude.
K. Bae, J. Krisiloff, J. Meseguer, and P. C. Ölveczky.
PALS-based analysis of an airplane multirate control system in Real-Time Maude.

K. Bae, J. Meseguer, and P. C. Ölveczky.
Formal patterns for multirate distributed real-time systems.

Al-Nayeem, A., Sha, L., Cofer, D.D., Miller, S.M.
Pattern-based composition and analysis of virtually synchronized real-time distributed systems.

P. C. Ölveczky and J. Meseguer.
Formalization and correctness of the PALS architectural pattern for distributed real-time systems.

Al-Nayeem, A., Sun, M., Qiu, X., Sha, L., Miller, S.P., Cofer, D.D.
A formal architecture pattern for real-time distributed systems.
Thank you
Rewrite theory $\mathcal{R} = (\Sigma, E, R)$: a formal specification of concurrent systems

- $\Sigma$: signature for logical terms $t \in T_\Sigma$
- $E$: equations that define equalities $t =_E t'$
- $R$: rewrite rules specifying labeled transitions $l : [t]_E \rightarrow [t']_E$

1. **naturally describes** many concurrent systems
   - including their states and events
   - can be used as a **universal system specification logic**

2. **executable** under reasonable assumptions

3. **Maude**: high-performance rewriting logic language and tool
Real-Time Maude: formal analysis tool for real-time systems
- expressiveness and ease of specification
- simulation and (LTL and timed CTL) model checking tool
- equational algebraic specification defines static parts
- rewrite rules define transitions
- suitable for object-oriented specification
Object-oriented semantics

“one-to-one” correspondence AADL model ↔ Real-Time Maude term

Example (the active standby)

< MAIN : System |
  features : none,
  properties : Synchronous(true) ; SynchPeriod(2),
  subcomponents : < env : System | ... >
    < sideOne : System | ... >
    < sideTwo : System | ... >,
  connections : sideOne . side1ActiveSide -->> sideTwo . side1ActiveSide ;
    sideTwo . side2ActiveSide -->> sideOne . side2ActiveSide ;
    ...
    env . side2Failed --> sideTwo . side2Failed >

Synchronous step formalized by rewrite rules
Rewrite rule defining synchronous dynamics for each step:

\[\text{crl} \ [\text{syncStepWithTime}] : \]
\[\{\text{SYSTEM}\} \]
\[\Rightarrow \]
\[\{\text{applyTransitions(} \]
\[\text{transferData(} \]
\[\text{applyEnvTransitions(VAL, SYSTEM)})\} \]
\[\text{in time period(SYSTEM)} \]
\[\text{if VAL ; VALS := allEnvAssignments(SYSTEM)} . \]
Equation defining deterministic thread behaviors:

\[
\text{ceq } \text{applyTransitions}(\< 0 : \text{Thread} \mid \text{properties} : \text{Deterministic(true)} ; \text{PROPS}, \text{features} : \text{PORTS}, \text{currState} : \text{L1}, \text{completeStates} : \text{LS}, \text{variables} : \text{VAL}, \text{transitions} : (\text{L1} -\text{[GUARD]}\rightarrow \text{L2} \{\text{SL}\}) ; \text{TRANS} >) = \\
\begin{cases} 
\text{if } \text{L2 in LS then} \\
\< 0 : \text{Thread} \mid \text{features} : \text{NEW-PORTS}, \text{currState} : \text{L2}, \\
\text{variables} : \text{NEW-VALUATION} > \\
\text{else} \\
\text{applyTransitions}(\< 0 : \text{Thread} \mid \text{features} : \text{NEW-PORTS}, \text{currState} : \text{L2}, \\
\text{variables} : \text{NEW-VALUATION} >) \\
\end{cases}
\]

\[
\text{if } \text{evalGuard(GUARD, PORTS, VAL)} = \text{true} \\
\text{not someTransEnabled(TRANS, L1, VAL, PORTS)} \\
\text{transResult(NEW-PORTS, NEW-VALUATION)} := \\
\text{executeTransition(L1 -[GUARD]-> L2 \{SL\}, PORTS, VAL)} .
\]