Synchronous AADL and its Formal Analysis in Maude

Abdullah Al-Nayeem\(^1\), Kyungmin Bae\(^1\), José Meseguer\(^1\), and Peter Ölveczky\(^2\)

\(^1\)University of Illinois at Urbana-Champaign
\(^2\)University of Oslo
Outline

1. AADL and Maude
2. PALS
3. Synchronous AADL
4. Formal analysis of Synchronous AADL in Maude
   - Real-Time Maude Semantics
   - SynchAADL2Maude Tool
   - Avionics Case Study
Acknowledgment

This is part of a broader PALS/AADL collaboration with:

- Lui Sha, Min Young Nam, and Mu Sun at UIUC
- Steve Miller and Darren Cofer at Rockwell-Collins
Outline

1. AADL and Maude
2. PALS
3. Synchronous AADL
4. Formal analysis of Synchronous AADL in Maude
   - Real-Time Maude Semantics
   - SynchAADL2Maude Tool
   - Avionics Case Study
Real-Time Maude: formal analysis tool for real-time systems

- formalism emphasizes expressiveness and ease of specification
Real-Time Maude: formal analysis tool for real-time systems

- formalism emphasizes expressiveness and ease of specification
- object-oriented specification
  - nested objects
Real-Time Maude: formal analysis tool for real-time systems

- formalism emphasizes expressiveness and ease of specification
- object-oriented specification
  - nested objects
- high-performance simulation and model checking tool
Real-Time Maude: formal analysis tool for real-time systems

- formalism emphasizes expressiveness and ease of specification
- object-oriented specification
  - nested objects
- high-performance simulation and model checking tool
- successfully applied to advanced state-of-the-art systems
  - 50+ page multicast protocol for active networks
  - wireless sensor network algorithms
  - embedded car software used by major car makers
- formal semantics and analysis for:
  - Ptolemy II discrete-event models
  - real-time model transformation languages
Real-Time Maude (II)

- Specification language:
  - algebraic **equational specification** for data types/functions
  - rewrite rules define instantaneous transitions
  - tick rewrite rules define time elapse
Real-Time Maude (II)

- Specification language:
  - algebraic equational specification for data types/functions
  - rewrite rules define instantaneous transitions
  - tick rewrite rules define time elapse

- Analysis capabilities:
  - rewriting for simulation/prototyping
  - explicit-state breadth-first search for reachability analysis
  - LTL and timed CTL model checking
industry standard for embedded/RT systems modeling


avionics, aerospace, medical devices, robotic, . . .
AADL

- **industry standard** for embedded/RT systems modeling
- avionics, aerospace, medical devices, robotic, . . .
- hierarchy of software and hardware components
  - process, thread, subprogram, . . .
  - processor, memory, device, bus, . . .
- **OSATE**: Eclipse plug-ins for AADL
- asynchronous
AADL

- **industry standard** for embedded/RT systems modeling
- avionics, aerospace, medical devices, robotic, . . .
- hierarchy of **software** and **hardware** components
  - process, thread, subprogram, . . .
  - processor, memory, device, bus, . . .
- **OSATE**: Eclipse plug-ins for AADL
- asynchronous
- **no formal semantics!**
AADL2Maude

- Presented earlier: Real-Time Maude semantics, code generation, and formal analysis for a subset of behavioral AADL models
  - only software components
  - model checked event-driven network of medical devices
- In this presentation, we discuss the SynchAADL2Maude semantics and translation to support simulation and formal analysis of synchronous systems in AADL
1. AADL and Maude
2. PALS
3. Synchronous AADL
4. Formal analysis of Synchronous AADL in Maude
   - Real-Time Maude Semantics
   - SynchAADL2Maude Tool
   - Avionics Case Study
Many distributed real-time systems:
- collection of components that communicate asynchronously
- each component has a local clock
- should change state and respond to environment input within hard real-time bounds
- should achieve virtual synchrony

Examples:
- integrated modular avionics
- distributed control in motor vehicles
- interoperation of medical devices

Often safety-critical systems

Design, verification, and implementation hard and error-prone
- asynchronous communication, message delays, and clock skews

Model checking verification often unfeasible due to the state space explosion caused by the system’s concurrency
Formal Architectural Patterns

Formal architectural patterns:

- Generic formal specification of an engineering solution to a 
  generic design problem
  - formal correctness guarantees
  - reduces system complexity
    - verification and correctness of implementation much easier
- Amortize verification effort on a general family of designs
The PALS Formal Architectural Pattern (I)

**PALS**: Physically Asynchronous, Logically Synchronous

- Reduces design and verification of a distributed real-time system (that should ensure virtual synchrony and satisfy hard real-time bounds) to its underlying synchronous version
- Relies on **asynchronous bounded delay (ABD)** network infrastructure
  - max bound on the communication delay
- Assumes underlying **clock synchronization** guarantees maximal bound on the clock skews
PALS can be seen as a formally verified model transformation

$$(\mathcal{E}, \Gamma) \mapsto A(\mathcal{E}, \Gamma)$$

- $\mathcal{E}$ is a synchronous design
- $\Gamma$ are performance bounds on
  - max clock skew $\epsilon$ between any local clock and a “global” clock
  - min and max time for reading input, performing a “transition,” and producing output
  - min and max message transmission delay
- $A(\mathcal{E}, \Gamma)$ is the corresponding asynchronous design of the distributed real-time system
Synchronous Model

A synchronous model is an ensemble of state machines.

In each “iteration,” each machine at the same time:
- reads input from environment and from the values generated in the previous round in the “feedback” loops
- produces output and changes its local state
- Environment nondeterministically generates outputs satisfying a constraint
Outline

1. AADL and Maude
2. PALS
3. Synchronous AADL
4. Formal analysis of Synchronous AADL in Maude
   - Real-Time Maude Semantics
   - SynchAADL2Maude Tool
   - Avionics Case Study
Why Synchronous AADL?

- Exploit the PALS pattern
- Define high-level **synchronous** designs in AADL
  - can also be mapped onto other distributed real-time architectures (TTA, ...)
- Verify synchronous designs with the SynchAADL2Maude tool
- Can use PALS to automatically generate **asynchronous** distributed AADL model from synchronous AADL model
Synchronous AADL

- Subset of AADL extended with new annotations
  - new property set SynchAADL
- Focus on AADL subset to model synchronous PALS designs
  - abstract from hardware and memory, etc.,
  - focus on behavioral and structuring subset
- AADL constructs have the same meaning as before
  - easy to use for AADL modeler
  - same behaviors as in AADL, without the intermediate states introduced by asynchrony
Targeted Behavioral AADL Subset

- Focus on **software** components
  - hierarchical components
  - connections, data ports,
  - thread, process, systems, ...

- Thread behaviors: **Behavior Annex**
  - transition systems with local state variables
Threads

- One nondeterministic environment thread with properties:
  
  SynchAADL::InputConstraints => "(not s1F and s2F and not s2FA)";

  SynchAADL::IsEnvironment => true;

- All other threads are assumed to be deterministic:
  
  SynchAADL::Deterministic => true;
Thread Dispatch

- **Event-driven** (aperiodic) dispatch not suitable for synchronous models
  - immediate transfer of control from one node to another
- Therefore: only **periodic** dispatch in Synchronous AADL
  - all threads have the same period
- All transitions should be instantaneous
- Declare synchrony in top-level system component

```cpp
SynchAA: :Synchronous => true;
SynchAA: :syncPeriod => 2 Ms;
```
Connections

- **Immediate** connections impose execution order on threads with the same dispatch time
  - unsuitable for threads executing synchronously
- **Delayed** connections make data available at the beginning of the next dispatch (for threads with the same dispatch time)
  - perfect for our synchronous models
  - only data ports/connections (therefore, no mode transitions)
- **Immediate** connections from the environment
- All other connections are **delayed** connections
Example

Environment
(period = 3)

T1
(period = 3)

T2
(period = 3)

T3
(period = 3)

T4
(period = 3)
Outline

1. AADL and Maude
2. PALS
3. Synchronous AADL
4. Formal analysis of Synchronous AADL in Maude
   - Real-Time Maude Semantics
   - SynchAADL2Maude Tool
   - Avionics Case Study
Real-Time Maude Semantics

- Object-oriented semantics
- “One-to-one” correspondence AADL model $\leftrightarrow$ Real-Time Maude term
  - hierarchical objects
  - easy to map counterexamples to AADL behaviors
Class for any software component:

```plaintext
class Component | features : Configuration,
                 subcomponents : Configuration,
                 properties : Properties,
                 connections : ConnectionSet .
```
Real-Time Maude Semantics (II)

Class for any software component:

```maude
class Component | features : Configuration, 
                  subcomponents : Configuration, 
                  properties : Properties, 
                  connections : ConnectionSet .
```

Thread components class:

```maude
class Thread | currState : Location, 
              completeStates : LocationSet, 
              transitions : TransitionSet, 
              variables : Valuation .
subclass Thread < Component .
```
Rewrite rule defining dynamics:

\[
\text{crl} \quad \text{[syncStepWithTime]} : \quad \{\text{SYSTEM}\} \\
\Rightarrow \\
\{\text{applyTransitions(} \\
\quad \text{transferData(} \\
\quad \quad \text{applyEnvTransitions(VAL, SYSTEM)))}\} \\
\text{in time period(SYSTEM)} \\
\text{if VAL ; VALS := allEnvAssignments(SYSTEM).}
\]
ceq applyTransitions(
    < 0 : Thread | properties : Deterministic(true) ; PROPS,
    features : PORTS,        currState : L1,
    completeStates : LS,      variables : VAL,
    transitions : (L1 -[GUARD]-> L2 {SL}) ; TRANS >)
= if L2 in LS then
   < 0 : Thread | features : NEW-PORTS, currState : L2,
           variables : NEW-VALUATION >
else
   applyTransitions(< 0 : Thread | features : NEW-PORTS,
                   currState : L2,
                   variables : NEW-VALUATION >) fi
if evalGuard(GUARD, PORTS, VAL)
   \ GUARD =/= otherwise or-else
   not someTransEnabled(TRANSITIONS, L1, VAL, PORTS)
   \ transResult(NEW-PORTS, NEW-VALUATION) :=
   executeTransition(L1 -[GUARD]-> L2 {SL}, PORTS, VAL) .
The SynchAADL2Maude Tool

- **OSATE** plug-in
- Automatic synthesis of Real-Time Maude verification model from Synchronous AADL model
- Checks whether AADL model is a Synchronous AADL model
- Useful generic atomic propositions
  - user can easily define LTL properties
In integrated modular avionics, there are for fault tolerance properties multiple cabinets (with power supply, computer, I/O, etc.) distributed in an aircraft.

**Active Standby System:** decide which cabinet is active
- standby side should monitor the side’s functionalities and the pilot’s manual switch
- standby side notifies active side if change of active sides needed

**Synchronous design** must be realized for the distributed cabinets in the aircraft.

Our models based on AADL model by Abdullah Al-Nayeem of a similar spec developed by Steve Miller and Darren Cofer.
Architecture of active standby for two sides:
system implementation ActiveStandbySystem.impl
subcomponents
    sideOne: system Side1.impl; sideTwo: system Side2.impl;
    env: system Environment.impl;
connections
data port sideOne.side1ActiveSide --> sideTwo.side1ActiveSide;
data port sideTwo.side2ActiveSide --> sideOne.side2ActiveSide;
data port env.side1FullyAvail -> sideOne.side1FullyAvail;
data port env.side1FullyAvail -> sideTwo.side1FullyAvail;
data port env.side2FullyAvail -> sideOne.side2FullyAvail;
data port env.side2FullyAvail -> sideTwo.side2FullyAvail;
data port env.manualSelection -> sideOne.manualSelection;
data port env.manualSelection -> sideTwo.manualSelection;
data port env.side1Failed -> sideOne.side1Failed;
data port env.side2Failed -> sideTwo.side2Failed;
properties
    SynchAADL::Synchronous => true; SynchAADL::synchPeriod => 3;
end ActiveStandbySystem.impl;
Requirement of Active Standby

The active standby system should satisfy the following 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.
SynchAADL2Maude Verification of Active Standby

- Have verified improved versions of the five desired properties
  - each model checking took about 18 seconds
  - asynchronous design of active standby cannot be feasibly model checked
- Example (Requirement 4):

  If a side is failed the other side should become active

  \[ eq \quad R4 = [\] \((\text{side 1 failed} \land \lnot \text{side 2 failed}) \rightarrow 0 \ (\lnot \text{side 2 failed} \rightarrow \text{side 2 active})) \land \((\text{side 2 failed} \land \lnot \text{side 1 failed}) \rightarrow 0 \ (\lnot \text{side 1 failed} \rightarrow \text{side 1 active})) \]. \]

  Maude> (mc \{initial\} |=u R4 .)

  rewrites: 1368833 in 18729ms cpu (18882ms real)
result Bool : true
Future Work

- Extension of Synchronous AADL
- More case studies
- Support for different periods