Executable AADL Models for Early System Qualification Test

WORKSHOP ADEPT2024, BARCELONA

S. RUBINI, S. LEVIEUX, E. CARIOU, F. SINGHOFF, H.N. TRAN, LAB-STICC, UNIV. OF BREST, BREST, FRANCE

G. LE PLUART, THALES DMS, BREST, FRANCE

06/14/2024

Objectives

Early “qualification test” on the architectural model of complex real-time applications

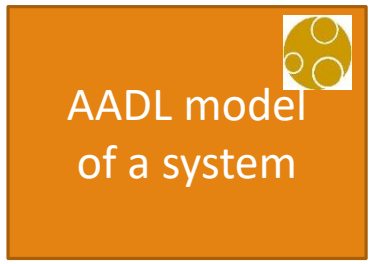
- Test performances, to validate the system temporal behavior
- Generation of test inputs according to the expected operational activity (by algorithms, traces, statistic laws, …)

- For analysis needs of a case study provided by THALES DMS.
  - System engineering of a large distributed system
  - With (soft) real-time constraints
  - Mixing function periodically released and others triggered by data arrivals.

Approach “CosiCosi”

- Transforms an AADL model into an executable model for simulation

Analysis objectives



Osate,
AADLInspector,
Cheddar, …

Static analysis (AADL model level)
- Schedulability feasibility tests (e.g. RTA analysis)
- Restricted scoop of applications

➔ Proof, certification

AADLInspector,
Marhzin, …

Dynamic analysis (AADL model level)
- Simulation with elements of the application behavior (AADL Behavioral Annex)
- Larger scope of applications

➔ Unit tests, debugging on temporal criteria, simple usage scenarios

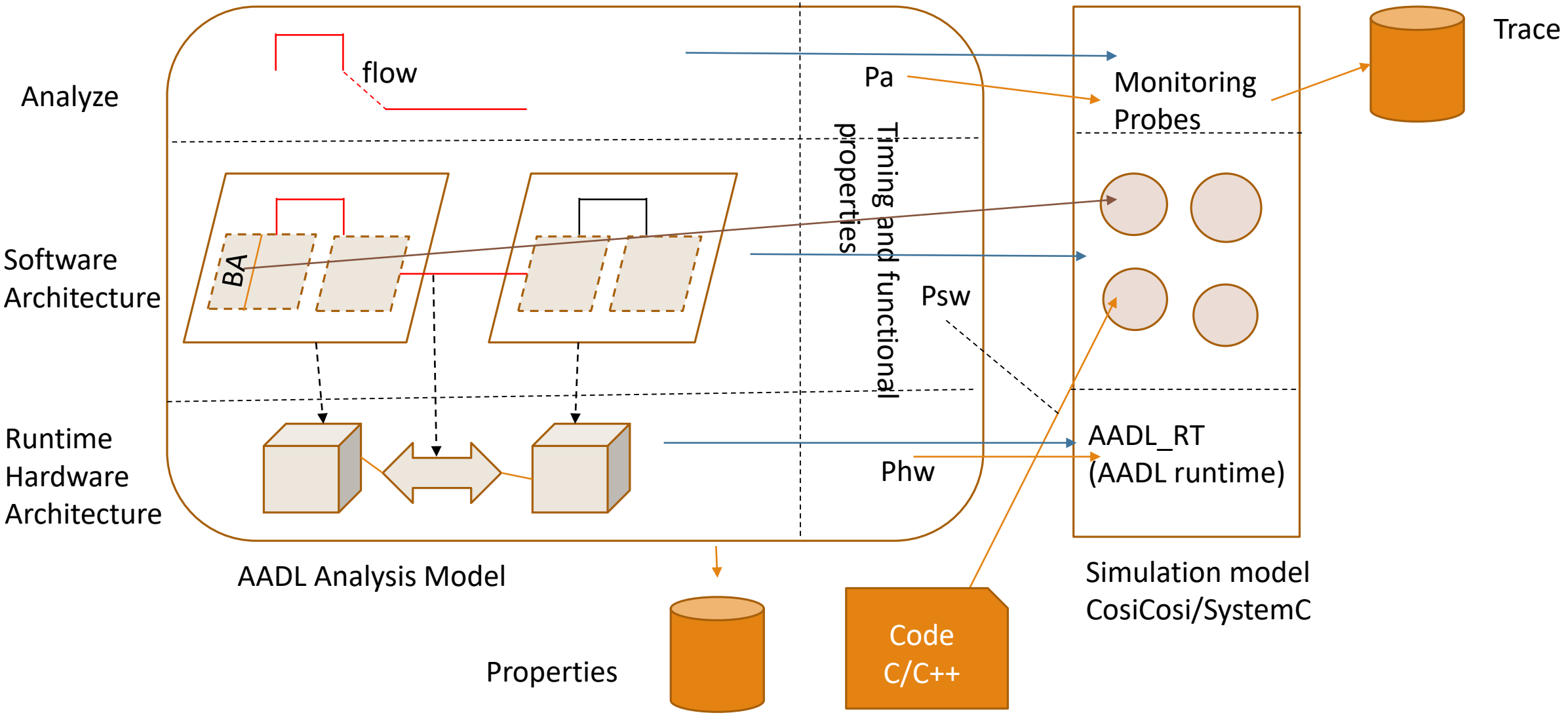
CosiCosi tool

Dynamic analysis (not only on the AADL model level)
- Simulation with elements of the application components
- Take into account of the system load
- Probes (usage rate, slack time, min/max latency)

➔ Intensive tests with not fully deterministic models

Outline

1. Introduction
2. CosiCosi approach
   - Principles
   - Implementation
3. Experiments
   - “Demo system” example
   - Simulator Scalability
4. Perspectives and Conclusion



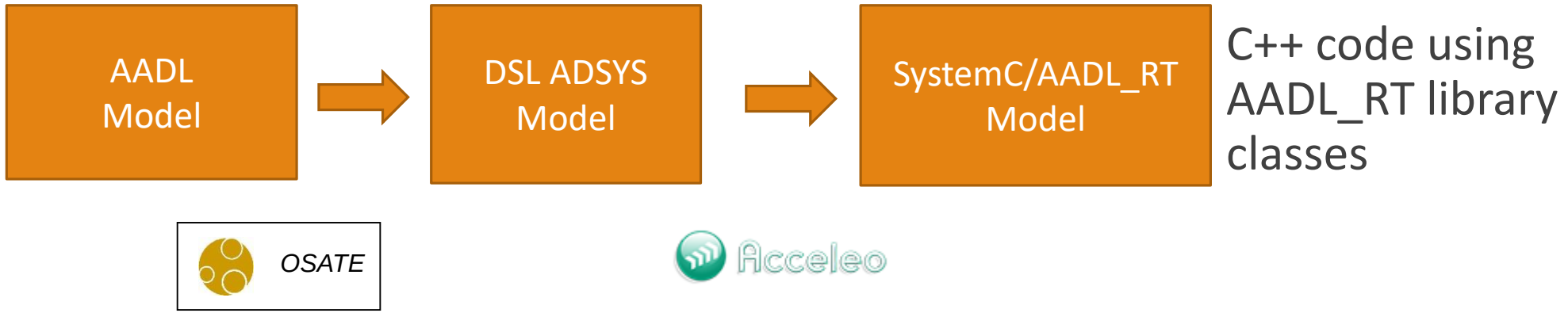
CosiCosi Approach: "Executable Models" based on SystemC simulator, and C++ usage scenarios, and/or C/C++ application components

CosiCosi, Simulation of AADL Model

- SystemC is a C++ library of classes for event-driven simulation of concurrent processes.
- CosiCosi: a SystemC/C++ code that simulates the software/hardware architecture defined in a source AADL model.
  - Generated by model transformation,
  - Supported AADL components/features (currently): thread, data, connection/port, processor
- Not the first AADL/SystemC implementation
  - For different purposes
  - Built from SysRT, a multiprocessor RTOS code written in SystemC
  - AADL_RT, a runtime conforming to the semantic defined in the AADL standard (scheduling, communication protocol …) extending SysRT

[Xiao 2017] Xiao, J., Pimentel, A., & Lipari, G. SysRT: A modular multiprocessor RTOS simulator for early design space exploration. In *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)* (pp. 38-45). IEEE.

CosiCosi executable Model Generation



DSL ADSYS = represents a selection of the AADL concepts allowing for executable models building

Outline

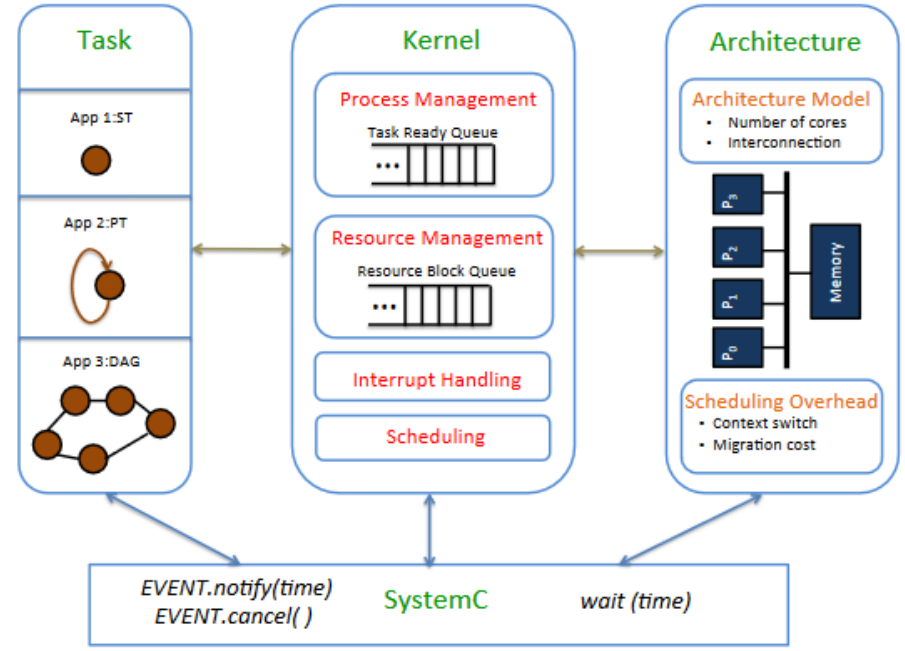
1. Introduction
2. The simulator CosiCosi
   - Principles
   - Implementation
3. Experiments
   - “Demo system” example
   - Simulator Scalability
4. Perspectives and Conclusion

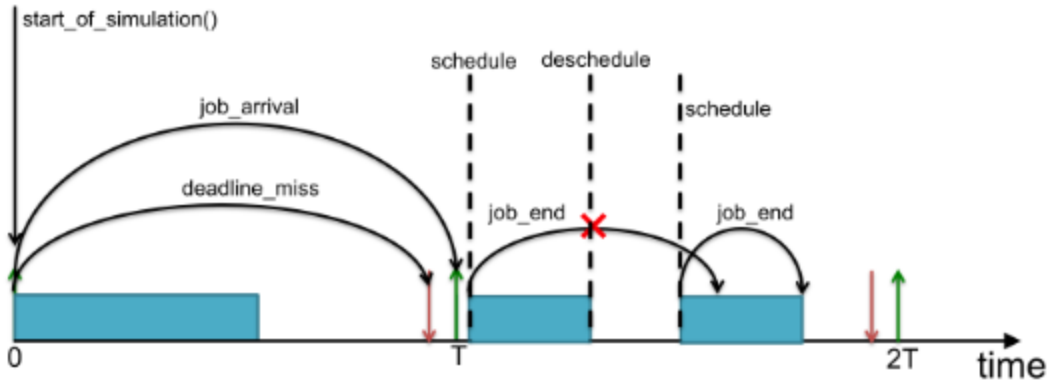
AADL_RT/SysRT Library

AADL_RT is built from the the SysRT library

SysRT is

- A high-level RTOS simulator written in SystemC
- Dedicated to check the respect of the timing requirements during the early design steps

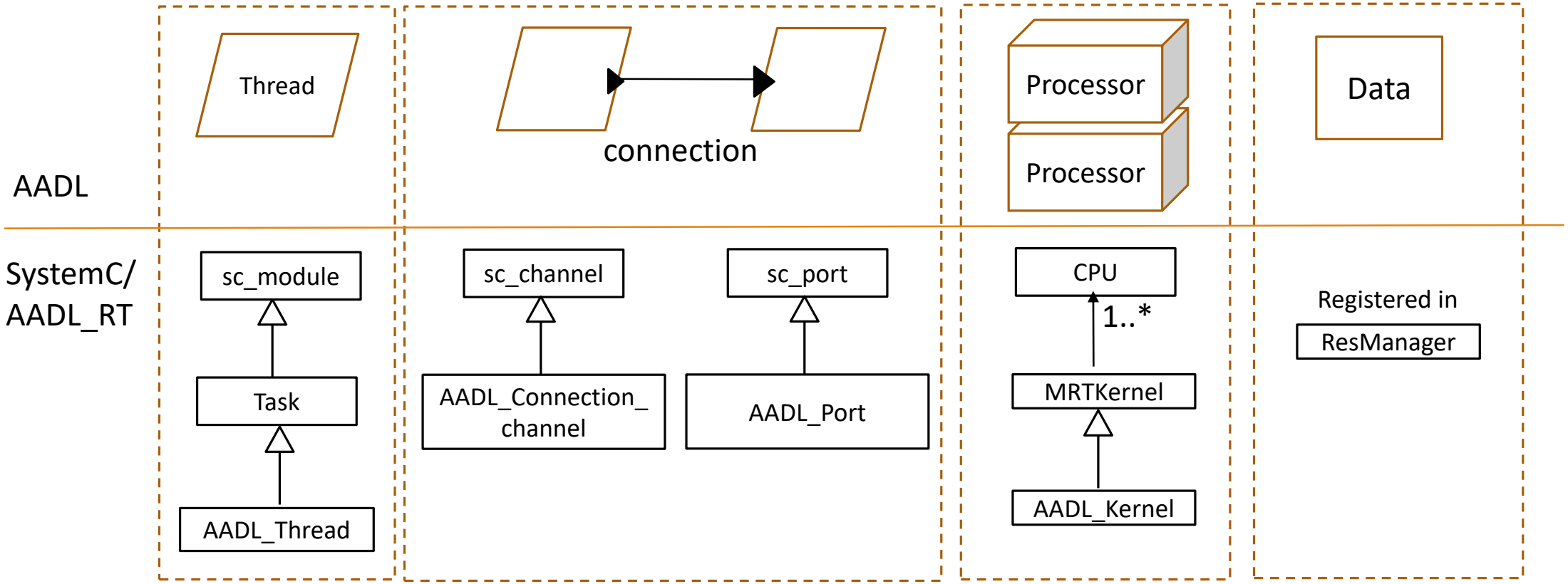




General principle of SysRT: the simulator programs SystemC event deliveries at the times when scheduling operations must be done.

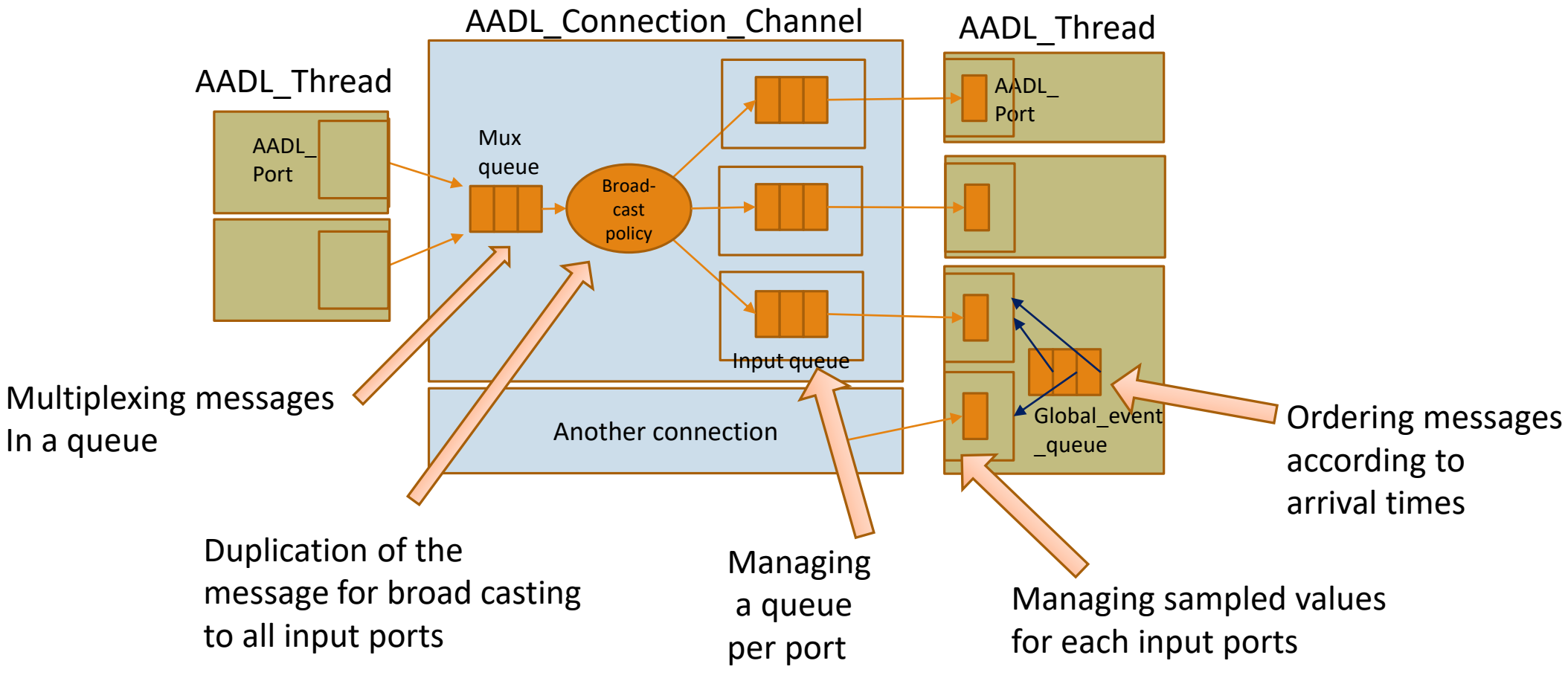
Figures are duplicated from [Xiao 2017]

Transformation Rules

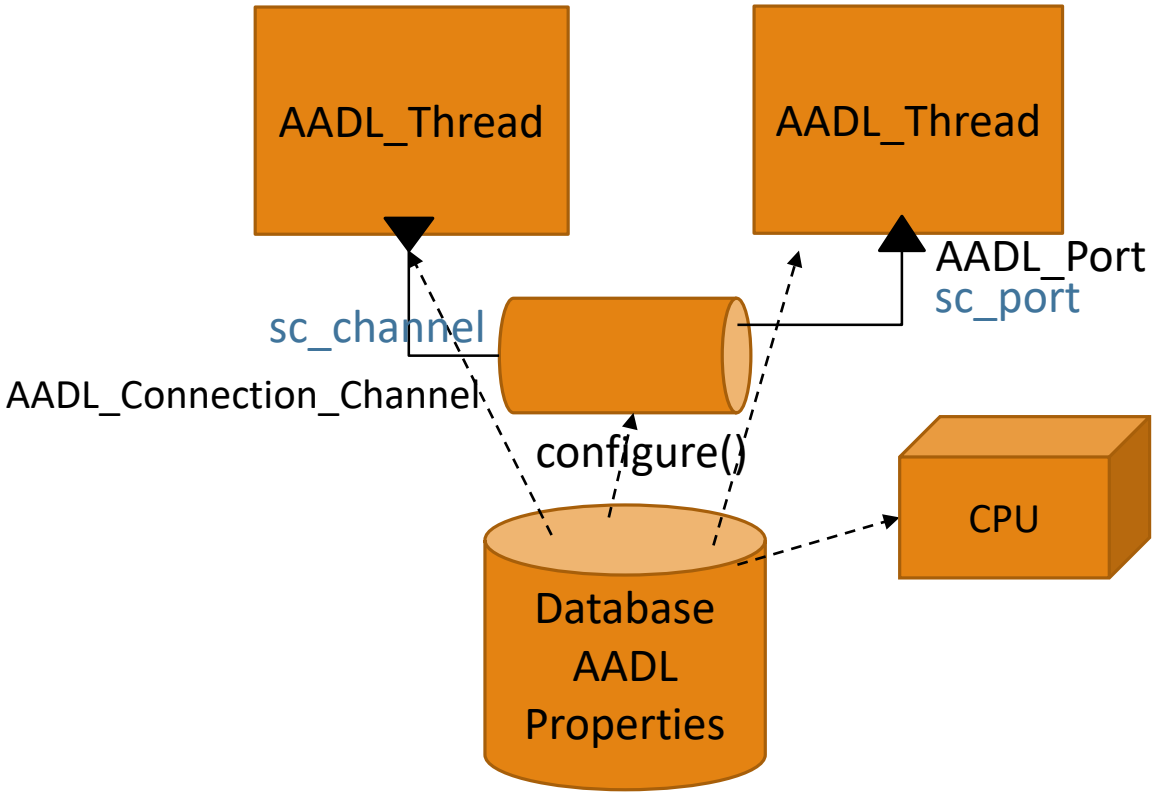


Implementation of AADL connections

AADL N-to-N connection model



Component Configuration

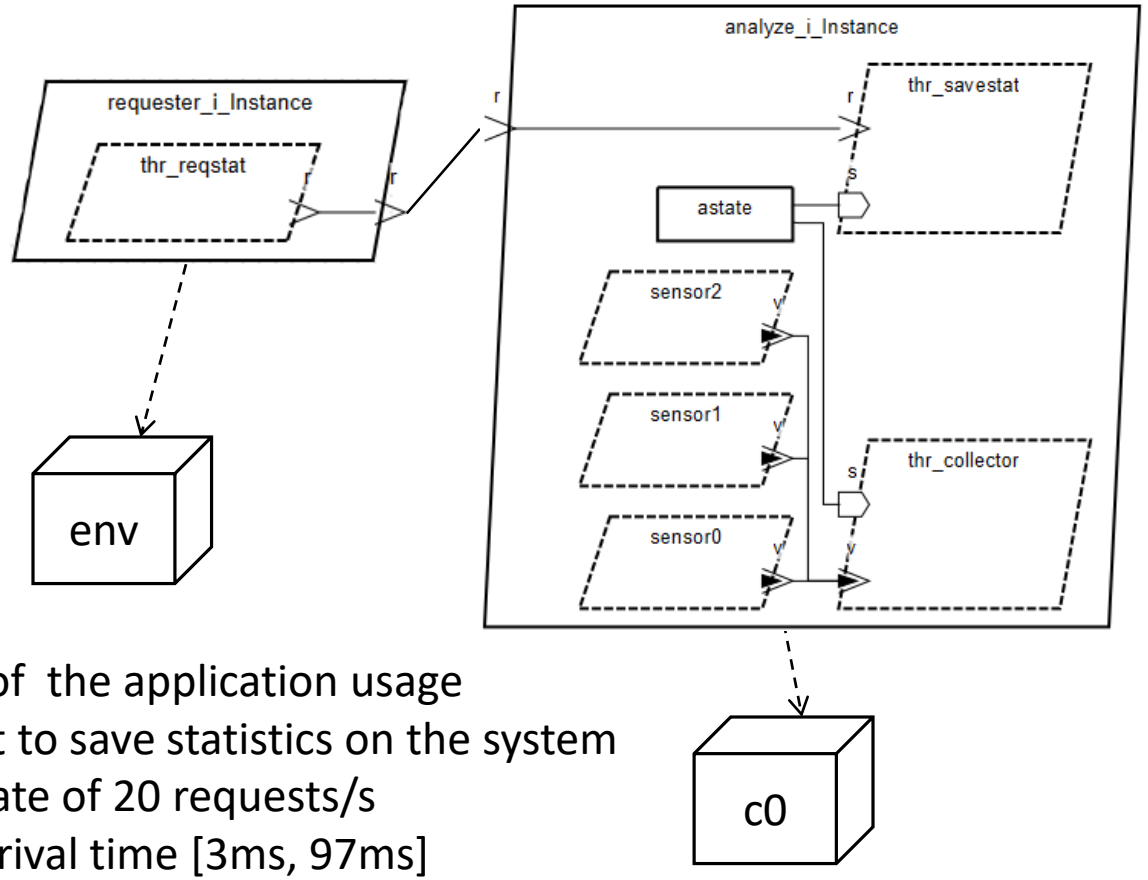


- AADL property values of the model are stored in a database
- Before the beginning of the simulation, the method *configure()* of each component extracts its related properties
- Do not change code generators when adding/changing/removing properties
- May help to setup design space exploration processes

Outline

1. Introduction
2. The simulator CosiCosi
   - Principles
   - Implementation
3. Experiments
   - “Demo system” example
   - Simulator Scalability
4. Perspectives and Conclusion

Example



Name	Period	Capacity	Dispatch	CPU
sensor0	20ms	2ms	Periodic	c0
sensor1	30ms	2ms	Periodic	c0
sensor2	40ms	2ms	Periodic	c0
thr_collector	8ms	5ms	Sporadic	c0
thr_savestat	-	5ms	Aperiodic	c0
thr_reqstat	50ms	$[2ms, 49ms]$	Periodic	env

Features:

- Event and event data ports
- Data access
- N-to-1 connection
- Multi-processor (partitioned)
- Accesses in mutual exclusion to the data “aState”

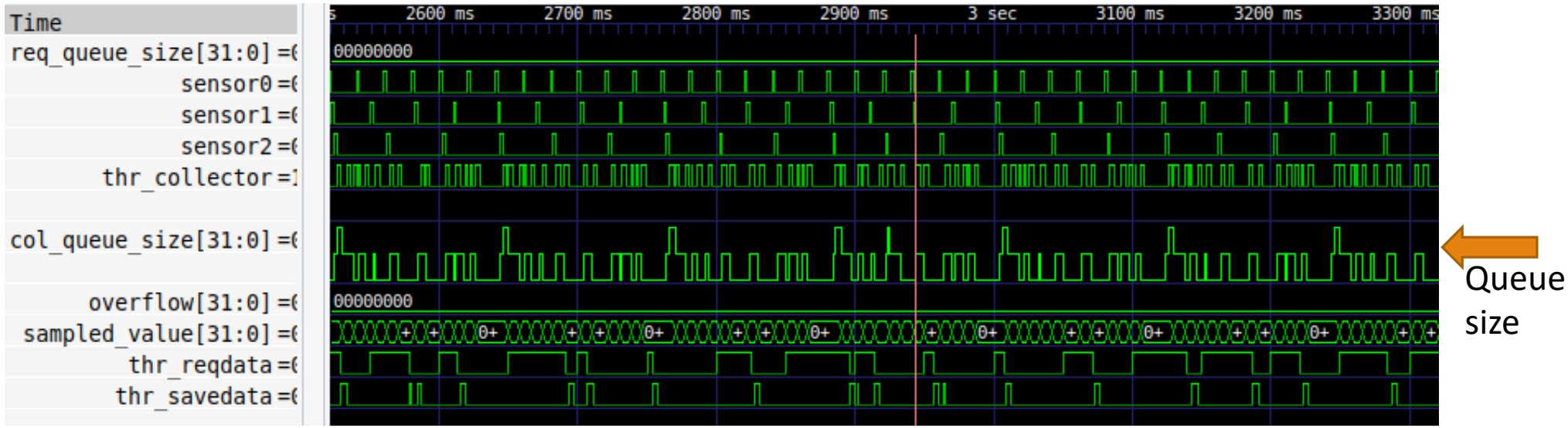
Modeling of the application usage

- Request to save statistics on the system
- Mean rate of 20 requests/s
- Inter-arrival time [3ms, 97ms]

=> Difficult or even impossible with static analysis

Example trace

- No deadline violation detected during simulation
- Point to watch out for: collector entry queue filling (defined to 3 in the AADL model) → <= 2 in the simulation, no overflow



Scalability for long simulations

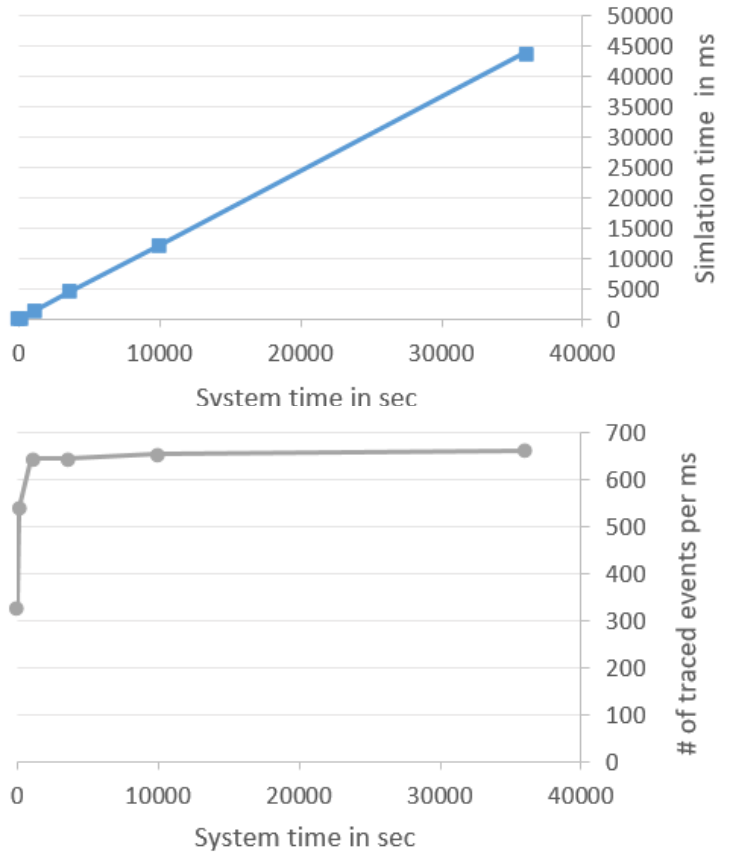
Ability to simulate a system during a long working period

- System time : time at the level of the simulated system
- Simulation time: execution time of the simulator itself

With the previous example (6 threads, 2 connections, 2 CPUs), 10 traced features/events

Comments

- The simulation time increases linearly with the system time (around 1 ms to simulate 1 s of the system time)
- The trace file (VCD format) contents around 3 millions of events for a simulation of 10 hours (of system time)



Execution platform : Intel Core i9-12900, 2.4GHz, 64GB, Linux 5.15, SystemC 2.3.4

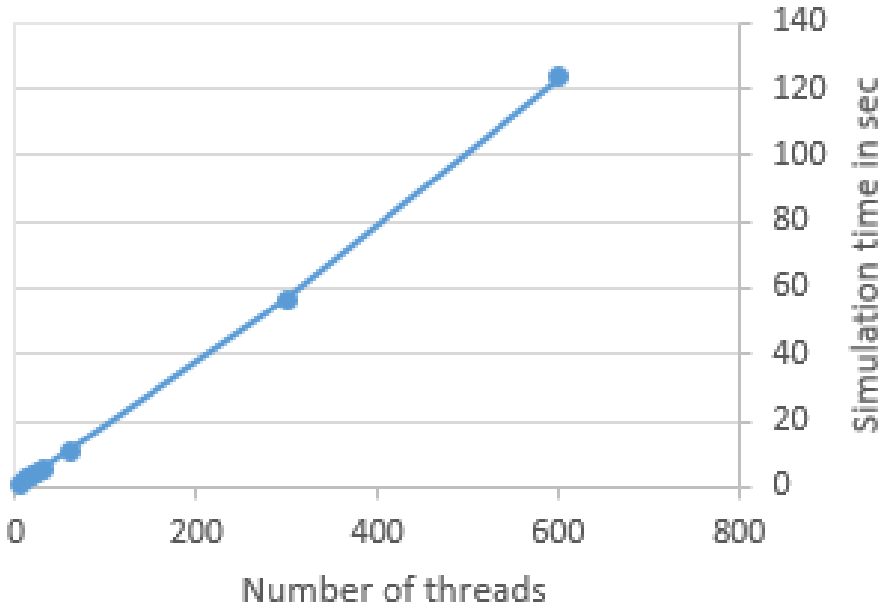
Scalability for large systems

Ability to simulate a large system

- Simulated system: multiple instances of the previous example (6 threads, 2 connections, 2 CPUs), 10 traced features/events
- Simulation of 1 hour of system time

Comments

- The simulation time increases linearly with the number of thread (around 200 ms per thread, with a slight upward inflection for the last measure)
- The trace file (VCD format) contents around 80 millions of events for a simulation of 600 threads on 200 CPUs



Execution platform : Intel Core i9-12900, 2.4GHz, 64GB, Linux 5.15, SystemC 2.3.4

Outline

1. Introduction
2. The simulator CosiCosi
   - Principles
   - Implementation
3. Experiments
   - “Demo system” example
   - Simulator Scalability
4. Perspectives and Conclusion

Conclusion and Perspective

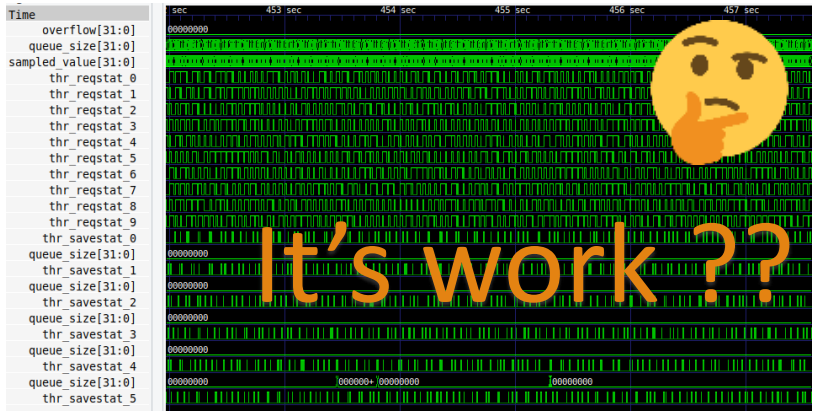
For a THALES DMS case study, for intensive model testing

The executable models allow the testers to run intensive simulation.

- But we need for code instrumentation/probes to monitor specific properties

  → Enrich the AADL model:

  - To represent properties to observe



Some AADL semantic rules were difficult to implement. For example:

- Periodic tasks on immediate connections
- Instantaneous communications on delayed connections

Project status

- Only partial AADL standard covering
- We plan to complete AADL -> CosiCosi transformation
- We plan to express the probes inside the source AADL model