Modeling and Verification of Network-on-Chip using Constrained-DEVS

Soroosh Gholami

Hessam S. Sarjoughian

School of Computing, Informatics, and Decision Systems Engineering

Arizona Center for Integrative Modeling and Simulation

Arizona State University

Spring Simulation Multi-Conference

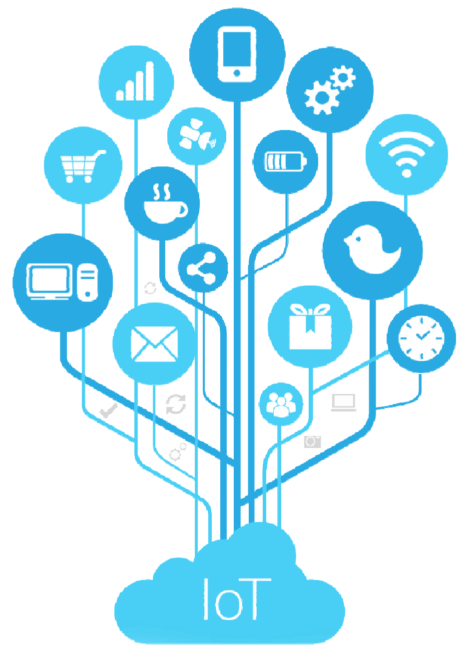
April 23-26, 2017

Electronic Complex Systems

- Network/interaction is inherent to electronic complex systems
- Complexity arises from:
  - Complexity of individual components
    - Functionality of individual components
    - Software, hardware, or physical
  - Interactions between these components
    - Time-sensitive information
    - Overall functionality
- Development steps:
  - Identifying requirements
  - Multiple phases of modeling using variety of methods
  - Multiple phases of model validation and verification
  - Conversion of models to HW/SW pieces
  - Develop communication modules
  - System/subsystem validation and verification
  - Deployment

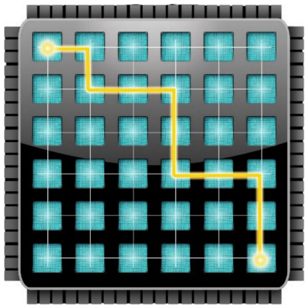
Design

Construction



Complexity and Network-on-Chips

- NoC is a communication system, connecting components of a chip
- NoC design requires
  - design of individual components within the network
  - design of the communication subsystem and protocols
- SoC as a set of software and hardware components interacting through NoC
  - Switches, Processing Elements, and Network Interfaces communicate through links
- Integrated Chip design process has three major phases
  - Electronic System Level (ESL) Design
  - Register Transfer Level (RTL) Design
  - Physical Design



V&V for NoC Models

- Models evaluation based on requirements
  - Verification: building the model correctly
  - Validation: building the correct model
  - Model complexity should not be sacrificed for the sake of V&V
  - Unified framework support is desirable

Overview

- Problem Description/Goals
- Background
- Proposed Research
  - Approach
- Conclusion and Future Work

Limitations of V&V for NoC Design

- Verification is not trivial for DEVS
  - DEVS language is undecidable
  - It is continuous time
  - Simulation is the major means for model evaluation

- Model Evaluation is limited
  - Models are repeatedly abstracted for evaluation

- Complex property (compound) expression
  - Aspects required to check for them are not even modeled (exclusion of information flow)
  - No method to check for them, no language to express them

Scope & Goals

- We limit the scope of the problem to:
  - Modeling framework: Discrete Event System Specification (DEVS)
  - Target system: Network-on-Chip + Processing Element (PE)
  - Validation method: Discrete Event Simulation
  - Verification method: Model Checking
  - Tool: DEVS-Suite[1,2]

- Goals:
  - Extending DEVS modeling with model checking capabilities
  - Extending DEVS-Suite with both modeling checking and simulation of constrained-DEVS

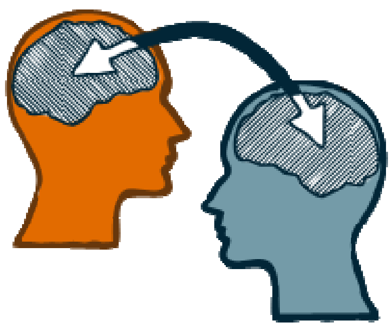
[1] ACIMS, DEVS-Suite Simulator, https://sourceforge.net/projects/devs-suitesim/
[2] Kim, Sungung, Hessam S. Sarjoughian, and Vignesh Elamvazhuthi. "DEVS-suite: a simulator supporting visual experimentation design and behavior monitoring." In Proceedings of the 2009 Spring Simulation Multiconference, Society for Computer Simulation International, 2009.

Elements of Research

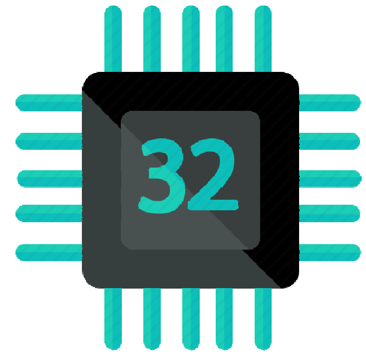
Constrained-DEVS Modeling & Simulation
Constrained-DEVS Model Checking (State exploration)
Timed event-handling

Support for DEVS Simulation
Support for DEVS model checking
Experimental frame-based evaluation

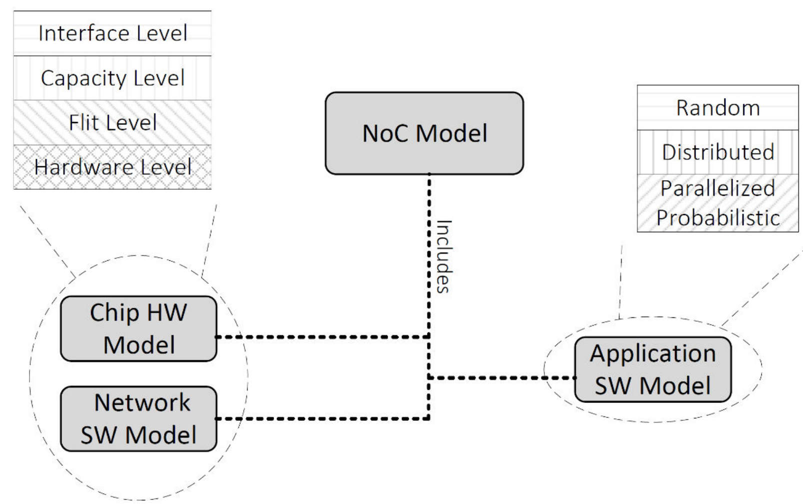
Background



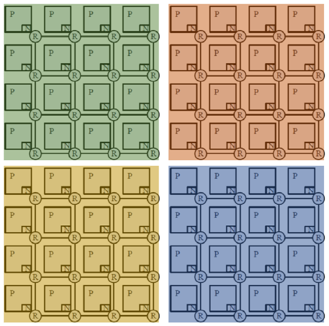
Network-on-Chip (1)



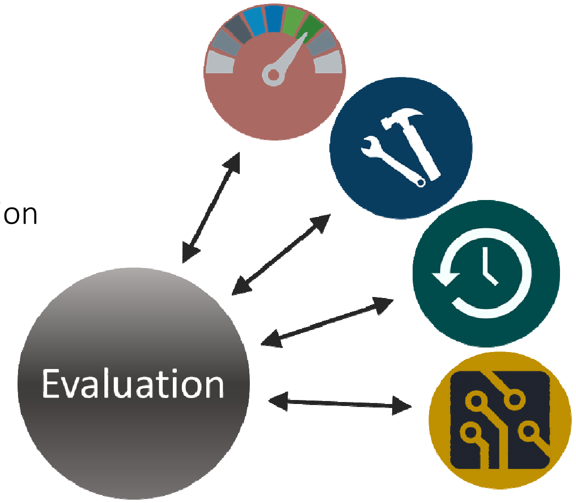
- Works as a communication subsystem for SoC
  - Design factors
    - Topology, routing algorithm, flow control, buffer size, hardware brand, flit size, …
- Major parts:
  - Chip Hardware
    - The electronic components of the circuit
  - Network Software
    - The software modules controlling the circuit
  - Application Software
    - The software running on this base



Network-on-Chip (2)

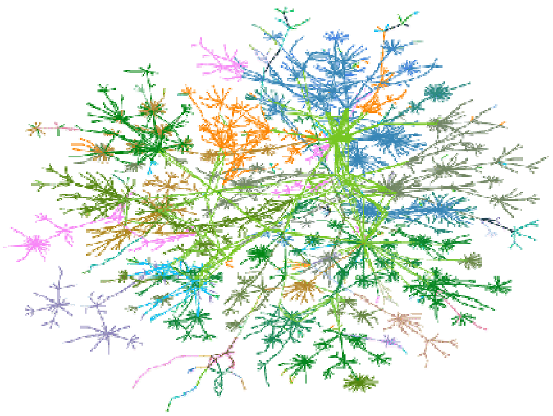


- Similar to combinational logic, parts (or the entire) NoC may operate independent of a clock signal
  - Globally Asynchronous Locally Synchronous (GALS) for large chips
  - Clock signal propagation issues
- NoC evaluation targets various aspects:
  - Performance
    - avg. latency, worst case latency, queueing time, network capacity
  - Functionality
    - deadlock freeness of routing, fairness of arbitration, error correction
  - Time
    - In time delivery of time sensitive information
  - Physical
    - Energy consumption, heat generation



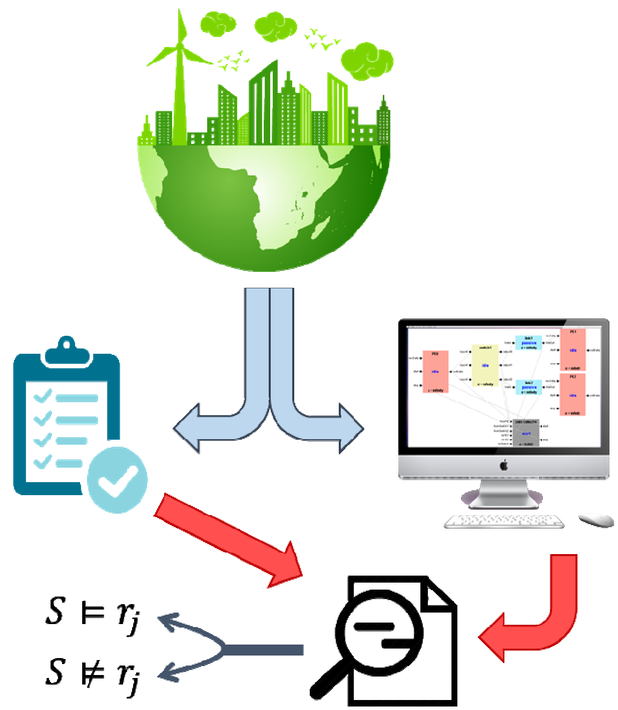
Model Checking (1)

- Exhaustively determining whether a model meets certain properties
  - Properties are derived from requirements (QoS, safety, liveness, etc.)
  - Why? Deciding whether a system meets a certain property is undecidable
  - When? For critical systems as a full-proof method of verification
- Issues
  - State explosion problem
    - The state space rapidly grows in size
  - Various methods to manage the size
    - Symbolic model checking
    - Bounded model checking
    - Abstraction



Model Checking (2)

- Various formalisms/method are introduced for model checking systems:
  - Timed Petri nets
  - Timed Automata (and its variations)
  - DEVS-based approaches (FD-DEVS, FP-DEVS)
- Major efforts for model checking
  - Use abstraction to simplify the model
    - Abstracting out information flow in basic Petri net and TA
  - Remove stochasticity
    - FD-DEVS[1] (finite deterministic DEVS)
  - Use model conversion
    - Conversion to timed automata for RTA-DEVS; model check using UPPAAL
    - Conversion to non-deterministic automata for FD-DEVS; model check using SPIN/PROMELA



[1] Hwang, M., and B.P. Zeigler. "Reachability graph of finite and deterministic DEVS networks." IEEE Transactions on Automation Science and Engineering 6, No. 3 (2009): 468-478.

DEVS M&S (1)

- Parallel DEVS models are made by atomic/coupled models

$$\textit{Atomic DEVS}: \langle X, S, Y, \delta_{ext}, \delta_{int}, \delta_{conf}, \lambda, ta \rangle$$

- X: Input Events
- S: State Set
- Y: Output Events
- δ_{ext}: External Transition Function
- δ_{int}: Internal Transition Function
- δ_{conf}: Confluent Transition Function
- λ: Output Function
- ta: Time Advance Function

$ta: \mathrm{S} \rightarrow R^{+}_{0,\infty}$

$\delta_{ext}: \mathrm{Q} \times \mathrm{X} \rightarrow \mathrm{S}$ where $\mathrm{Q} = \{(\mathrm{s}, \mathrm{e}) \mid s \in S, 0 \leq e \leq ta(s)\}$

DEVS M&S (2)

- Coupled DEVS models define couplings between Atomic/Coupled models
  - No behavior (external/internal transition functions or output function) for coupled models

$$Coupled\ DEVS:\ \langle X, Y, D, \{M_d\}, EIC, EOC, IC \rangle$$

Input Events

Output Events

Index set
$d \in D$

Set of atomic/coupled models

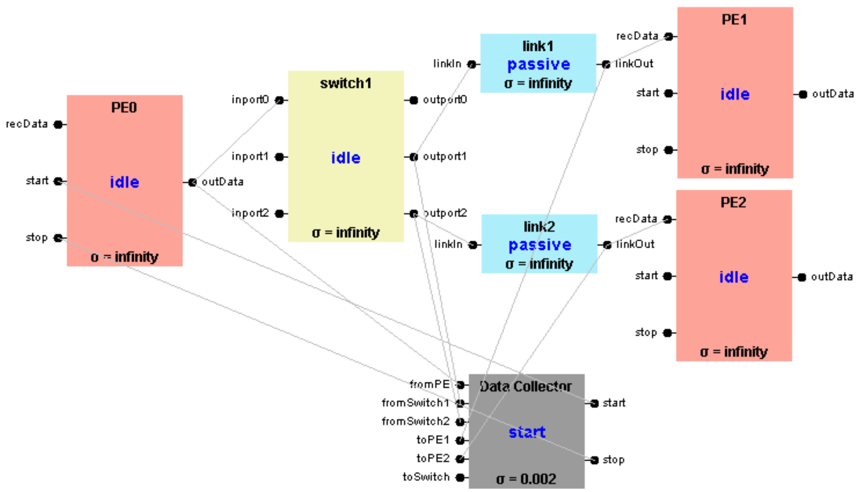
External Input Couplings

External Input Couplings

Internal Couplings

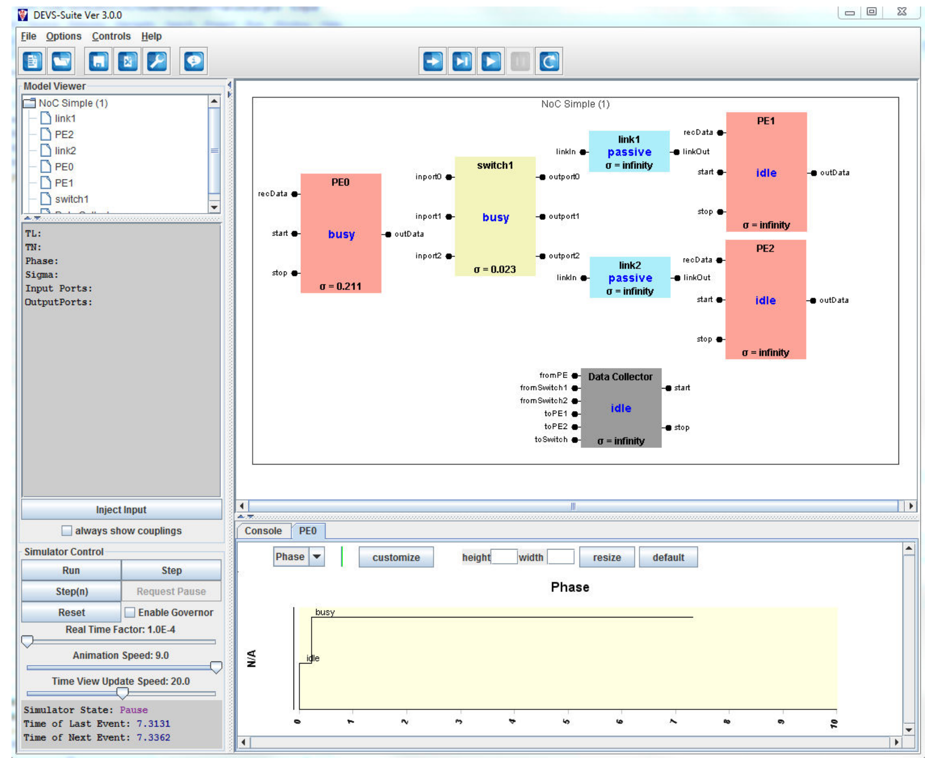
DEVS M&S (3)

- DEVS Modeling
  - Features
    - Continuous time, discrete event
    - Parallel
    - Synchronized time between models
    - Reactive
- DEVS Simulation
  - Can be conducted in
    - Logical time: time is advanced to the most urgent event
    - Real-time: simulation time is synchronized with the physical clock
  - Various implementations
    - eCD++, DEVS-Suite, MS4Me



DEVS M&S (4)

- DEVS-Suite
  - Model development through coding
  - Discrete Event Simulation
  - Model visualization, Simulation animation
  - Tracking
    - Time View (basic types)
    - Superdense time
  - Add-on libraries
    - Real-time simulation
    - Network-on-Chip
    - Real-time hardware interaction
    - RTL DEVS
    - EMF-DEVS (Eclipse Modeling Framework)

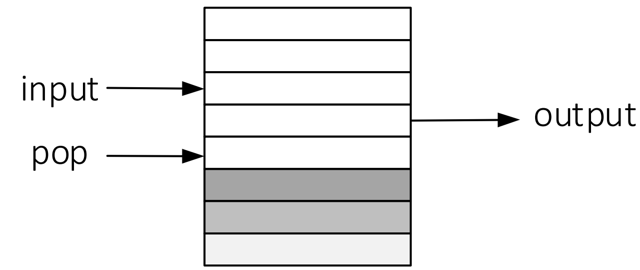


H.S. Sarjoughian, S. Sundaramoorthi, 2015, "Superdense Time Trajectories for DEVS Simulation Models", TMS/DEVS Symposium, Washington DC.

Constrained DEVS and Model Checking

Model Checking in DEVS – Example

- DEVS models are not well-suited for model checking



$$S = \overbrace{\{Active, Idle\}}^{Phase} \times \overbrace{\sigma}^{sigma} \times \overbrace{\mathbb{N}^8}^{values} \times \overbrace{\mathbb{N}}^{index} \times \overbrace{\beta}^{popped}$$

$$X = \{(input, \mathbb{N}), (pop, 1)\}$$

$$Y = \{(output, \mathbb{N})\}$$

$$\delta_{ext}((Idle, \sigma, values[0..7], index, \emptyset), e, (input, x)) = \begin{cases} (\dots, index + 1, \emptyset) \text{ where } values[index] = x \text{ if } index < 7 \\ (\dots, index, \emptyset) \text{ if } index = 7 \end{cases}$$

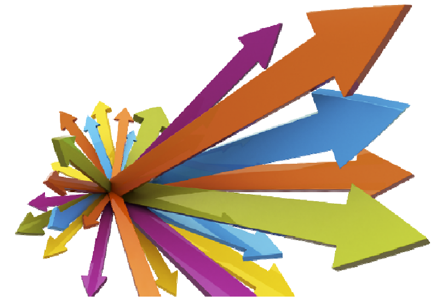
$$\delta_{ext}((Idle, \sigma, values[0..7], index, \emptyset), e, (pop, x)) = \begin{cases} (Active, \dots, index - 1, values[index]) \text{ if } index > 0 \\ (Idle, \dots, index, \emptyset) \text{ if } index = 0 \end{cases}$$

$$\delta_{int}(Active, \sigma, values[0..7], index, popped) = (Idle, \infty, values[0..7], index, \emptyset)$$

$$\lambda(Active, \sigma, values[0..7], index, popped) = (output, popped)$$

Model Checking in DEVS – Shortcomings

- Earlier approaches have certain shortcomings
  - Non-determinism and stochasticity
    - Stochasticity: randomness in models
    - Non-determinism: possibility of multiple states at one instance of time



  - Property checking capabilities
    - Specific languages for model checking
    - Limited expressive power
      - Deadlock detection vs. minimum accepted quality of service for specific data

  - Data exchange constraints
    - Some modeling languages do not support complex data flow
      - Such as Petri net and timed automata
    - NoC component models requires exchanging complex data types
      - How does one verify those models?

Model Checking in DEVS – Requirements

- What do we need to make DEVS verifiable (via model checking)?
- Answer:
  - Constrain state set and input set values
  - Discretize time for input events
  - Finite number of internal transitions
- Example:
  - Complex data type containing an array of strings (of size 8 holding strings of size 24) and integers under 10

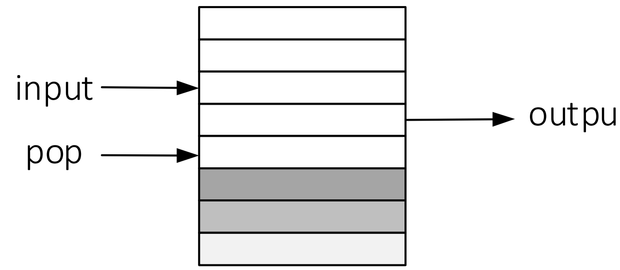
Array of strings: $\left((Char)^{24}\right)^{8}$

Numbers: $[(1|2|3|4|5|6|7|8|9) \in Integer]$

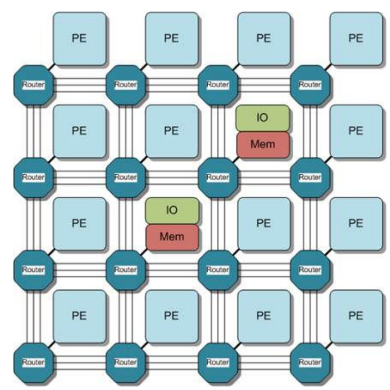
Entire state space: $\left((Char)^{24}\right)^{8} \times (1|2|3|4|5|6|7|8|9) \longrightarrow \left((Char)^{24}\right)^{8} \times [Integer < 10]$

Model Checking in DEVS – NoC

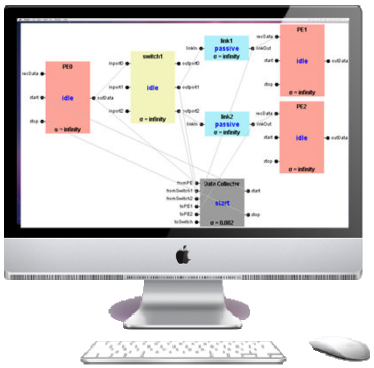
- How the stack model changes?
- Answer:
  - No more than 8 numbers in the stack
  - Only positive numbers less than 10
  - Time resolution for input events (new input or pop) has the granularity of 1 cycle



- How do we leverage this for modeling NoC?
  - Data is only limited to flits and flow control signals
  - Events can only happen at clock edges
- What is our property checking method?
  - We use the experimental frame (EF)



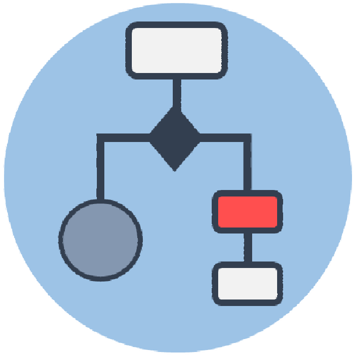
Tool Support

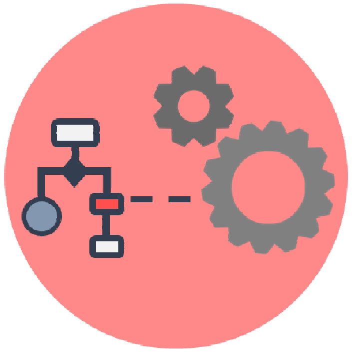


DEVS-Suite Extensions

- DEVS-Suite were extended to support

1. Constrained-DEVS modeling
    - Base classes for constrained state variables
    - Invalid state specification
    - Initial state set
    - Input/output value sets
2. Constrained-DEVS execution
    - State space exploration for model checking mode
    - Invalid state reporting for model checking mode
    - Parallel DEVS execution for simulation mode
    - Model checking engine uses the simulation for state exploration

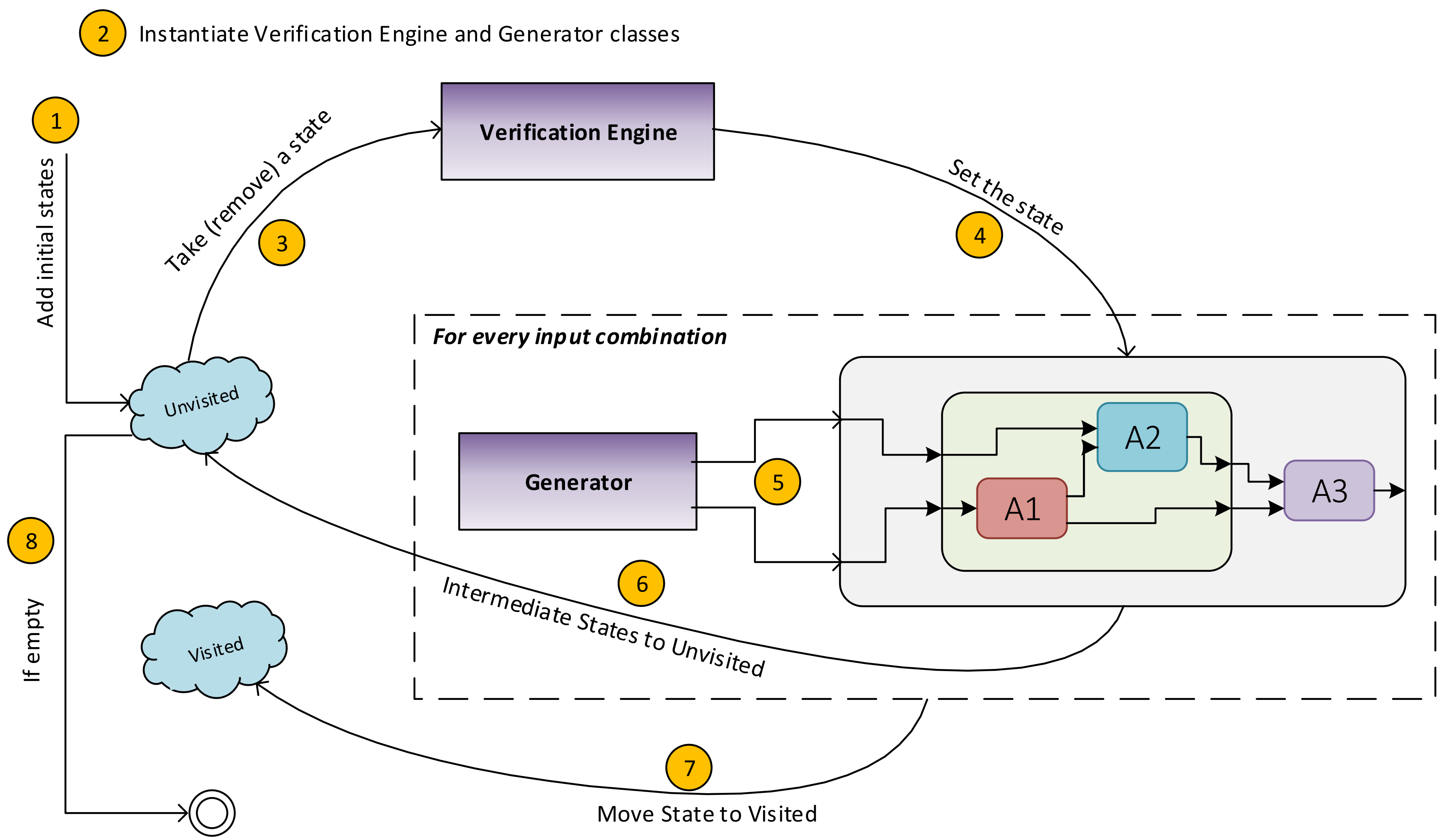




DEVS-Suite State Space Exploration Protocol

- In model checking mode, DEVS-Suite carries out the following steps:
  - Initialization
    - Model is loaded, state variables are recognized, input ports identified
    - *Verification Engine* and *Generator* models are is instantiated
    - Initial states are put into *Unvisited* data structure
  - Main Loop: take state from *Unvisited*, set the state of the model
    - Nested Loop: apply all combinations of input to the model
      - Store resulting states (if not seen before) into the *Unvisited*
    - Add the original state to the *Visited* data structure
    - Continue until *Unvisited* is empty

- Transducer model(s) stores the trace and verifies properties

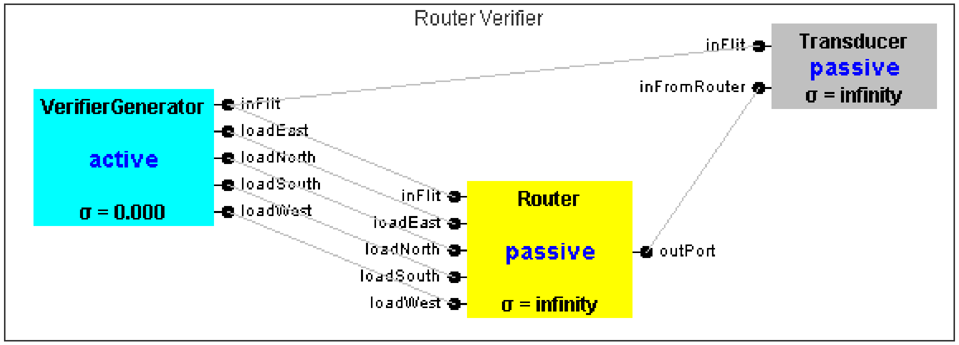


2
Instantiate Verification Engine and Generator classes
1
Add initial states
Take (remove) a state
3
Verification Engine
Set the state
4
For every input combination
Unvisited
Generator
5
A2
A1
A3
8
If empty
6
Intermediate States to Unvisited
Visited
7
Move State to Visited

Atomic Model Verification

- DEVS-Suite experimentation is based on Experimental Frame (EF)
  - Data generation by *Generator*
  - Data collection and analysis by *Transducer*

- Model checking a minimal adaptive router
  - The *Generator* injects flits and traffic information
  - *Transducer* collects outgoing flits and verifies whether the routing decision is correct



Adaptive Router – DEVS Model

$$S = \overbrace{\{Active, Idle\}}^{Phase} \times \overbrace{\sigma}^{sigma} \times \overbrace{\{1,2,3\}}^{Load\ East} \times \overbrace{\{1,2,3\}}^{Load\ North} \times \overbrace{\{1,2,3\}}^{Load\ West} \times \overbrace{\{1,2,3\}}^{Load\ South} \times \overbrace{\{0,1,2,3,4\}}^{target\ port} \times \overbrace{\{x < 10\}}^{xPos} \times \overbrace{\{y < 10\}}^{yPos}$$

$$X = \{(inFlit, \{0,1\}^{24}), (loadEast, \{1,2,3\}), (loadNorth, \{1,2,3\}), (loadWest, \{1,2,3\}), (loadSouth, \{1,2,3\})\}$$

$$Y = \{(outPort, \{0,1,2,3,4\})\}$$

$$\delta_{ext}\big((Idle, \sigma, LE, LN, LW, LS, targetPort), e, (loadEast, x)\big) = (Idle, \sigma, x, LN, LW, LS, targetPort)$$

$$\delta_{ext}\big((Idle, \sigma, LE, LN, LW, LS, targetPort), e, (loadNorth, x)\big) = (Idle, \sigma, LE, x, LW, LS, targetPort)$$

$$\delta_{ext}\big((Idle, \sigma, LE, LN, LW, LS, targetPort), e, (loadWest, x)\big) = (Idle, \sigma, LE, LN, x, LS, targetPort)$$

$$\delta_{ext}\big((Idle, \sigma, LE, LN, LW, LS, targetPort), e, (loadSouth, x)\big) = (Idle, \sigma, LE, LN, LW, x, targetPort)$$

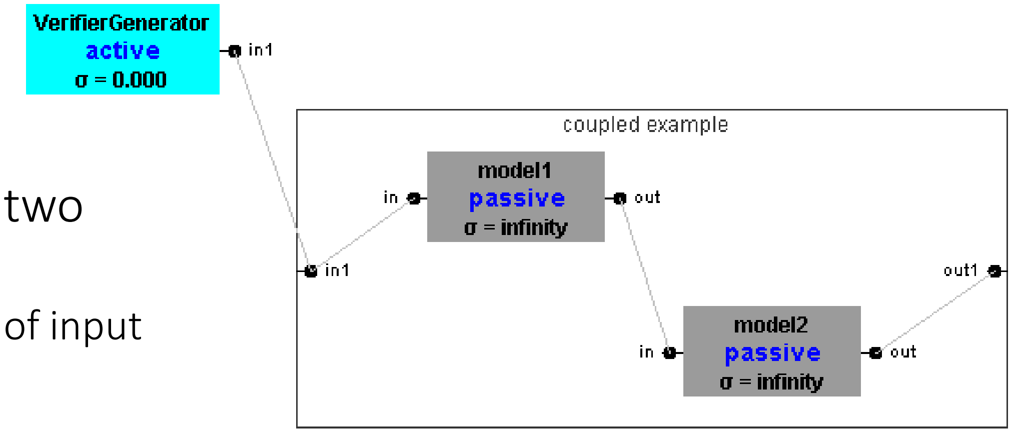
$$\delta_{ext}\big((Idle, \sigma, LE, LN, LW, LS, targetPort), e, (inFlit, x)\big) \longrightarrow \begin{cases} (Active, rDelay, LE, LN, LW, LS, 0) & if\ xPos = x \wedge yPos = y \\ (Active, rDelay, LE, LN, LW, LS, 1) & if\ xPos > x \wedge yPos = y \\ (Active, rDelay, LE, LN, LW, LS, 2) & if\ xPos = x \wedge yPos < y \\ (Active, rDelay, LE, LN, LW, LS, 3) & if\ xPos < x \wedge yPos = y \\ (Active, rDelay, LE, LN, LW, LS, 4) & if\ xPos = x \wedge yPos > y \\ (Active, rDelay, LE, LN, LW, LS, 1) & if\ xPos > x \wedge yPos < y \wedge LW \le LN \\ (Active, rDelay, LE, LN, LW, LS, 1) & if\ xPos > x \wedge yPos > y \wedge LW \le LS \\ (Active, rDelay, LE, LN, LW, LS, 2) & if\ xPos > x \wedge yPos < y \wedge LN < LW \\ (Active, rDelay, LE, LN, LW, LS, 2) & if\ xPos < x \wedge yPos < y \wedge LN < LE \\ (Active, rDelay, LE, LN, LW, LS, 3) & if\ xPos < x \wedge yPos < y \wedge LE \le LN \\ (Active, rDelay, LE, LN, LW, LS, 3) & if\ xPos < x \wedge yPos > y \wedge LE \le LS \\ (Active, rDelay, LE, LN, LW, LS, 4) & if\ xPos < x \wedge yPos > y \wedge LS < LE \\ (Active, rDelay, LE, LN, LW, LS, 4) & if\ xPos > x \wedge yPos > y \wedge LS < LW \end{cases}$$

$$\delta_{int}(Active, \sigma, LE, LN, LW, LS, targetPort) = (Idle, \infty, LE, LN, LW, LS, targetPort)$$

$$\lambda(Active, \sigma, LE, LN, LW, LS, targetPort) = (outPort, targetPort)$$

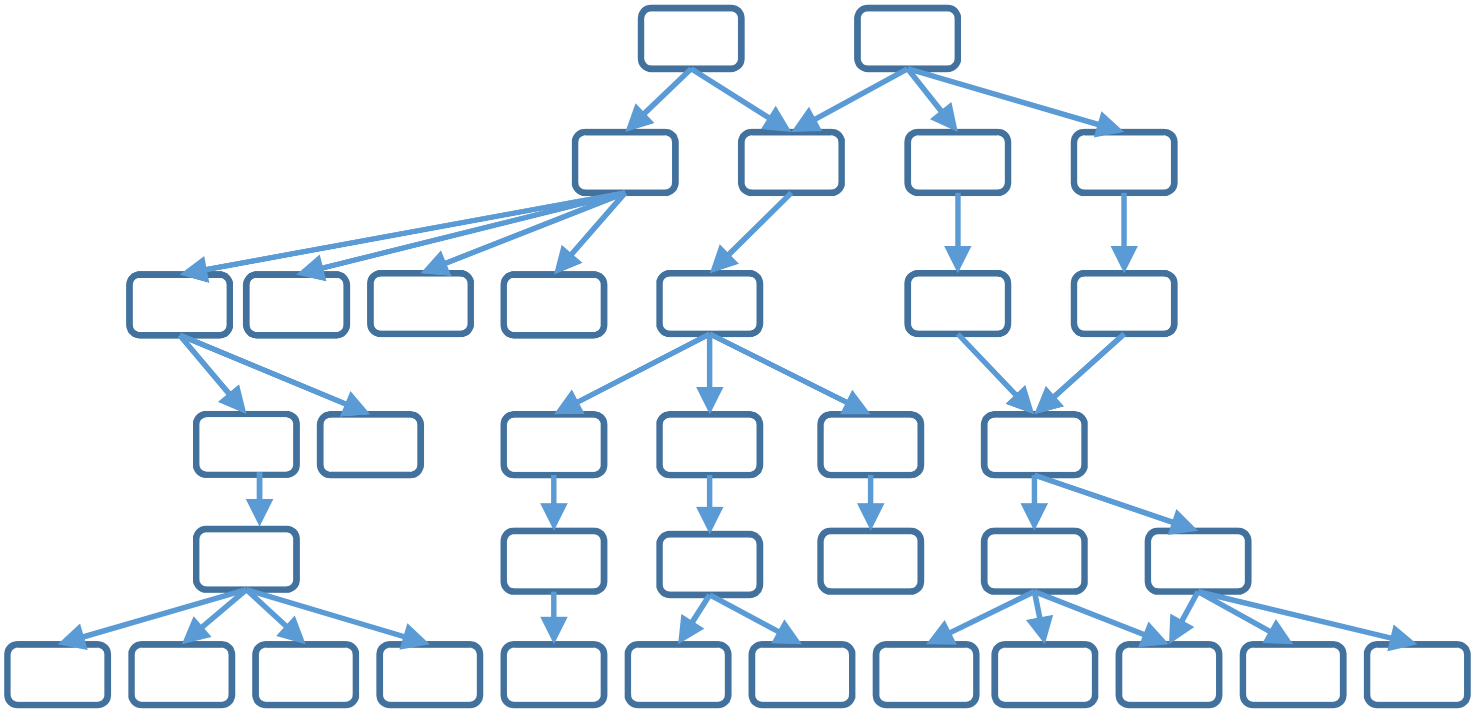
Coupled Model Verification

- Works similar to the atomic version
  - The *generator* injects data based on the input ports of the coupled model
  - The state of the coupled model is the aggregate state of inner models

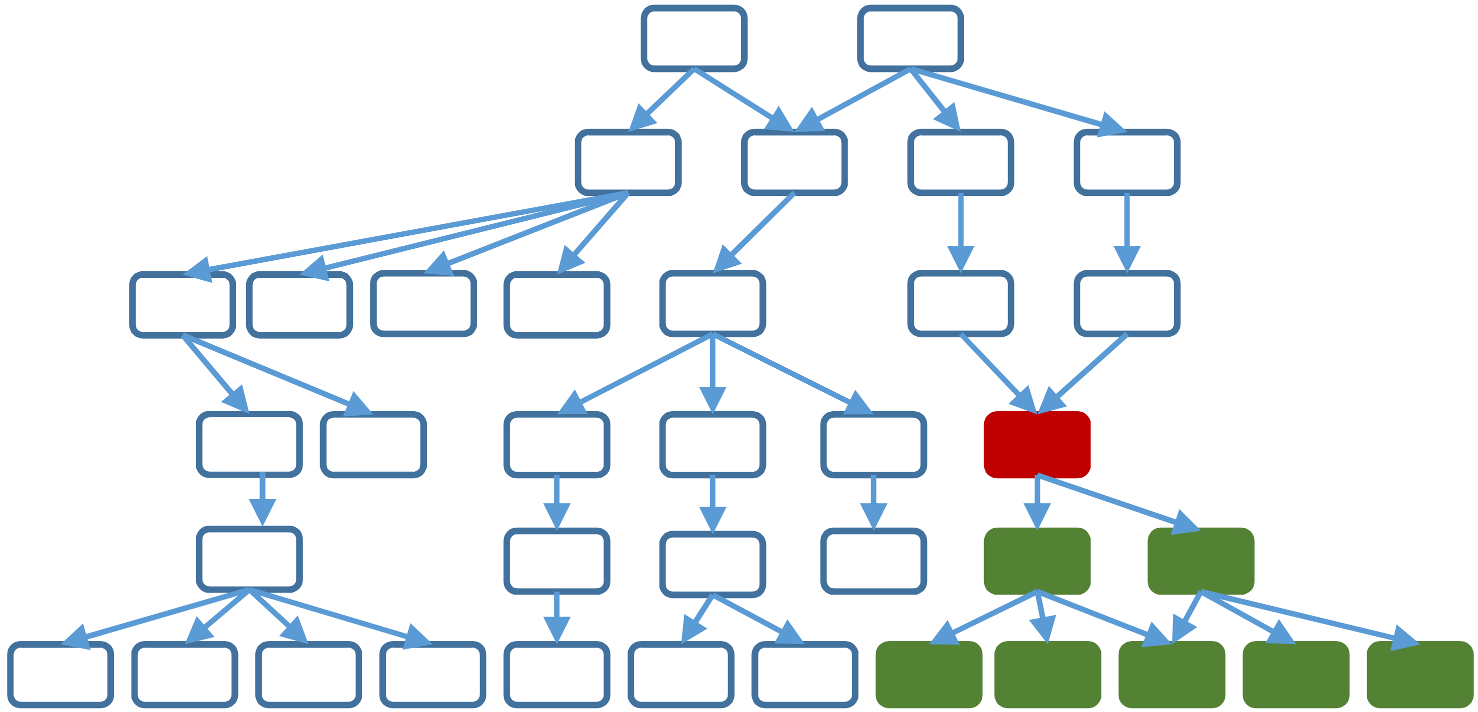


- Model checking a coupled model with two inner components
  - *VerifierGenerator* injects all combinations of input values for model1

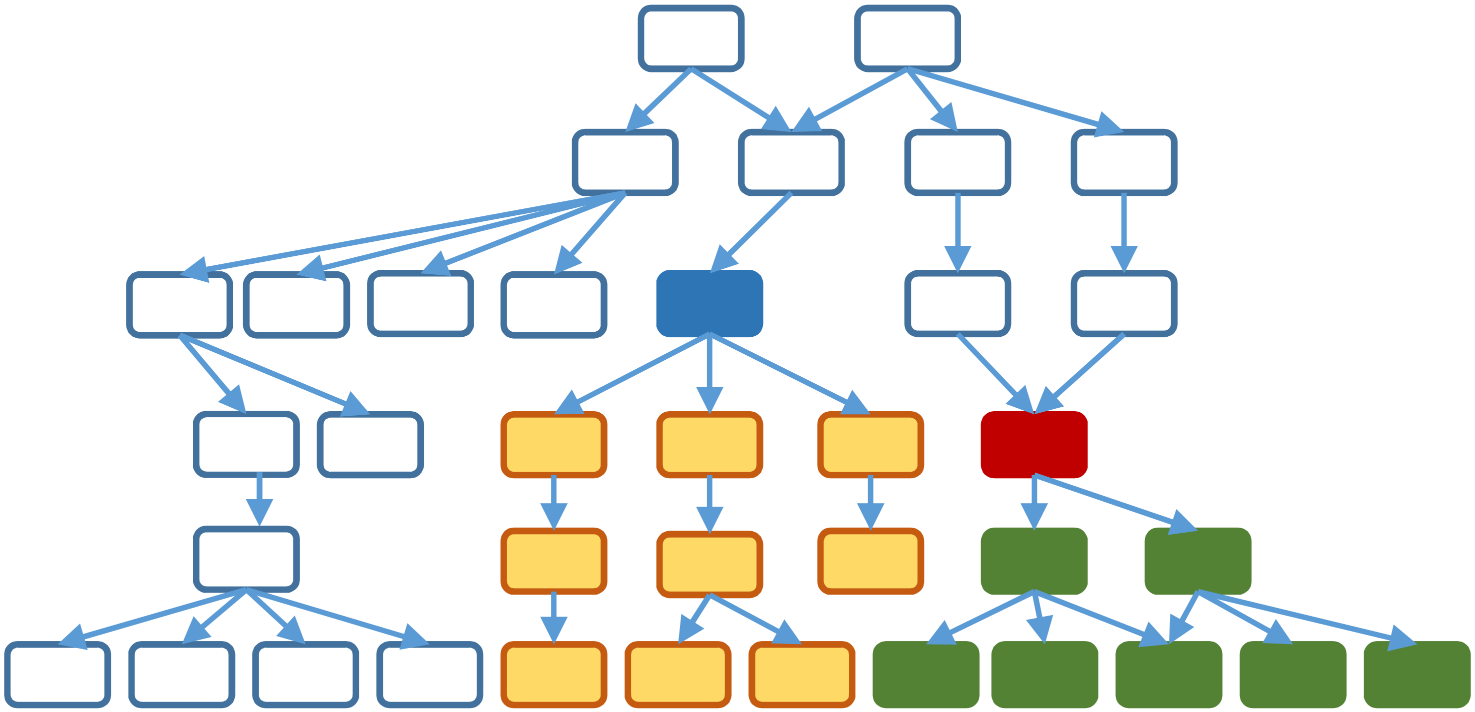
Analyzing Traces



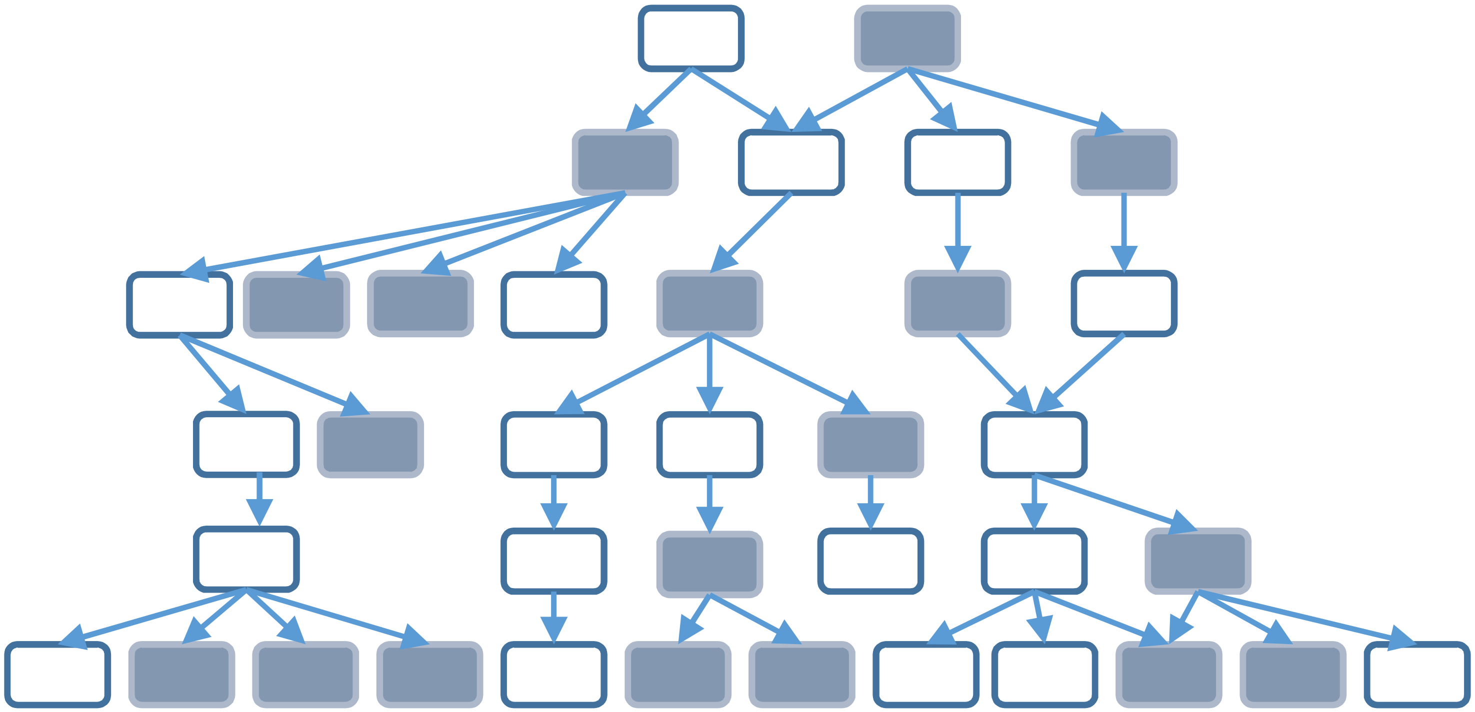
Analyzing Traces



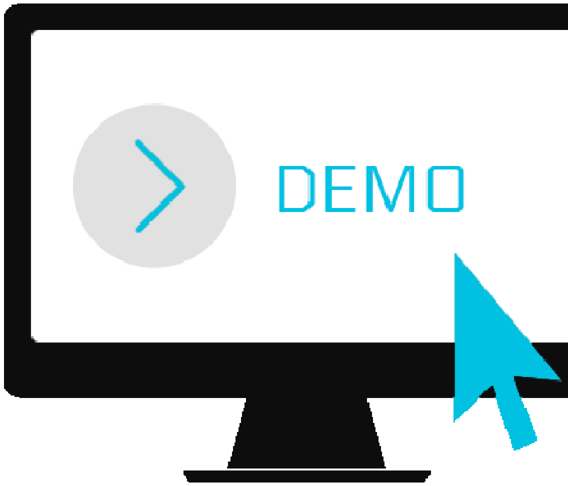
Analyzing Traces



Analyzing Traces



Demo



Conclusion & Future Work

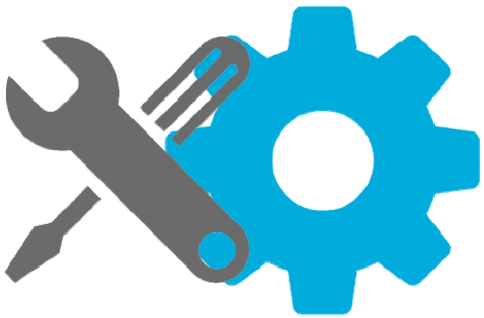
Conclusion

- Model checking capability
  - Constrained-DEVS formalism for model checking
  - State exploration algorithm for constrained-DEVS models
- An attempt toward unified design environments
  - With support for simulation & model checking
  - EF-based experimentation and model evaluation

Future Work

- Ongoing
  - Hardware-level model library for NoC using Constrained-DEVS
  - Integration with multiresolution modeling – the right abstraction is chosen automatically based on the property which is being verified

- A new version of DEVS-Suite (v 4.0) is scheduled for release by the end of summer 2017
  - Contains the verification engine for Constrained-DEVS models



Thank You