Causal Block Diagram: compiler to LaTeX and DEVS

Nicolas Demarbaix

University of Antwerp

Overview

- Introduction
- Theoretical background
- Design & Implementation
- Conclusion

Introduction

Introduction

Why?

- Clear and detailed description in LaTeX
- Simulating model using PypDEVS
- Plugging CBD as component into CoupledDEVS

Introduction

How?

- Using Python based CBD model
- Compile to LaTeX format
- Compile to DEVS format

Technical background

Technical background

Preparation

- Construction of Dependency Graph
- Identify Strong Components
- Identify cyclic Components and verify Linearity

Technical background

CBD to DEVS mapping

- Causal Block Diagram -> Single Atomic Devs
- Strong Component -> State in the Atomic Devs
- Block Computation in State

Technical background

Dynamic Rate Change

- Check allowed user-defined deviation
- Update block rate accordingly

Christis, N., 2012-2013. Research internship 2: Hybrid systems.

Design & Implementation

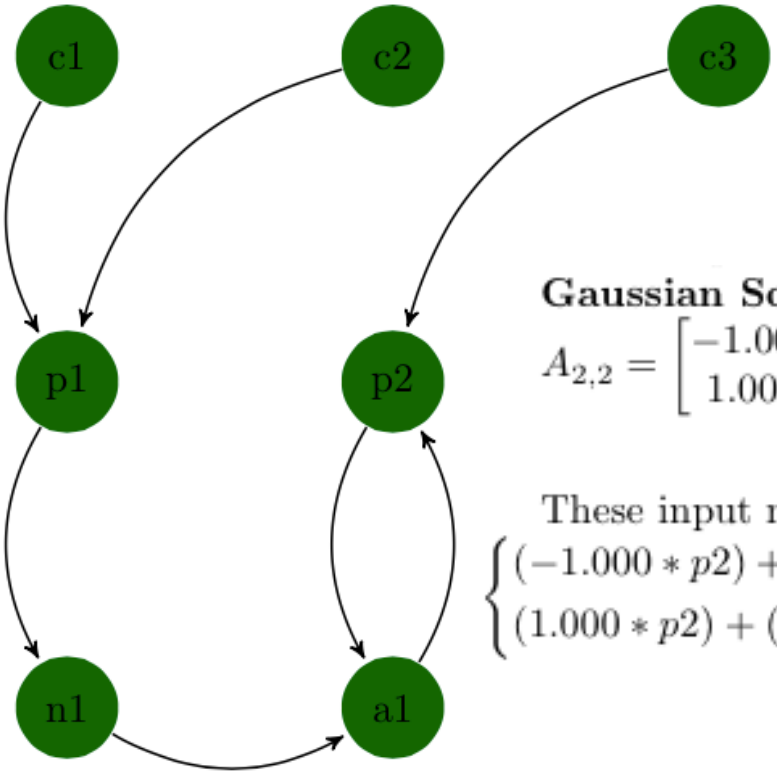
General Design

- Starting from an arbitrary CBD model
- Flatten the CBD model if it is hierarchical
- Construct the Dependency Graph
- Check for algebraic loops and linearity

Design - CBD to LaTeX

- 3 general parts
  1. Causal Block Diagram description
  2. Dependency Graph description
  3. Solution

CBD to LaTeX example



Gaussian Solver input for component 6

$$A_{2,2} = \begin{bmatrix} -1.000 & 2.000 \\ 1.000 & -1.000 \end{bmatrix} \quad B_{1,2} = \begin{bmatrix} 0.000 \\ 10.000 \end{bmatrix}$$

These input matrices correspond to the following system of equations:

$$\begin{cases} (-1.000 * p2) + (2.000 * a1) = 0.000 \\ (1.000 * p2) + (-1.000 * a1) = 10.000 \end{cases}$$

Implementation - CBD to LaTeX

- Graphical Representation using TikZ package
- Document contains information about:
    1. Block type and equation
    2. Component structure and block dependencies
    3. Solution (either direct or via solver)

Design - CBD to DEVS

- 3 parts
  1. CBDState = State Definition
  2. CBD = AtomicDEVS
  3. Root = CoupledDEVS

Implementation - CBD to DEVS

- Initially propagate block values
- For each non-determined component create a state
- Compute blocks upon transition:

  current component ⟶ next component

- Compute linear algebraic loops using Gaussian Solver

CBD to DEVS example

```
class CBDState:
    def __init__(self, name="", block_values=dict()):
        self.name = name
        self.rate = block_values["rate"]
        self.iteration = block_values["iteration"]
        self.c1 = 5.0000
        self.c2 = 2.0000
        self.c3 = 2.0000
        self.p1 = 10.0000
        self.p2 = block_values["p2"]
        self.n1 = -10.0000
        self.a1 = block_values["a1"]


def intTransition(self):
    def cond_int_start_to_5():
        if self.state.n1 != None and self.state.c3 != None :
            return True
        else:
            return False
    def action_int_start_to_5():
        component = componentDict["5"]
        solverInput = self.constructLinearOutput(component)
        self.__gaussjLinearSolver(solverInput)
        solutionVector = solverInput[1]
        return {"a1": solutionVector[0], "p2": solutionVector[1]}
```

Conclusion

- The CBD to LaTeX Compiler provides clear information about the CBD
- The CBD to DEVS Compiler generates a AtomicDEVS model that produces the same results

Future work

- Complete dynamic rate monitoring
- Automatically provide input/output ports for the AtomicDEVS and the corresponding external transition

University of Antwerp