Early Analysis of Cyber-Physical Systems using Co-simulation and Multi-level Modelling

IEEE ICPS 2019, May 8, 2019

Thomas Nägele
t.nagele@cs.ru.nl

NWO Applied and Engineering Sciences

iCIS | Software Science
Radboud University

Contents

iCIS | Software Science
Radboud University

Introduction

- It is hard to gain insight in the behaviour of a CPS in development
  - Multi-disciplinary nature
  - Many different components
- Errors in the system's design are hard to fix later on
  - Time consuming, thus expensive
- Early co-simulation of the system allows early analysis
  - Early detection of errors
  - Design space exploration

Introduction

Goal

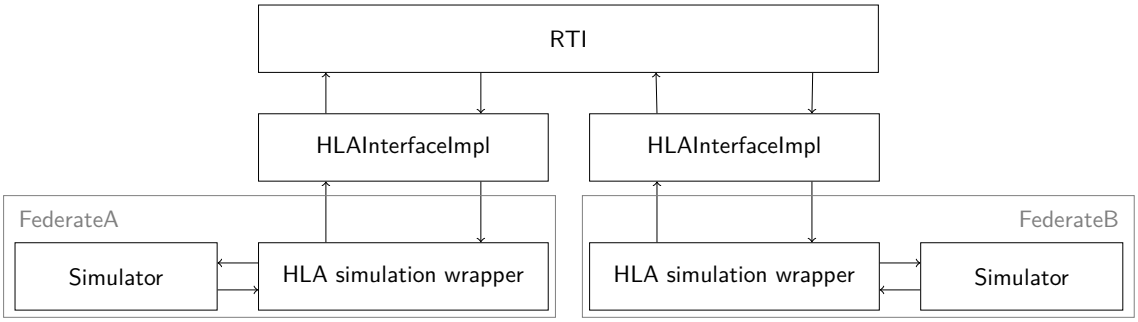
- Rapid construction of co-simulations of models
- Model or interface changes adapted quickly
- Wide support of modelling tools
- Use existing standard if possible

iCIS | Software Science
Radboud University

Background

High-Level Architecture (HLA)

- Architecture specification for co-simulation
- Handles time and attribute synchronisation
- Federation containing Federates
- One central Run-Time Infrastructure (RTI)
- Simulators in co-simulation must comply to this standard
  - Usually requires wrappers to be implemented



Background

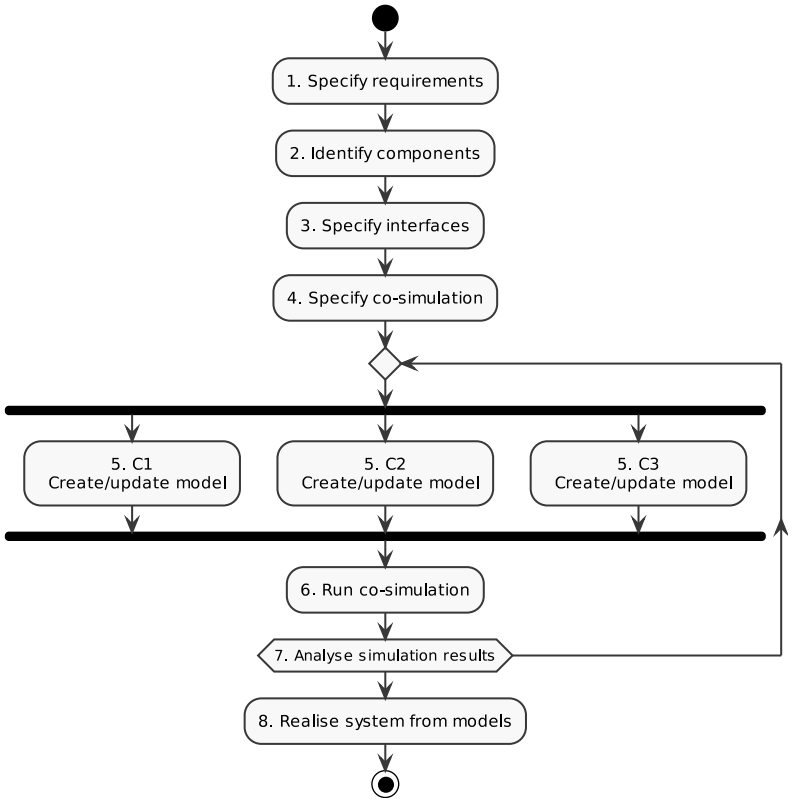
Functional Mock-up Interface (FMI)

- Standard interface to support co-simulation
- Functional Mock-up Units (FMUs)
  - Contains model description, binaries and/or sources
  - Binaries can be simulated
- Co-simulation requires master algorithm
- Supported by a wide range of modelling tools

Current focus

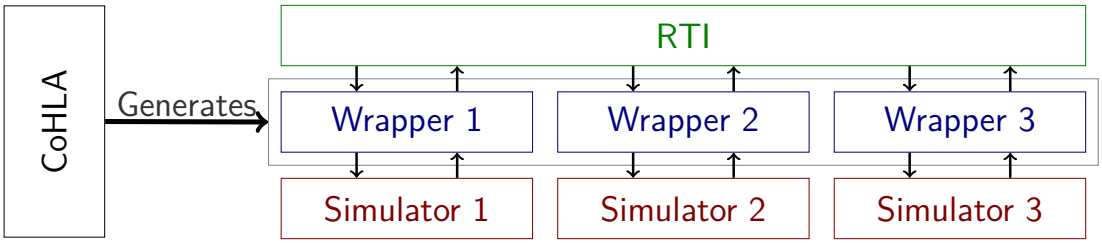
- Construct a co-simulation early in the development process
- Apply variations rapidly to support the design phase

Approach



Approach

- Allowing to create or adapt co-simulations early and quickly
- Co-simulation construction using CoHLA
  - Domain specific language (DSL) to specify co-simulations
  - Using HLA and FMI standards
  - Supports discrete-time POOSL models
    - POOSL: Parallel Object-Oriented Specification Language
    - Suitable for modelling software architectures



Case study: SliderSetup

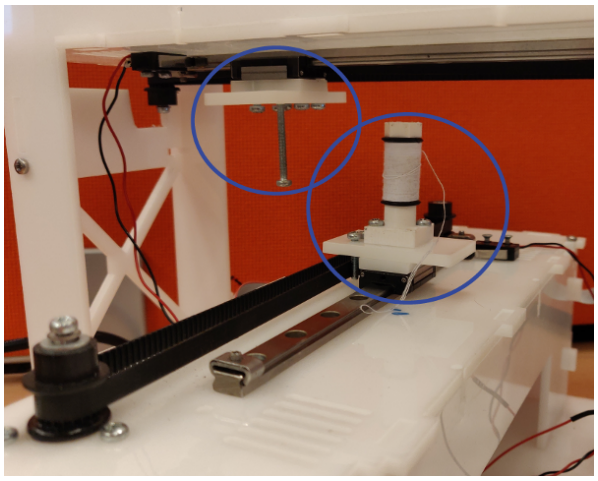
- Consists of two independent, intersecting and movable axes
- Goal: Unwind thread from one coil to the other
- Sliders must orbit around each other...
- ...without colliding

Requirements

1. The system is capable of (un)winding the thread coil at a minimum speed of 2 rotations per second.
2. The components of the system may not collide.

Case study: Slider setup

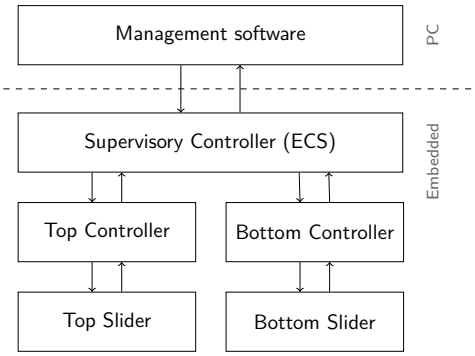
- Consists of two independent, intersecting and movable axes
- Goal: Unwind thread from one coil to the other
- Sliders must orbit around each other...
- ...without colliding



SliderSetup development

Component interfaces

- Plant dynamics (axes)
  - Inputs: voltage, enabled state
  - Outputs: position
- Control laws (controllers)
  - Inputs: axis position, set-point, duration
  - Outputs: voltage, enabled state, estimated position
- Embedded control software (supervisory controller)
  - Inputs: estimated positions
  - Outputs: set-points and durations



SliderSetup development

Modelling

- Plant dynamics (axes)
  - Continuous-time model (20-sim)
  - Describes the motor, belt, sensor and the slider itself
  - 3D model is created for visualisation and collision detection
- Control laws (controllers)
  - Discrete-time model (20-sim)
  - Possibility to generate C code
- Embedded control software (supervisory controller)
  - Discrete-time model (POOSL)
  - Coordinates both sliders via their controllers
  - Allows for more complex actions
  - Provides an interface for management software

SliderSetup development

Co-simulation

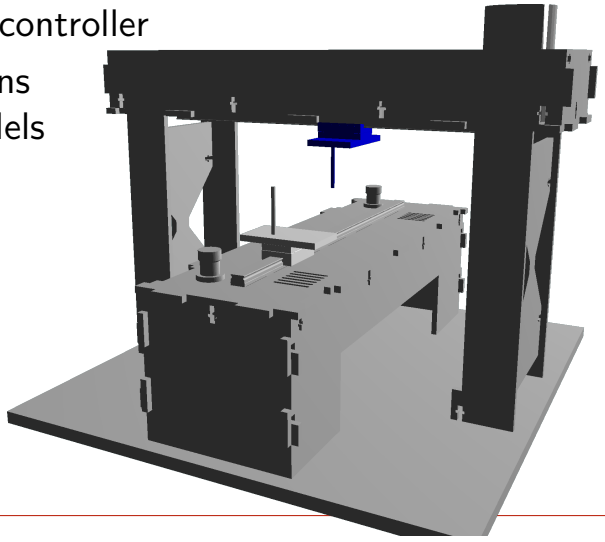
```
1  | FederateClass Axis {
2  |   Type FMU
3  |   Attributes {
4  |     Input Boolean enable
5  |     Input Real motor
6  |     Output Real encoder
7  |     Output Real position
8  |   }
9  |   Parameters {
10 |     ...
11 |   }
12 |   TimePolicy RegulatedAndConstrained
13 |   DefaultModel "models/SliderAxis.fmu"
14 |   AdvanceType TimeAdvanceRequest
15 |   DefaultStepSize 0.0005
16 |   DefaultLookahead 0.0001
17 | }
```

```
1  | Federation SliderSetup {
2  |   Instances {
3  |     bottomAxis : Axis
4  |     topAxis : Axis
5  |     bottomController : Controller
6  |     topController : Controller
7  |     hlController : HighLevelSliderController
8  |   }
9  |   Connections {
10 |     { bottomController — bottomAxis }
11 |     { topController — topAxis }
12 |     { bottomAxis.enable <— hlController.
   |           bottomEnable }
13 |     ...
14 |     { topController.mode <— hlController.topMode
   |           }
15 |   }
16 | }
```

A co-simulation of the system can now be executed!

Refinement

- Co-simulation is now possible in an early stage
  - No need to integrate software models
- Allows for impact analysis of design decisions
- Initial design already showed one issue
  - Updated write sequence to controller
- Allows for refinement in iterations
  - Increasing detail in the models
  - Maybe even replace models

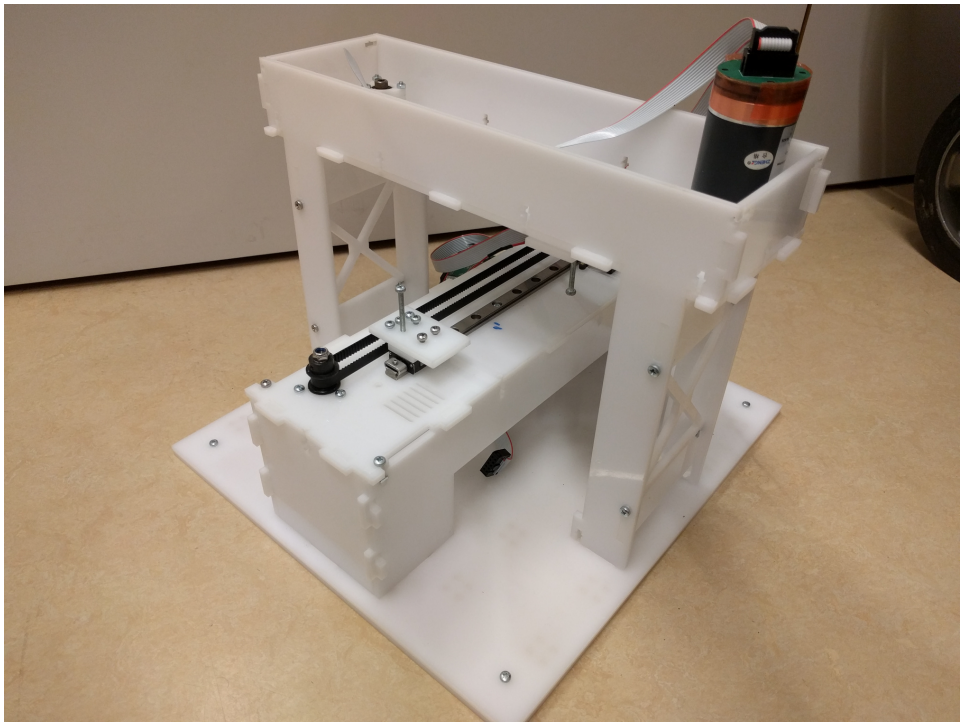


Realisation

- Available motors were selected
- Raspberry Pi 3 for embedded software
  - Running an image from the Yocto Project[1]
  - Implementation of ECS
    1. Implementation of interface
       ⇒ POOSL to C++
    2. Implementation of control loops
       ⇒ Generated by 20-sim
    3. Final implementation steps
       ⇒ Implementation of management interface
- Implementation of management software
  - Controls both the co-simulation and real system

[1] https://www.yoctoproject.org/

Realisation



Conclusion

- CoHLA supports system-level analysis in an early stage
- Design errors were found early
- System changes can be adapted easily in CoHLA
- Design of a small CPS illustrated the approach
- Previous work shows scalability of CoHLA
  - Distributed co-simulation in the cloud

Questions

Thanks for your attention. Are there any questions?

iCIS | Software Science
Radboud University