Co-simulation

Bert Van Acker, Cláudio Gomes, Joachim Denil and Bart Meyers
The Modern Car

- Complexity
  - 40+ subsystems
- Competitive Market
- Concurrent Development
  - Late Integration Problems
- Distributed Development
  - Specialized suppliers
  - Late Integration (due to IP)
M&S in MBSE

• V-Process
  – Design
    • Requirements (0D model)
    • Dynamics (1D model)
    • Mesh (3D model)
  – Validation
    • Reuse design experimentation results

• Simulation in all stages
• V-process also applies to more complex systems
M&S in MBSE

• Early access to models of components.
  – Test different control approaches
  – Evaluate same component from different suppliers

• Challenges:
  – Different teams/suppliers use different modelling tools
  – IP Protection

from www.ni.com/
A Solution: Remote Simulators

- Suppliers make a simulator available as a web service
  - Integrator takes care of programming an interface
  - Good IP Protection
  - Different suppliers require different interfaces
Solution: Functional Mock-up Interface Standard [2]

- Simulator and model exported as a standardized C library
- Standard interaction with any simulator
- Every simulator is a black box.
  - Executed locally but can communicate with a remote server
Functional Mock-up Interface Standard

• A Functional Mockup Unit is a zip-file (.fmu) consisting of
  – C Library (.dll or .so)
  – XML file (metadata)

• The coupling (a.k.a master algorithm) must be provided
FMU Example

```c
fmi2Status fmi2DoStep(fmi2Component fc, fmi2Real currentCommPoint, fmi2Real commStepSize, fmi2Boolean noPrevFMUState)
{
    FMUInstance* fi = (FMUInstance *)fc;
    fmi2Status simStatus = fmi2OK;
    printf("%s in fmiDoStep()\n", fi->instanceName);
    fi->currentTime = currentCommPoint + commStepSize;
    printf("Motor_in: %f\n", fi->r[_motor_in]);
    printf("slave CBD_PART2 now at time: %f\n", fi->currentTime);

    fi->r[_position] = fi->r[_position] + fi->r[_velocity] * commStepSize;
    fi->r[_velocity] = fi->r[_velocity] + fi->r[_acceleration_after_friclion] * commStepSize;
    fi->r[_friction] = fi->r[_velocity] * 5.81;
    fi->r[_motor_acceleration] = fi->r[_motor_in] * 40;
    fi->r[_acceleration_after_friclion] = fi->r[_motor_acceleration] - fi->r[_friction];

    return simStatus;
}

fmi2Status fmi2GetReal(fmi2Component fc, const fmi2ValueReference vr[], size_t nvr, fmi2Real value[])
{
    FMUInstance* comp = (FMUInstance *)fc;
    int i;
    for (i = 0; i < nvr; i++)
    {
        value[i] = comp->r((vr[i]));
    }
    return fmi2OK;
}
```
FMU States

• Master algorithm
  – Communicates with each individual simulator
  – Moves data from one simulator to the other
  – Coordinates time
Co-simulation

• Simulation of a system
  – Coupling of multiple simulators
  – Optionally as black-boxes
  – Each simulating one or more models
  – Built with different formalisms/tools.

• Co-simulation scenario
  – Description of the system
  – The simulators and their dependencies
  – Data about the capabilities of each simulator.
Correct Co-simulation

• Correct simulation trace
  – Accuracy
  – Accumulated error between ideal (analytical) trace of the system and the simulated trace.
    • Ideal solution given by the formal semantics.

• Correct co-simulation trace
  – Ideal solution given by the formal semantics of the composition of the languages.
Challenges

• Factors enabling/inhibiting correctness:
  – Physics
  – Numerical
  – Computer Science
Challenges: Physics

• Incompatible units
  – Eg.: metric with imperial, voltage, etc...

• Invalid quantities
  – Eg.: negative concentration
Challenges: Numerical

• Time synchronization
  – Correct interleaving of the execution of each simulator.
  – Including data dependencies

• Time progression
  – No Zeno-behaviour if no such thing occurs in the ideal solution

• Algebraic (instant) dependencies
  – Detect and solve.
Challenges: Numerical

• Stability
  – Boundedness
  – A co-simulation solution is stable iff the ideal solution is stable
Challenges: Numerical

• State event location
  – Co-simulation traces must be valid
    • Reactions to changes must be communicated with a delay that approximates reality.
  • Electric coupling ≈ instant communication
Challenges: Computer Science

• Determinism
  – Uniquely defined behaviour of the coupling algorithm.

• Deadlocks

• Fairness
  – Every simulator gets a chance to execute

• Distribution
Challenges: Computer Science

• Real-time constrains
  – E.g., Hardware in the loop
• Make the most of heterogeneous capabilities
  – Fixed or adaptive time-step; no/single/multiple rollback support
• Hierarchical co-simulation
• Different information exposed about each simulator
  – No/Static/Dynamic IO Dependencies
  – No/Static/Dynamic Recommended step size
  – Jacobian matrices
  – Operating conditions (e.g., range of stability)
Current work: Semantic adaptation

- Correctness of a co-simulation scenario
- Requires formal semantics for the coupling of the languages used in the scenario.
- Leading to ad-hoc creation of hybrid languages
  - Reuse the semantics of each language.
  - Define the semantic adaptation for the interactions.
- Only after we can measure correctness
- White-box -> Black-box
Current work: Semantic adaptation

- Semantic adaptation of non-deterministic formalisms
  - Petri-nets
Current Work: Automatic Generation of Coupling

• Generic master is cumbersome
  – Too many capabilities to deal with
  – And varying levels of information exposure
• Error-prone

```
Algorithm 9 Generic MA - simulation step
#
# malloc for all output ports of all FMUs
# parse the XML files of all FMUs
#
time_step = 0
#
time = time_initial
schedule = LOOP_DETECT(DEPGRAF(fmu_IM))
#
while not end_condition do
  for blocks in schedule do
    if block is aBlockType then
      IL = getInpulList(block, fmu_interactionModel)
      for input in IL do
        linkedOutput = getCorrespondingOutput(input, fmu_IM)
        if input < (currentTime) then
          STORESTATE(previous_block)
          EXECUTE(previous_block)
          OL = getOutputList(previous_block, fmu_IM)
          for output in OL do
            output_struct[0] = output.value
            output_struct[1] = setTimestamp()
          end for
          RESTORESTATE(previous_block)
        end if
        # determined datatype...
        input.value = linkedOutput_struct[0]
      end for
      EXECUTE(block)
      OL = getOutputList(block, fmu_IM)
      for output in OL do
        # determined datatype...
        output_struct[0] = output.value
        output_struct[1] = setTimestamp()
      end for
      if block is aAlgebraicLoopType then
        ...
      end if
    end if
  end for
  time = time + time_increment()
end while
```
Current Work: Automatic Generation of Coupling

• Inputs:
  – FMUs
  – Co-simulation model
    • Scenario
    • Capabilities
    • Semantic adaptation

• Output:
  – Optimized coupling
Summary

• M&S in Industry
• Co-simulation as solution for full virtual development of complex systems
  – FMI as a specific co-simulation standard
• Correctness of co-simulation
• Challenges
  – Physical
  – Numerical
  – Computer science related
• Current work
  – Semantic adaptation
  – Generation of optimized coupling
Thank you!
References


