Simulation and Visualization
Tool Design for Robot Software

Zhou Lu, Tjalling Ran and Jan Broenink
Robotics and Mechatronics, University of Twente
August 22, 2016
Outline

- Introduction
- Design of Simulation
- Visualizing Simulation Results
- Results
- Conclusions and Recommendations
Introduction - Context

- Cyber-Physical Systems (CPS) co-design: why challenging?
  - Combine multiple different engineering disciplines/domains
  - Seamless interaction with physical environments
  - Concurrency is intrinsically presented in CPS
  - Most CPS are safety-critical
Design CPS using Model-Driven Design (MDD)
- Models can be formalized and checked
- Modelled systems can be tested and simulated off-line
- Ease tedious and error-prone concurrent software development
Introduction - Problem Statements and Motivation

- Iterative and incremental design and development in MDD
  - Sufficient verification and/or validation of models are required
- MDD in CPS: different domain models are involved
  - Discrete-Event and Continuous-Time domains
- Co-simulation is needed to support co-design
- Current infrastructure: TERRA
  - Does not provide sufficient simulation nor visualization facilities
Design of Simulation - Obtain Executable Models

- Executable models
  - Executability: depends more on execution tools
  - Execution tools: depend on assessment requirements

- Required assessments in CPS
  - Process execution order
  - Results of algorithms
Design of Simulation - Obtain Executable Models

- Executable models
  - Executability: depends more on execution tools
  - Execution tools: depend on assessment requirements
- Required assessments in CPS
  - Process execution order
  - Results of algorithms
- Two strategies to obtain executable models
  - Model interpretation
  - Code generation
Design of Simulation - Analysis

- Model Interpretation
  - Relies on the existence of a Virtual Execution Environment (VEE)
  - Interpretation can be done dynamically
- Code generation
  - Uses M2T transformation to generate lower-level system representation
  - Platform-dependent
Design of Simulation - Analysis

- From a practical perspective: code generation is preferred
  - Control algorithms generated from 20-sim must be taken into account
  - TERRA is able to generate C++ code
  - Model interpretation will just be simulation without code implementation
Design of Simulation - Analysis

- Coupling strategies
  - Loose-Coupling Execution
    - Generate source code from different models
    - Generate APIs for interacting purpose
    - Execution coordinator
  - Tight-Coupling Execution
    - Integrate different models by using M2M transformations
    - Generate code from an integrated model
Design of Simulation - A hybrid simulation approach

- **Tight-Coupling Execution**
  - Integrate 20-sim controller model into TERRA
  - Generate code from the integrated model
Design of Simulation - A hybrid simulation approach

- Loose-Coupling Execution
  - Generate code from C/P models
  - Generate APIs from TERRA FMI interface model
  - FMI wrapper as coordinator
Design of Simulation - A hybrid simulation approach

- Visualizing simulation results
  - Process execution order
  - Results of algorithms
- Iterative and incremental design and development
Visualizing Simulation Results

- Five states for CSP constructs and processes
  - Activate, Activating other processes, Waiting, Running, Done

- Logging facilities were designed to capture state changes
  - Registration phase
  - States recording phase

<table>
<thead>
<tr>
<th>time stamp</th>
<th>State Index 1</th>
<th>State Index 2</th>
<th>State Index 3</th>
<th>State Index n-1</th>
<th>State Index n</th>
</tr>
</thead>
</table>
Visualizing Simulation Results

- Overall structure of the visualization
  - Execute models
  - Logged data will be stored as CSV files
  - Mapping model elements
  - Parsing logged data
  - Publishing states to a graphical view
Results

- Loop control model for testing
Results

- Simulation Comparison

Simulation Comparison
TERRA-LUNA-sim vs 20-sim

![Simulation Comparison Graph]

Minor differences
1.95 2 2.05 2.1 2.15 2.2
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18

Simulation Time (s)

Values

Step-output
Controller-output
20sim-step-output
20sim-controller-output
Plant-y
20sim-plant-y
Results

- One snapshot of logged process states

<table>
<thead>
<tr>
<th>Process ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>time stamp</td>
<td>0.230</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

- Using different colors to represent process states
Conclusions and Recommendations

- **Conclusions**
  - The simulation provides comparable results as the ground truth
  - The animation can sufficiently indicate process execution order
  - Opportunity to implement a rapid prototyping system
  - Opportunity to obtain an executable and deployable binary which can be right-first-time

- **Recommendations**
  - Signal values are not automatically visualized as state changes
  - Options to include or exclude processes/states from animations
  - Timing analysis need to be implemented
  - FMI interfacing and wrapping facilities
Thanks!
Tree structure of the example model