Introduction to CoHLA and relation with INTO-CPS

Thomas Nägele
Radboud University
The Netherlands
thomas@cohla.nl

Jozef Hooman
Radboud University & ESI (TNO)
The Netherlands
hooman@cs.ru.nl

2nd August 2019

Contents

1 Introduction

The INTO-CPS project[1] [6] and CoHLA [7] both provide a co-simulation framework for models adhering to the FMI standard [2]. INTO-CPS provides a tool chain that supports the development of a CPS from its requirements to realisation. From the requirements, a set of SysML [5] specifications is created. These specifications include the interfaces of all the component models, after which these models should be developed. A co-simulation of the models can be created and is executed by a Co-simulation Orchestration Engine (COE) called Maestro[2]. The tool chain also provides support for design space exploration (DSE) [3], and HIL and SIL simulation. All models, meta-models and specifications are collectively presented by the INTO-CPS application. The application allows the construction and execution of a co-simulation as well as showing the results.

Configuring HLA (CoHLA)[3] is a domain specific language (DSL) [4] that allows the user to quickly construct a co-simulation from a set of simulation models. In contrast with INTO-CPS, CoHLA only focuses on the construction of the co-simulation and a number of additions for the co-simulation instead of the full CPS design. Its goal is to minimise the effort that is required to construct a co-simulation, so that existing methodologies could easily adapt CoHLA for co-simulation construction. CoHLA uses the HLA standard [1] for the coordination of the co-simulation. From a CoHLA co-simulation specification,

[1]http://projects.au.dk/into-cps/
[2]https://github.com/INTO-CPS-Association/maestro
[3]https://cohla.nl/

source code is generated that can be used with an implementation of the HLA standard. Currently, only the open source HLA implementation OpenRTI[4] is supported by CoHLA.

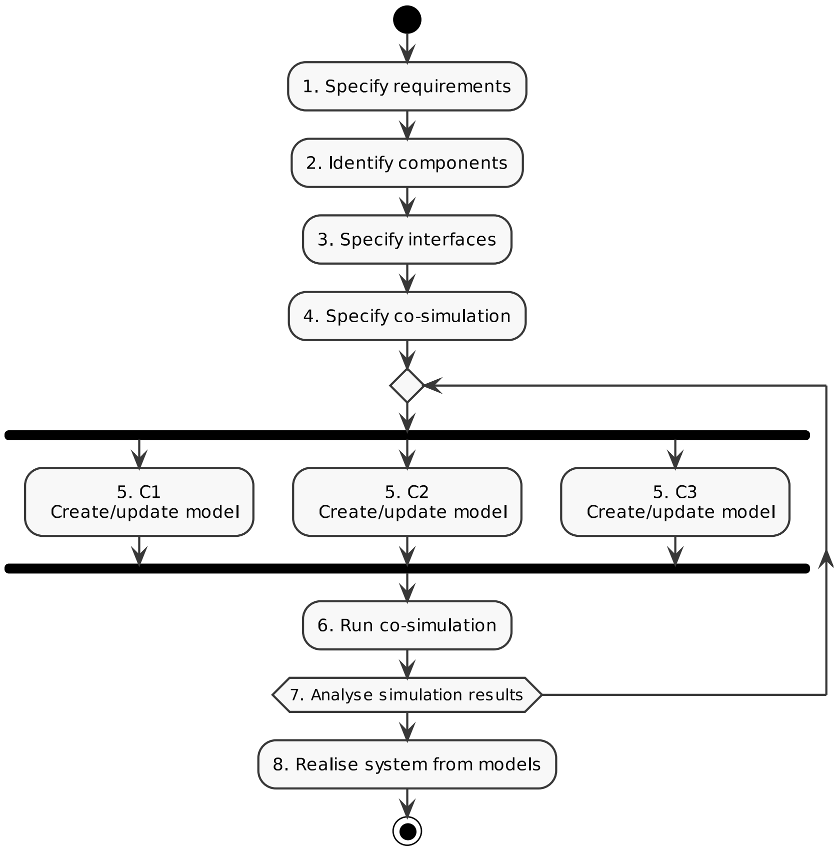
While both co-simulation frameworks support the FMI standard for their simulation models, the frameworks have different approaches of running the co-simulation. The INTO-CPS project has developed its own co-simulation engine, while CoHLA uses an implementation of the HLA standard for this. The focus of CoHLA mainly lies on the rapid construction of a co-simulation, while the INTO-CPS project focuses on traceability throughout the design process of the system. This document describes the CoHLA framework and compares co-simulation results of a co-simulation as generated by CoHLA with the results of the same co-simulation as run by the INTO-CPS framework.

2 CoHLA

This section briefly describes an example design flow CoHLA supports and the language itself.

2.1 Design flow

CoHLA supports a model-based approach for the design of a CPS. An example of a design flow is displayed in Figure 1. CoHLA can be used for steps 4, 6 and 7 to construct and run a co-simulation of the models, after which the results can be analysed. The results are used as input for the next iteration of the model development. During this iterative process, interfaces between the components may change, which also requires adjustments to the co-simulation specification. Since CoHLA allows the rapid changing of the model interfaces, the overhead of applying such changes is limited.



1. Specify requirements
2. Identify components
3. Specify interfaces
4. Specify co-simulation
5. C1 Create/update model
5. C2 Create/update model
5. C3 Create/update model
6. Run co-simulation
7. Analyse simulation results
8. Realise system from models

Figure 1: *Activity diagram of an example design flow of a CPS. C1, C2 and C3 represent three different components of the system. This is just exemplary, as the design flow also applies for systems consisting of more components.*

[4] `https://sourceforge.net/projects/openrti/`

The design flow described above is only a schematic representation of the development process. Not all models always need the same number of iterations and their levels of detail may vary. Therefor, the development process is typically less linear than suggested in the figure.

2.2 Co-simulation specification

A co-simulation from a set of simulation models can be constructed in CoHLA by specifying all models in terms of inputs and outputs. Such a specification is similar to a class definition of the model. Model parameters and simulation properties such as the step size can be specified as well.

The co-simulation specification basically consists of a set of instances of these previously specified classes. Every instance has a name and multiple instances of the same class may participate in the co-simulation. A set of connections between the attributes of the instances specifies how these are connected to each other, i.e., how input and output are related.

A CoHLA co-simulation specification allows the specification of parameter values that are used during initialisation of the co-simulation. The values of specific attributes at given points in time during the simulation can be specified in a scenario and fault scenarios may be used to introduce a number of basic faults into the co-simulation.

Section 2.4 briefly introduces a small example of a co-simulation. Its sources are available online and include examples of a scenario and default parameter valuations.

2.3 Code generation

From a co-simulation specification a number of files are generated. Wrapper code extending the CoHLA libraries is generated for each of the specified federate classes. This wrapper implementation compiles to an executable that connects the simulation model to the HLA RunTime Infrastructure (RTI). Configuration files that contain, for instance, the parameter configurations are also generated. Finally, a Python run script is generated that enables the user to build and run the co-simulation easily. The run script provides three basic methods to the user.

- **Build**: Compiles all federate class wrappers and stores the executables in a folder with the name "build".
- **Run**: Starts the co-simulation execution with the parameters provided to the run script or the default parameters. This method allows for customisation by providing generated configurations to the run script to modify the co-simulation initialisation. By starting a co-simulation, the RTI, all simulators and wrappers are started in separate threads.
- **Display**: Prints the current configuration with respect to the provided arguments of the federation. This method does not start a simulation, allowing the user to verify whether the correct configurations are used.

2.4 Example: RoomThermostat

A co-simulation of a small thermostat system – called the RoomThermostat – was developed to illustrate CoHLA. The system consists of one thermostat and three rooms. Each of the rooms contains a heater and a window. The thermostat reads the temperature in one of the rooms and is responsible for maintaining the temperature around the target temperature by controlling the heater state. The thermostat controls the heater state in all connected rooms, while the state is determined by the temperature in only one of the rooms. The three rooms have different characteristics such as the heater size, window size and surface area of the room.

The model of the room is a continuous-time model created in 20-sim. The Overture tool[5] was used to create a discrete-time model of the thermostat using VDM-RT. Both models are exported to FMUs for use in a co-simulation.

The co-simulation of the system is specified in CoHLA to construct a co-simulation. The class specification of the model of the room is shown in Listing 1. The model of the room has one input, one

[5]`http://overturetool.org/`

output and five parameters specified. The strings after the parameter names represent the exact name within the FMU. A number of default simulation settings is provided, such as the default model to load, time policy of HLA and step size.

```
FederateClass Room {
  Type FMU
  Attributes {
    Input Boolean heaterState
    Output Real temperature
  }
  Parameters {
    Real RadiatorSize "radiator.A"
    Real Surface "Roomcapacity.surface"
    Real Height "Roomcapacity.height"
    Real InitTemp "Roomcapacity.initialtemp"
    Real WindowSize "window.A"
  }
  TimePolicy RegulatedAndConstrained
  DefaultModel "../../models/Room.fmu"
  AdvanceType NextMessageRequest
  DefaultStepSize 5.0
  DefaultLookahead 0.1
}
```

Listing 1: *Class specification of the room model.*

A similar specification of the thermostat model is displayed in Listing 2. This class also has a number of attributes and one parameter. Simulation defaults are also provided.

```
FederateClass Thermostat {
  Type FMU
  Attributes {
    InOutput Real targetTemperature
    Input Real temperature
    Output Boolean heaterState
  }
  Parameters {
    Real targetTemperature "targetTemperature"
  }
  TimePolicy RegulatedAndConstrained
  DefaultModel "../../models/Thermostat.fmu"
  AdvanceType TimeAdvanceRequest
  DefaultStepSize 30.0
  DefaultLookahead 0.1
}
```

Listing 2: *Class specification of the thermostat model.*

To be able to generate a log file from a co-simulation execution, CoHLA also supports the addition of a logger class. Listing 3 shows the class specification for this class. Note that the default measure time of the log can be changed upon starting the co-simulation: the current value only provides a default.

```
FederateClass Logger {
  Type CSV-logger {
    DefaultMeasureTime 3600.0
  }
}
```

Listing 3: *Class specification of the logger.*

From these simulation classes a co-simulation can be specified, which is displayed in Listing 4. The co-simulation consists of three instances of the room simulation, named *Livingroom*, *Kitchen* and *Hall*. An instance of the thermostat and a logger are also included. The connections specify how the different attributes are connected to each other. The logger instance receives multiple attributes as input, as these represent the attributes that should be included in the resulting log file.

```
Federation House {
  Instances {
    Thermostat : Thermostat
    Livingroom : Room
    Kitchen : Room
    Hall : Room
    log : Logger
  }
  Connections {
    { Thermostat.temperature <- Livingroom.temperature }
    { Livingroom.heaterState <- Thermostat.heaterState }
    { Kitchen.heaterState <- Thermostat.heaterState }
    { Hall.heaterState <- Thermostat.heaterState }
    { log <- Thermostat.targetTemperature, Livingroom.temperature, Kitchen.
        temperature, Hall.temperature }
  }
}
```

Listing 4: *Co-simulation specification for the RoomThermostat system.*

From these specifications, the sources for the co-simulation are generated. Running this co-simulation without any further configuration, however, results in three identical simulations of the rooms. Parameter configurations can be specified to be easily applied on the co-simulation. Listing 5 shows four parameter configurations for the different simulation models.

```
Configuration Large for Room {
  Surface = "45.0"
  WindowSize = "11"
  RadiatorSize = "1.0"
}

Configuration Medium for Room {
  Surface = "10.0"
  WindowSize = "2.0"
  RadiatorSize = "0.25"
}

Configuration Small for Room {
  Surface = "4.5"
  WindowSize = "2.5"
  RadiatorSize = "0.1"
}

Configuration Comfortable for Thermostat {
  targetTemperature = "20.5"
}
```

Listing 5: *Parameter configurations for the models.*

For a specific co-simulation, these parameter configurations can be bundled to be applied on the co-simulation. Such a bundle of parameter configurations is called a *Situation*, of which an example is shown in Listing 6. Here, the three different room configurations are applied to the three instances in the co-simulation. The thermostat, for example, is provided with a default target temperature, as specified in Listing 5.

```
Situation ComfyBase {
  Apply Comfortable to Thermostat
  Apply Large to Livingroom
  Apply Medium to Kitchen
  Apply Small to Hall
}
```

Listing 6: *Situation for the RoomThermostat system.*

The source code that is generated can be compiled and executed with the generated run script. More information about the CoHLA grammar and on how to use the run script can be found in the CoHLA User Manual[6]. The described co-simulation can be run with the following command.

[6]https://cohla.nl/docs/

```
./run.py -t conf/house.topo -s conf/House/ComfyBase.situation -be
```

Listing 7: *Build (-b) and run (-e) command for the RoomThermostat system.*

The resulting log file of this command is plotted in Figure 2. It can be seen that the Livingroom is the only simulation that provides input for the thermostat, as it is the only room that is maintained around the set target temperature.

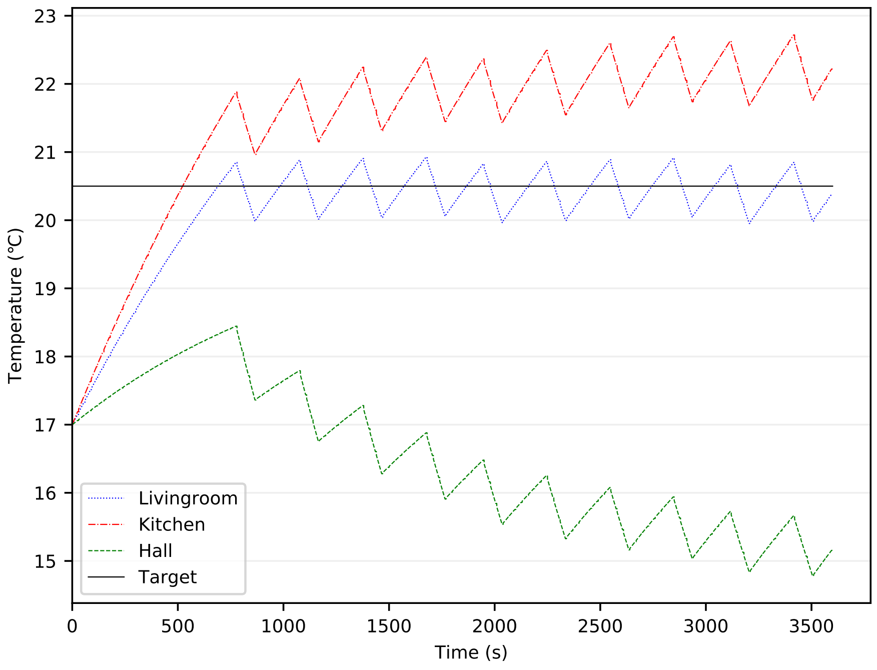


Figure 2: *Resulting log file for the RoomThermostat co-simulation.*

3 Comparison of CoHLA and INTO-CPS

The results of identical co-simulations in CoHLA and INTO-CPS have been compared with each other. Our goal was to find out whether the use of HLA as running algorithm and the CoHLA framework affect the co-simulation results in a way that is different from the INTO-CPS approach. For this, a relatively easy to comprehend case study was selected from the set of open source samples provided by the INTO-CPS Association: the Single-tank Water Tank example, henceforth called SingleWatertank.

The SingleWatertank[7] system consists of two simulation models. There is a water tank that fills with water at a constant rate, which increases the water level. The water tank has a valve that can be opened or closed to drain the water to avoid the tank from being too full. A valve controller controls the state of the valve to maintain the water level between specified limits.

3.1 Co-simulation

Constructing a co-simulation of the two components is rather straightforward in both co-simulation frameworks. The valve state output of the controller is connected to the valve state input of the water tank and the water level output of the water tank is connected to the water level input of the controller. Initially, the water tank is empty and the controller is configured to maintain the water level between 1 and 2. Both models are simulated with a step size of 0.1 seconds from time 0 to 30 seconds. The full specification of the SingleWatertank system in CoHLA is included in Appendix A.

[7] `https://github.com/INTO-CPS-Association/example-single_watertank`

3.2 Results

Figure 3 shows the results as simulated by both frameworks. The results are very similar, showing that the controller fails to maintain the water level within the specified range.

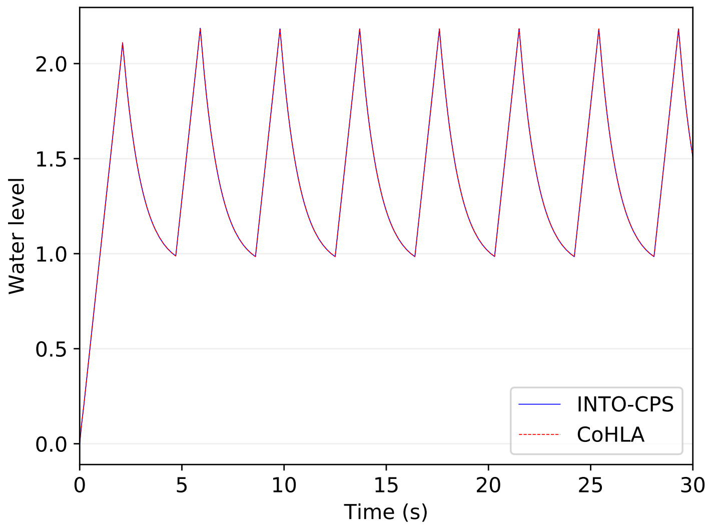


Figure 3: *Results of the SingleWatertank system for the INTO-CPS and CoHLA co-simulations.*

Figure 4a shows the absolute difference in water level between the co-simulation according to their logs. This figure shows two spikes for which there is a relatively large difference in the water levels of the co-simulations. The first spike (at time 0) can be explained by the fact that the implementation of CoHLA triggers one initial computation step in the FMU, which then reports a water level that has been changed slightly since time point 0. Additional experiments show that the second spike appears to be caused by the FMU that simulates a little more than expected. The FMU as executed by CoHLA simulated until time 17.81 instead of the requested 17.8. Since the valve is open, the water level lowers at roughly 0.01 unit per second, which corresponds to the measured difference. Although the FMUs are exactly the same, the target time that is provided to the FMU by the co-simulation framework is might be slightly different because of floating point inaccuracy.

Figure 4b shows the water level differences between time 1 and 15. Note that the step size for the vertical axis has changed from $5 \cdot 10^{-2}$ to $2 \cdot 10^{-7}$. The figure shows that the differences vary, but remain small enough to be caused by different encodings of floating point values or rounding differences. From this experiment we conclude that both co-simulation frameworks yield nearly identical results.

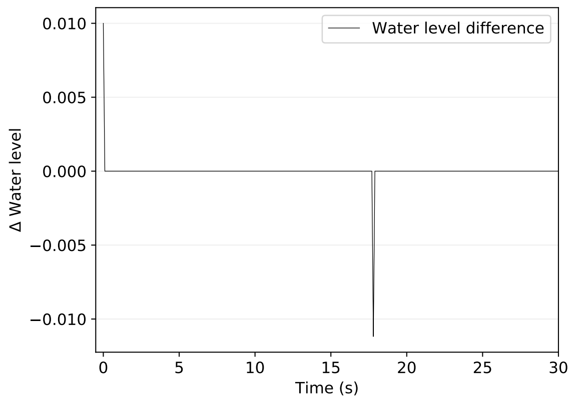
4 Conclusion

This document has briefly introduced CoHLA in the context of the INTO-CPS project. An example of a thermostat system as specified in CoHLA is given. The water tank system is used to compare the co-simulation results of the two co-simulation frameworks with each other, which shows that the results are very similar.

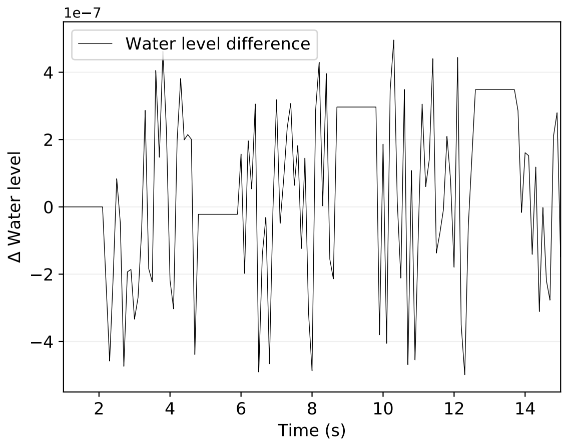
4.1 Resources

The CoHLA sources are available open source on Github[8]. This repository also included an installation manual and user manual to get started with CoHLA. A number of sample projects for CoHLA are available online as well[9]. The experiments discussed in this document are included in the latter repository in the directory “Chapter 5. Confidence in Co-simulation Results”.

[8]`https://github.com/phpnerd/CoHLA`
[9]`https://github.com/phpnerd/CoHLA-projects`



(a) *Full simulation of 30 seconds.*



(b) *Simulation from 1 to 15 seconds.*

Figure 4: *Water level differences of the Watertank between the INTO-CPS and CoHLA co-simulations.*

4.2 List of publications

The following peer reviewed papers related to CoHLA have been published.

P.1 T. Nägele and J. Hooman. Co-simulation of Cyber-Physical Systems using HLA. In *Proceedings 7th IEEE Annual Computing and Communication Workshop and Conference (CCWC)*, pages 267–272, 2017.

P.2 T. Nägele and J. Hooman. Rapid Construction of Co-simulations of Cyber-Physical Systems in HLA using a DSL. In *Proceedings 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 247–251, 2017.

P.3 T. Nägele, J. Hooman, T. Broenink and J. Broenink. CoHLA: Design Space Exploration and Co-simulation Made Easy. In *Proceedings IEEE Industrial Cyber Physical Systems (ICPS)*, pages 225–231, 2018.

P.4 T. Nägele, J. Hooman and J. Sleuters. Building Distributed Co-simulations using CoHLA. In *Proceedings 21st Euromicro Conference on Digital System Design (DSD)*, pages 342–346, 2018.

P.5 T. Nägele and J. Hooman. Scalability Analysis of Cloud-based Distributed Simulations of IoT Systems using HLA. In *Proceedings 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 1075–1080, 2018.

P.6 T. Nägele, T. Broenink, J. Hooman and J. Broenink. Early Analysis of Cyber-Physical Systems using Co-simulation and Multi-level Modelling. In *Proceedings IEEE Industrial Cyber Physical Systems (ICPS)*, pages 133–138, 2019.

References

[1] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules. *IEEE Std 1516-2010*, pages 1–38, Aug 2010.

[2] T. Blochwitz, M. Otter, et al. The Functional Mockup Interface for tool independent exchange of simulation models. In *8th Modelica Conference*, pages 105–114, 2011.

[3] John Fitzgerald, Carl Gamble, Richard Payne, and Benjamin Lam. Exploring the Cyber-Physical Design Space. In *INCOSE International Symposium*, volume 27, pages 371–385. Wiley Online Library, 2017.

[4] Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

[5] Matthew Hause et al. The SysML modelling language. In *Fifteenth European Systems Engineering Conference*, volume 9, pages 1–12. Citeseer, 2006.

[6] P. G. Larsen, J. Fitzgerald, et al. Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project. In *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, pages 1–6, April 2016.

[7] T. Nägele, J. Hooman, T. Broenink, and J. Broenink. CoHLA: Design Space Exploration and Co-simulation Made Easy. In *IEEE 1st Industrial Cyber-Physical Systems (ICPS 2018)*, pages 225–331, May 2018.

A SingleWatertank

```
Environment {
  RTI {
    OpenRTI
    Libraries "/opt/OpenRTI-libs"
  }
  PublishOnlyChanges
}

FederateClass SingleWatertank {
  Type FMU
  Attributes {
    Input Real valvecontrol
    Output Real level
  }
  DefaultModel "../../models/singlewatertank-20-sim.fmu"
  AdvanceType TimeAdvanceRequest
  DefaultStepSize 0.1
  DefaultLookahead 0.000001
}

FederateClass WatertankController {
    Type FMU
    Attributes {
      Input Real level
      Output Boolean valve
    }
    Parameters {
      Real Maxlevel "maxlevel"
      Real Minlevel "minlevel"
    }
    DefaultModel "../../models/SingleWT-controller.fmu"
    AdvanceType TimeAdvanceRequest
    DefaultStepSize 0.1
    DefaultLookahead 0.000001
}

FederateClass Logger {
  Type CSV-logger {
    DefaultMeasureTime 30.0
  }
}

Configuration defaultWC for WatertankController {
  Maxlevel = "2.0"
  Minlevel = "1.0"
}

Federation SingleWatertankSystem {
  Instances {
    wt: SingleWatertank
    controller : WatertankController
    log : Logger
  }
  Connections {
    { controller.level <- wt.level }
    { wt.valvecontrol <- controller.valve }
    { log <- wt.level, controller.valve }
  }

  Situation default {
    Apply defaultWC to controller
  }
}
```

Listing 8: *CoHLA specification of the SingleWatertank system.*