User Manual for HLA-PTII federates

For more information about the HLA-PTII co-simulation framework read [1, 2, 3].

HLA-PTII demos are in `$PTII/org/hlacerti/demo`. Prior to revision r71890 demos were in `$PTII/ptolemy/apps/hl`

The actors can be found in the library under
`MoreLibrairies->Co-Simulation->HLA`: HlaManager, HlaPublisher and HlaSubscriber.
You need to install the HLA compliant RTI called CERTI (tested with versions 3.4.2, 3.4.3 and 3.5) `http://www.nongnu.org/certi/certi_doc/Install/html/index.html`.

Put in your .bash_profile un export for $PTII and $CERTI_HOME. Or at least put aliases:

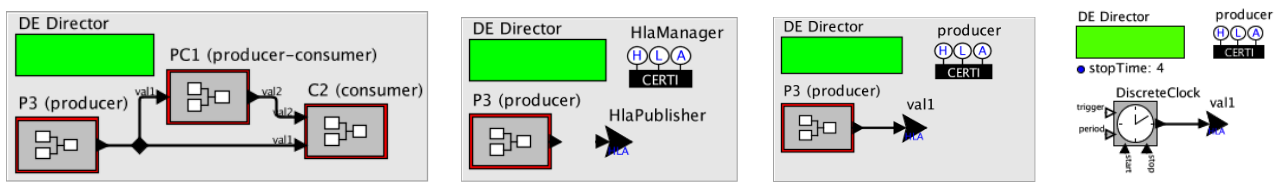
```
alias myPtII=export PTII=your-path-to-ptII''
alias cfgCerti=''source your-CERTI_HOME/share/scripts/myCERTI_env.sh.''
```

Do not forget to execute these aliases in each terminal you open.

1 Building a model with HLA-PTII co-simulation framework

Let us suppose you have already a (centralized) Ptolemy model as the one of figure 1.a. You want to simulate this model in a distributed way using 3 simulators, one for each composite actor.

Important remark: The top-level model must use a DE director, but you can have a Continuous director in a composite actor inside.



a) Centralized model b) Hla actors c) Naming the actors d) flat federate

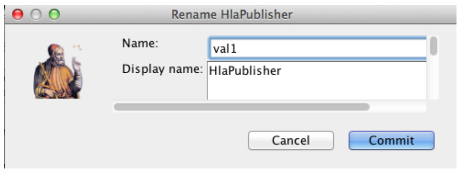
Figure 1: Building Ptolemy federates

Consider that you want to create the `producer` federate that only sends data. The steps are the following:

1. Create a FOM file with the classes and attributes used in the simulation as in figure 2.c
2. Duplicate the model of figure 1.a, removing all the blocs except the composite actor "P3 (producer)".
3. Drag the actors HlaManager and HlaPublisher from `MoreLibrairies->Co-Simulation->HLA` (see figure 1.b).
4. Configuring HlaPublisher. Connect the output of the composite actor to the HlaPublisher actor, then: a) right-click on HlaPublisher, Configure/Rename and put `val1` (the name of the class attribute in the fed file as in figure 2.a) in the field `name` ; the name displayed will be the same (figure 1.c); b) double-click HlaPublisher (the window of figure 2.b pops out) and put `MyObjectClass` (the name of the class according to your fed file in figure 2.c) in the parameter `Object class in FOM`.
5. Configuring HlaManager. Double-click this actor; the window in figure 2.d appears. Put the name of the federation and the federate as in the fed file (figure 2.c) and browse the fed file. You can change the lookahead value. If you need a synchronization point, tick the field "Require synchronizationPoint?" and choose a same name for all federates of your distributed simulation. Remark: only the last federate to be launched must have the field "Is synchronization point creator?" ticked.

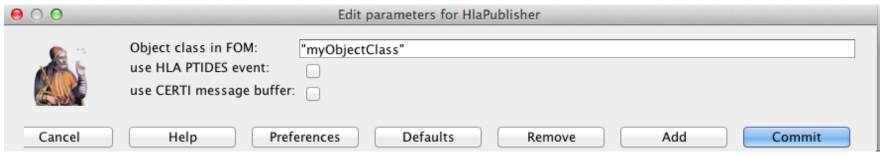
Consider that you want to create the `consumer` federate that only receives data. The steps are the following:

1. Duplicate the model of figure 1.a, removing all the blocs except the composite actor "C2 (consumer)".



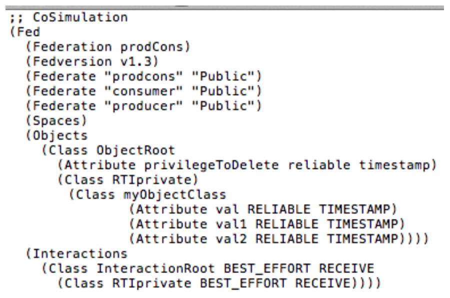
Name: val1
Display name: HlaPublisher

a) Renaming with attribute



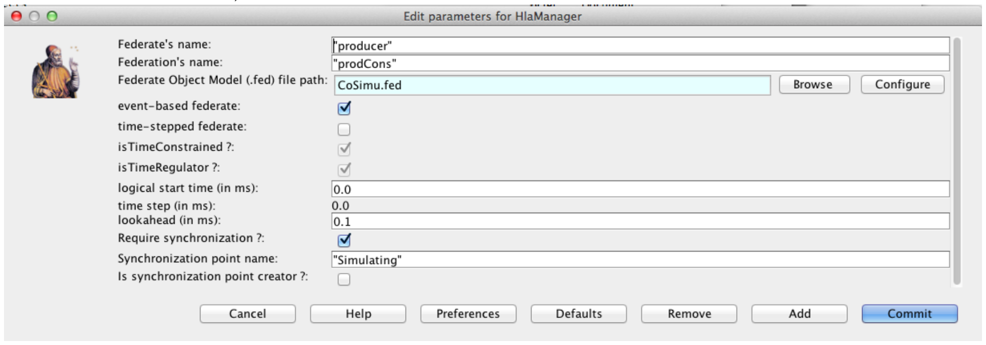
Object class in FOM: "myObjectClass"
use HLA PTIDES event: ☐
use CERTI message buffer: ☐

b) Naming the object class in HlaPublisher



```
;; CoSimulation
(Fed
  (Federation prodCons)
  (Fedversion v1.3)
  (Federate "prodcons" "Public")
  (Federate "consumer" "Public")
  (Federate "producer" "Public")
  (Spaces)
  (Objects
    (Class ObjectRoot
      (Attribute privilegeToDelete reliable timestamp)
      (Class RTIprivate)
       (Class myObjectClass
                (Attribute val RELIABLE TIMESTAMP)
                (Attribute val1 RELIABLE TIMESTAMP)
                (Attribute val2 RELIABLE TIMESTAMP))))
  (Interactions
    (Class InteractionRoot BEST_EFFORT RECEIVE
      (Class RTIprivate BEST_EFFORT RECEIVE))))
```

c) CoSimu.fed file (FOM)



Federate's name: "producer"
Federation's name: "prodCons"
Federate Object Model (.fed) file path: CoSimu.fed
event-based federate: ☑
time-stepped federate: ☐
isTimeConstrained ?: ✓
isTimeRegulator ?: ✓
logical start time (in ms): 0.0
time step (in ms): 0.0
lookahead (in ms): 0.1
Require synchronization ?: ☑
Synchronization point name: "Simulating"
Is synchronization point creator ?: ☐

d) Configuring HlaManager

Figure 2: FOM

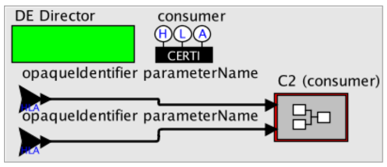
2. Drag one HlaManager and two HlaSubscriber actors from `MoreLibrairies->Co-Simulation->HLA`. Configure the HlaManager as in step 5 above. Connect the inputs of the composite actor `C2` to the HlaSubscriber actors (see figure 3.a).

3. Configuring HlaSubscriber: double-click on the first HlaSubscriber (the window of figure 3.b pops out)

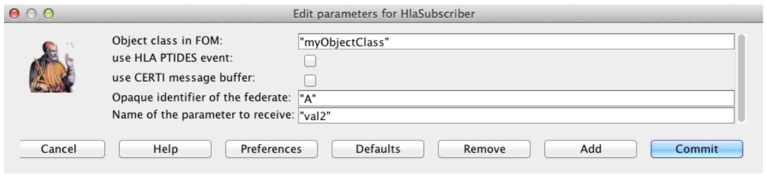
   - put "myObjectClass" in `Object class in FOM` (the name of the class in the fed file as in figure 2.c).
   - choose a name (e.g., "A") as `Opaque identifier of the federate`.
   - put `val2` in `Name of the parameter to receive` (the name of the attribute as in the fed file).

   The name displayed will be the concatenation of the opaque identifier and the attribute, `A val2` (figure 3.c).

   Now configure the second HlaSubscriber in the same way, but put `val1` in `Name of the parameter to receive`.

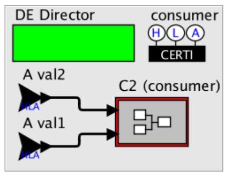

DE Director
consumer
H L A
CERTI
opaqueIdentifier parameterName
opaqueIdentifier parameterName
C2 (consumer)

a) HlaSubscriber



Object class in FOM: "myObjectClass"
use HLA PTIDES event: ☐
use CERTI message buffer: ☐
Opaque identifier of the federate: "A"
Name of the parameter to receive: "val2"

b) Configuration window of HlaSubscriber

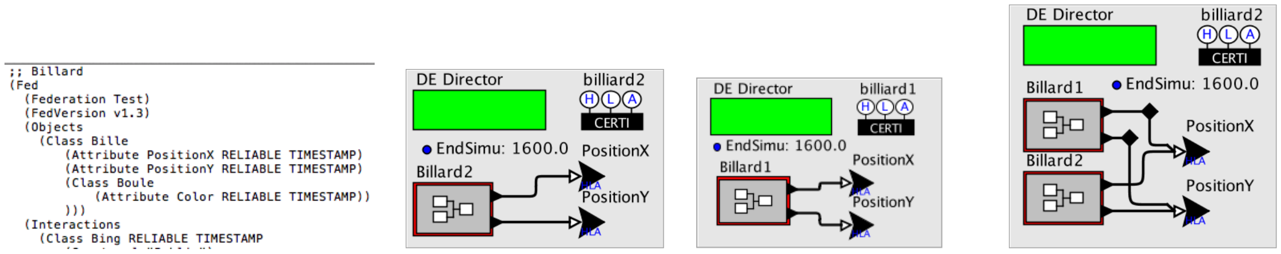


c) Naming the actors

Figure 3: Building Ptolemy federates

Now you can create the federate `prodcons`. The steps are similar.

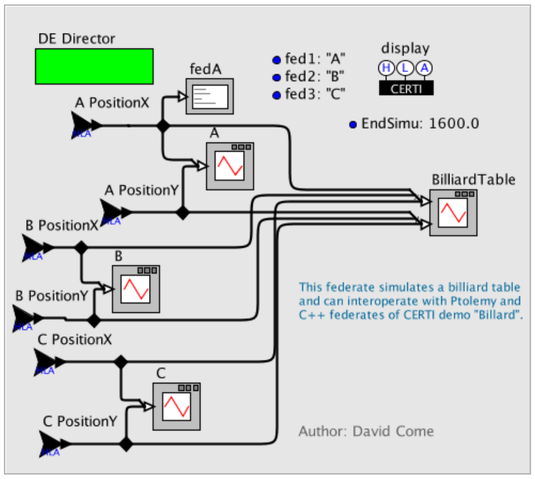
- Do step 1 (duplicate the centralized model removing all composite actors except the one you want to simulate).
- Add a HlaPublisher for `val2` and a HlaSubscriber for `val1`. Add also a HlaManager (step 5). Pay attention to name the federate as `prodcons` in the configuration window of HlaManager (figure 2.d).
- Check if all your federates (but one) have the field "Is synchronization point creator?" unticked. Only one of them must have this field ticked and it must be the last one to be launched.



```
;; Billard
(Fed
  (Federation Test)
  (FedVersion v1.3)
  (Objects
    (Class Bille
       (Attribute PositionX RELIABLE TIMESTAMP)
       (Attribute PositionY RELIABLE TIMESTAMP)
       (Class Boule
          (Attribute Color RELIABLE TIMESTAMP))
       )))
  (Interactions
    (Class Bing RELIABLE TIMESTAMP
```

a) Test.fed (FOM) b) c) d) Two instances of `Bille`

Figure 4: Two different Ptolemy federates sending attributes `positionX, positionY` of class `Bille`



This federate simulates a billiard table and can interoperate with Ptolemy and C++ federates of CERTI demo "Billard".

Figure 5: HlaSubscriber actor with registering multiple instances

Remark: if your composite actor has a DE director, you can use directly the composite actor as in figure 1.c or you can have a *flat* model as in figure 1.d. But in this case, you cannot have multiple instances of an attribute. If the composite actor has a Continuous model then you need to use the composite actor.

2 Running the federation

1. Open a terminal and run `cfgCerti` as said in the beginning of this manual or execute the `$CERTI_HOME/share/scripts/myCERTI_env.sh` shell

2. Go to the folder where the 3 federate models are (or give the absolute address) and open the models: `$PTII/bin/vergil consumer.xml producer.xml prod-cons.xml &`

3. Check there is no `rtig` process running (the first model to be run will automatically launch this process). If there is a `rtig` running, kill the process

4. Check there is only one model that has the field "Is synchronization point creator?" ticked. Run the other models in any order but the last to be run is the one that has the cited field ticked.

3 Registering multiple instances of an attribute

Let us consider a general case where several instances of object are published, as in the example of a billiard game, a demo of CERTI. For example, let us consider that we want to represent in a billiard table (federate `display` in figure 5) two instances of class `Bille` with attributes `positionX` and `positionY` as indicated in the FOM on figure 4.a. There are two ways of modeling:

- create two federates, each one registering one instance of class `Bille` as in figure 4.b and c, or
- create one federate that register two instances of class `Bille` each one with 2 attributs (`positionX` and `positionY`) as in figure 4.d.

In both cases, the federate depicted in figure 5 is used for displaying the movement of the ball (`positionX, positionY` of `Bille`).

In the previous example, as there was only one instance of each attribute, a same name "A" could be used as `Opaque identifier of the federate`.

In this case, the different instances must be differentiate. You can create 3 parameters fed1, fed2 and fed3 as in figure 5. Then, put `fed1` in the field `Opaque identifier of the federate` of the first HlaSubscriber and so on.

The way of running the federation is similar to the one described in section 2.

References

[1] Lasnier, G., Cardoso, J., Siron, P., Pagetti, C. and Derler, P.. *Distributed Simulation of Heterogeneous and Real-time Systems*, 17th IEEE/ACM Inter. Symposium on Distributed Simulation and Real Time Applications - DSRT 2013, 30 Oct. 2013 - 01 Nov. 2013 (Delft, Netherlands). *Best paper award.*

[2] Lasnier, G., Cardoso, J., Siron, P. and Pagetti, C. *Environnement de cooperation de simulation pour la conception de systemes cyber-physiques*, Journal europeen des systemes automatises. Vol. 47 n. 1-2-3, 2013.

[3] Come, D. *Improving Ptolemy-HLA co-simulation by allowing multiple instances.* Report ISAE-SUPAERO, March 2014.