User Manual for HLA-PTII federates

This manual must be used for models created after revision 72233 where instances can be dynamically discovered.

For more information about the HLA-PTII co-simulation framework read [2, 3, 4].

HLA-PTII demos are in `$PTII/org/hlacerti/demo`. Prior to revision r71890 demos were in `$PTII/ptolemy/apps/hla`
This manual must be used for models created after revision 72233 where instances can be dynamically discovered.

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:

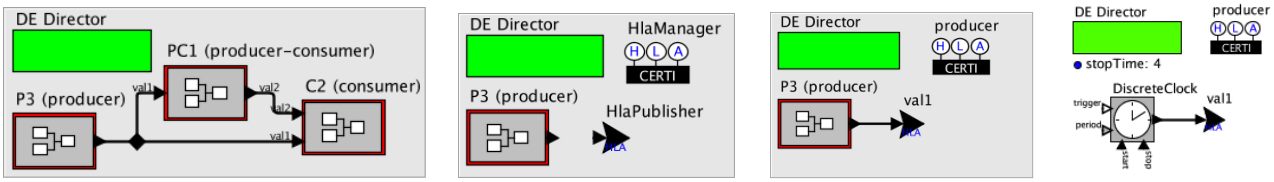
```
alias myPtII=export PTII=your-path-to-ptII''
alias cfgCerti=''source your-CERTI_HOME/share/scripts/myCERTI_env.sh.''
```

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

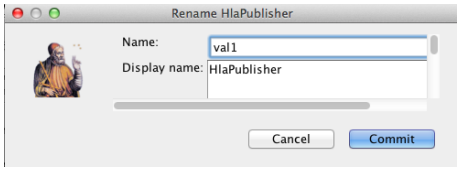
Figure 1: Building Ptolemy federates (producer)

1.1 Configuring a federate that produces data

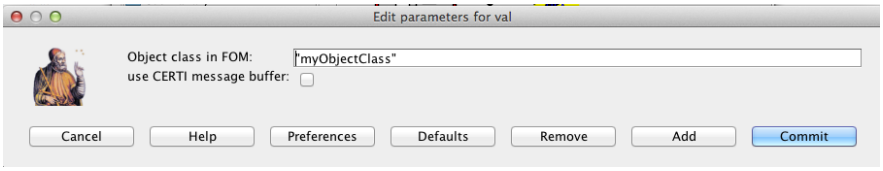
Consider that you want to create the `producer` federate that only sends data. The steps are the following:

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`. The name of the published class is "Federate name + Actor name", where actor is the composite actor connected to HlaPublisher. This name must be unique in the federation.
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.



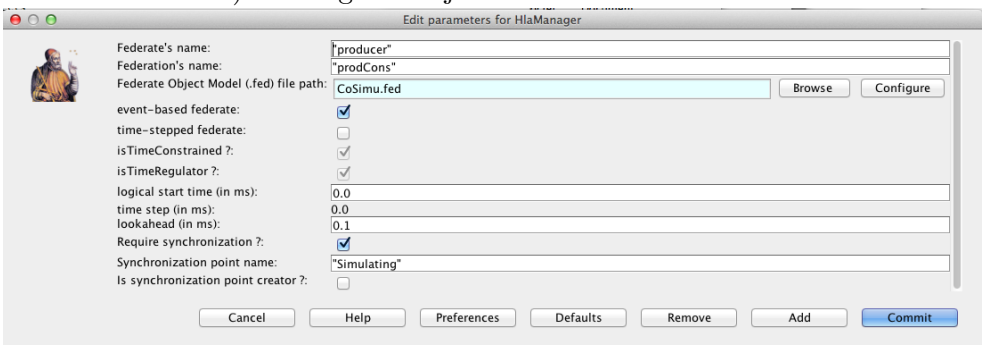
a) Renaming with attribute



b) Naming the object class in HlaPublisher

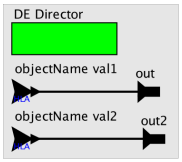
```
;; CoSimulation
(Fed
  (Federation prodCons)
  (Fedversion v1.3)
  (Federate "prodcons" "Public")
  (Federate "consumer" "Public")
  (Federate "producer" "Public")
  (Spaces)
  (Objects
    (Class ObjectRoot
      (Attribute privilegeToDelete reliable timestamp)
      (Class RTIprivate)
       (Class 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)

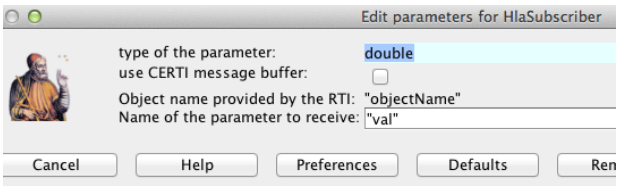


d) Configuring HlaManager

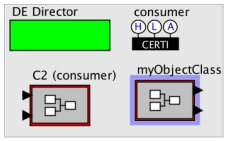
Figure 2: FOM



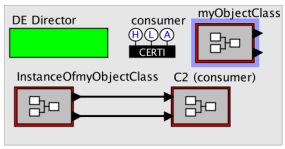
a) HlaSubscriber



b) Configuration window of HlaSubscriber



c) (Ptolemy) class



d) Class instance

Figure 3: Building Ptolemy federates (consumer)

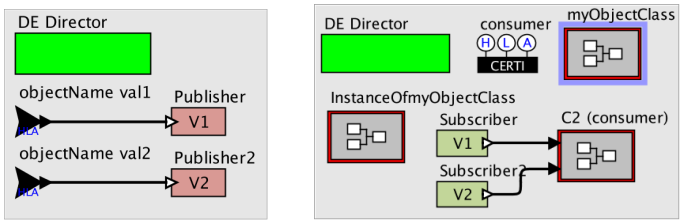
1.2 Configuring a federate that consumes data

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)".
2. Drag a HlaManager from `MoreLibrairies->Co-Simulation->HLA` and configure it as in step 5 above.
3. Create a (Ptolemy) class of object as TypedCompositeActor (see [1] section 2.6) in the federate model:
   - Drag a `CompositeClassDefinition` from `Utilities`; right-click for edit the actor: populate it with a DE director and two HlaSubscriber actors (`MoreLibrairies->Co-Simulation->HLA`) as figure 3.a; the name of this actor (class) must be the same name of the (HLA) class in the FOM (MyObjectClass in figure 2.c). The result is depicted in figure 3.c.
   - Configure the HlaSubscriber: double-click on the first HlaSubscriber (the window of figure 3.b pops out) put `val1` in `Name of the parameter to receive` (the name of the attribute as in the fed file). Do the same for the other HlaSubscriber actor and name it `val2`. The name displayed after running the model will be the concatenation of the federate name (i.e. name of the HlaManager actor) and the attribute, e.g., `PC1 (producer - consumer) val2` (figure 3.c).

- Creating an instance[1] of the (Ptolemy) class `myObjectClass`: right-click on the actor `myObjectClass` (in figure 3.c), `Class Actions -> Create Instance` (or command-I). The actor `InstanceofmyObjectClass` appears; connect its outputs to the input ports of actor `C2` as depicted in figure 3.d (following the good attributes).

In the case you do not need to work with several instances of class, there is another way to design a federate, represented in figure 4. Instead of creating output ports in the (Ptolemy) class as in figure 3.a and 3.c, you can connect the HlaSubscribe actors to Publisher actors (tick parameter `globel` and name the chanel) as depicted in figure 4.a. In the federate model, use Subscriber actor named as in the InstanceOfmyObjectClass actor and connect to the actor. You need at least create one instance of the (Ptolemy) class.



a) Inside Ptolemy class b) Ptolemy model

Figure 4: Building Ptolemy federates (consumer) with Publisher and Subscriber actors

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 HlaManager (step 5) (section 1.1). Pay attention to name the federate as `prodcons` in the configuration window of HlaManager (figure 2.d).
- Add a HlaPublisher for `val2`
- Copy the class `myObjectClass` (from federate `cons`) and instantiate this class; connect only `val1` output to `PC1` actor.
- 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.

Remark: After revision 71935 *flat* models as in figure 1.d cannot be used anymore. For being allowed to have multiple instances of a class a composite actor as in figure 1.c must be used.

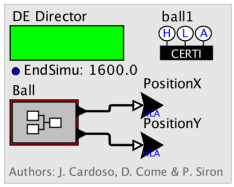
2 Running the federation

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.

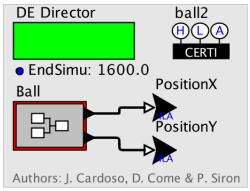
[1] At least one instance of this class must be instantiated in the model and connected to some actor. Each new discovered object will be dynamically instantiated and connected to the same actors that the preexistent instance was connected to (fixme: test case where $A1 \rightarrow B$, $A2 \rightarrow C$ and a new instance $A3$ is discovered).

```
;; 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
```

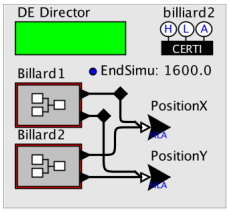
a) Test.fed (FOM)



b) `ball1` federate



c) `ball2` federate



d) Two instances of `Bille`

Figure 5: Two different Ptolemy federates sending attributes `positionX, positionY` of class `Bille`

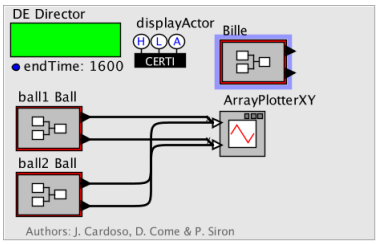


Figure 6: HlaSubscriber actor registering multiple instances

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 6) two instances of class `Bille` with attributes `positionX` and `positionY` as indicated in the FOM on figure 5.a. There are two ways of modeling:

- create two federates, each one registering one instance of class `Bille` as in figure 5.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 5.d.

In both cases, the federate depicted in figure 6 is used for displaying the movement of the ball (`positionX, positionY` of `Bille`).

The different instances are differentiated. If a new `bille` federate joins the federation, a new instance of (Ptolemy) class `Bille` will be dynamically instantiated (discovered) and connected to the actor `ArrayPlotterXY` in the same way the other instances (`ball1 Ball` and `ball2 Ball`) are connected. Notice that after running the composed name "federate + actor" is displayed.

The way of running the federation is similar to the one described in section 2. This federation can run also C++ federates from CERTI demo. In this case:

- in a terminal, launch the rtig
- launch C++ federate: in a terminal, type: billard -n1 -fTest -FTest.fed -t10
- launch PtII federates `ball1` and `Display` in any order
- come back to the terminal, and press "Enter". The simulation starts.

References

[1] C. Brooks, E. A. Lee, S. Neuendorffer, and J. Reekie. *Building Graphical Models* in System Design, Modeling, and Simulation using Ptolemy II, Editor Claudius Ptolemaeus, 2014.

[2] 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.*

[3] 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.

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