User Manual for HLA-PTII federates

(Updated after revision r73687, October, 20, 2015.)

1 Introduction

The HLA-PTII co-simulation framework leverages two open source tools: Ptolemy II and HLA/CERTI. It allows to distribute the execution of a Ptolemy model by using the HLA standard (implemented by CERTI [7]), and is a easy way to produce a HLA federate in a Federation using CERTI. For more information about the HLA-PTII co-simulation framework read [2, 3, 4, 5, 6]. HLA-PTII demos are in `$PTII/org/hlacerti/demo`.

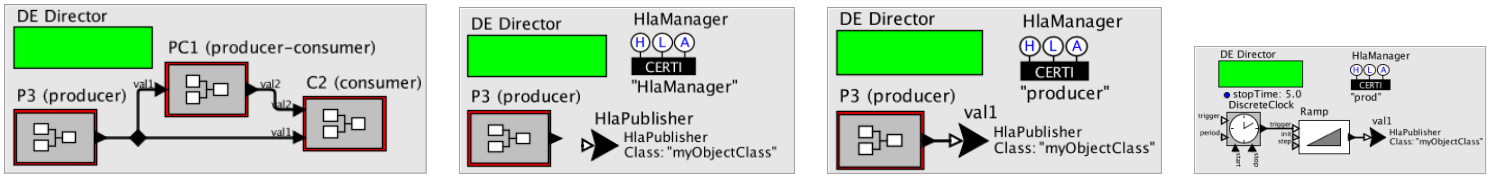
The HLA-PTII framework (called `hlacerti` in Ptolemy tree) is an on-going work. Some important changes were made at revision 71890 (for allowing multiple instances of an object), 72233 (where instances can be dynamically discovered) and 72943 (TAR mechanism for time management was added [6]). Revisions 71890 and 72233 broke backward compatibility, but are an important improvement of the framework. The most important changes in this framework can be found in section 5.

The actors can be found in the library under `MoreLibrairies->Co-Simulation->HLA`: HlaManager, HlaPublisher and HlaSubscriber (see section 6 for installing Ptolemy). You need to install the HLA compliant RTI called CERTI (tested with versions 3.4.2, 3.4.3, 3.5.0 and 3.5.1). See section 7 for installing CERTI and section 3 for running a Federation. A check list for creating federates using `hlacerti` is presented in Appendix A.

2 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)

2.1 Configuring a federate with `HlaPublisher`

In `hlacerti` framework, the `HlaPublisher` actor provides three information to the `HlaManager`: Publish an object class (HLA service `publishObjectClass()`), Register an object (HLA service `registerObjectInstance()`) and update values of a class attribute (HLA service `updateAttributeValues() or UAV`).

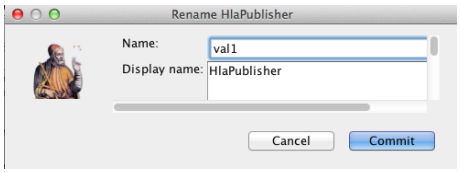
The input port of the `HlaPublisher` is a multiport, and each connection is treated as a separate channel [1].

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

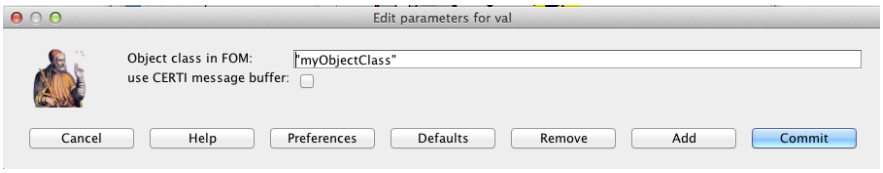
1. Create a .fed file representing the FOM with the classes and attributes used in the simulation as in figure 2.c.
2. Duplicate the model of figure 1.a, removing all the blocs except the composite actor "P3 (producer)".
3. Drag the actors HlaManager and HlaPublisher from `MoreLibrairies->Co-Simulation->HLA` (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` [1].

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. If you need a synchronization point, tick the field "Require synchronizationPoint?" and choose a same name ("Simulating" in figure 2.d) for all federates of your distributed simulation. Only the last federate to be launched must have the field "Is synchronization point register?" ticked. You can choose NER or TAR [6] time management, and change the values of the lookahead, the HLA time unit and the time step (only if TAR is used).



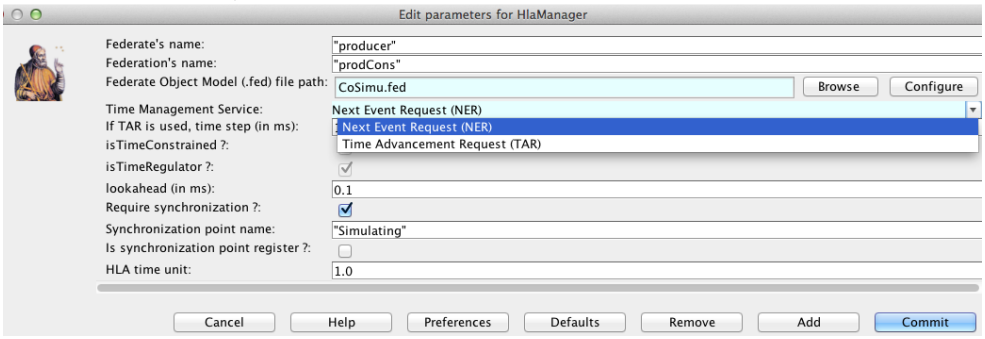
a) Renaming with attribute `val`



b) Naming the object class in HlaPublisher

```
;; CoSimulation
(Fed
  (Federation prodCons)
  (Fedversion v1.0)
  (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

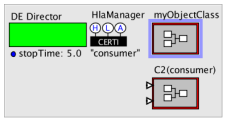
Figure 2: Configuring HLAPublisher and HlaManager.

2.2 Configuring a federate with `HlaSubscriber` [5]

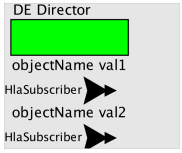
In `hlacerti` framework, the `HlaSubscriber` actor provides three information to the `HlaManager`: Subscribe an object attribute (HLA service `subscribeObjectClassAttribute()`), Discover an object (HLA service `discoverObjectInstance()`) and receive (updated) values of a class attribute (HLA service `reflectAttributeValues() or RAV`).

2.2.1 The federate will discover multiple instances

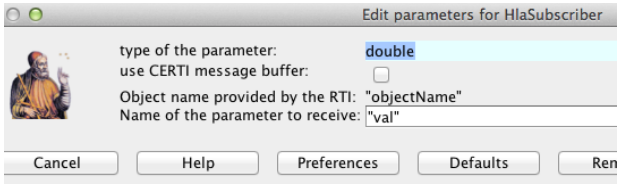
Consider that you want to create the `consumer` federate that only subscribes to some attributes and can *discover* several instances of objects, e.g, a `Display` federate that receives the location of three quadrotors.



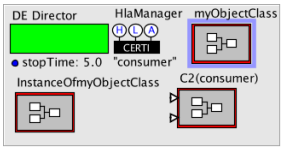
a) (Ptolemy) class



b) Inside the class



c) Configuration window of HlaSubscriber



d) Class instance

Figure 3: Building Ptolemy federates (consumer)

[1] At the discovering of an object instance (in the federate that subscribes to this attribute using a HlaSubscriber), the name that will appear is "FederateP-name Actor-name", where FederateP is the federate (`producer` in figure 1.c) that publishes the object class and Actor is the composite actor (`P3 (producer)` in figure 1.c) connected to the HlaPublisher. This name must be unique in the federation.

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. For allowing the discovering of multiple instances, replace the input ports of composite actor `C2(consumer)` by input multiports as in figure 3.a.

4. Create a (Ptolemy) class of object as `TypedCompositeActor` (see [1] section 2.6) in the federate model for each class this federate will subscribe:

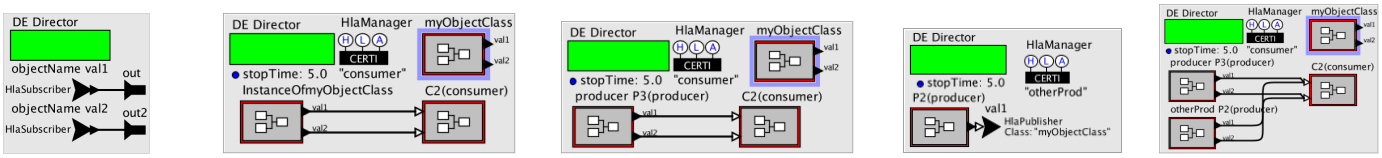
   (a) Drag a `CompositeClassDefinition` from `Utilities`; right-click for editing the actor (Customize/Rename) and rename it as a (HLA) class in the FOM in figure 2.c, MyObjectClass in this example (see figure 3.a).

   (b) Right-click on `CompositeClassDefinition`, Open actor (command-L) and populate it with and a DE director with the parameter `stopTime` > 0.0. Add as many HlaSubscriber actors (`MoreLibrairies->Co-Simulat` as the attributes of the (HLA) class this Federate subscribes to. In this example, federate `consumer` subscribes `val1` and `val2`, so add two HlaSubscriber actors (see figure 3.b).

   (c) Configure the HlaSubscriber: double-click on the first HlaSubscriber (the window of figure 3.c pops out) put `val1` in `Name of the parameter to receive` (the name of the attribute as in the `.fed` file of figure 2.c). Put a data type for this attribute in the field `type of the parameter`. Do the same for the other HlaSubscriber actor, name it `val2` and put a data type.

   (d) Create at least one instance of the (Ptolemy) class `myObjectClass`: right-click on the actor `myObjectClass`, `Class Actions -> Create Instance` (or command-I). The actor `InstanceofmyObjectClass` [2] appears (see figure 3.d). All changes in the (Ptolemy) class `myObjectClass` will appear automatically to its instances.

5. If the federate that subscribes to some class attribute will receive multiple instances, connect an output port to each HlaSubscriber actor inside the (Ptolemy) class as in figure 4.a.



a) Inside the class b) Connecting instance c) Discovering 1 instance d) A new federate e) New run

Figure 4: When multiple instances of a class will be discovered

6. Connect the outputs of `InstanceofmyObjectClass` to the input ports of actor `C2(consumer)` as depicted in figure 4.b. After running a federation with federates `producer` and `consumer`, the instance registered by Federate `producer` is discovered by Federate `consumer` and the name of `InstanceofmyObjectClass` changes to `producer P3(producer)` as indicated in step 4(d) above (see figure 4.c).

7. Each new discovered object will be dynamically instantiated and connected to the same actors that the preexistent instance was connected to. If the federation is launched with the same federates as in previous step and another federate called `otherProd` (figure 4.d) then the new object instance is registered as `otherProd P2(producer)` and connected to composite actor `C2(consumer)` as depicted in figure 4.e.

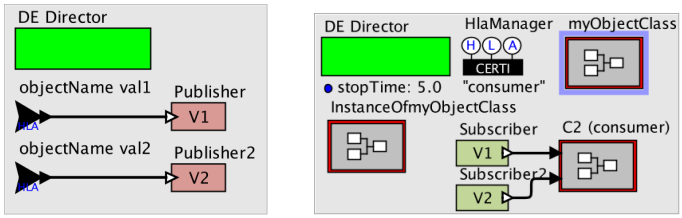
[2] The name displayed after running the model (i.e., after discover an instance of object) will be the concatenation of the federate name publishing the attribute (i.e. name of the HlaManager actor) and the name of the composite actor connected to the HlaPublisher actor, in this example, `producer P3(producer)`.

2.2.2 The federate will discover only one instance of a class

In some applications a federate will receive only one instance of a class, e.g, a federate `Quadrotor` that subscribes to attribute `position` published by a federate `GPS`. In the case the federate will not receive several instances of a same attribute, and if the federate subscribe to several classes with several attributes, it can be difficult to design a graphical model because there is a lot of links (between the output of the class instances and the input port of the following actor) crossing. In this case, instead of connect the `HlaSubscriber` to an output port (inside the `CompositeClassDefinition`), the `HlaSubscriber` is connected to a (Ptolemy) actor `Publisher` and the input port of the following actor is connected to a (Ptolemy) actor `Subscriber`. Actors `Publisher` (from `Actors->Sinks->GenericSinks`) and `Subscriber` (from `Actors->Source->GenericSources`), are (normal) Ptolemy actors and are used for avoiding the use of links. Do not mix up (Ptolemy) actor `Publisher` and (hlacerti) actor `HlaPublisher`.

The way to configure the `CompositeClassDefinition` and design such a federate is the following:

1. If the `.fed` file exists already, skip Step 1 above.
2. Step 2 is the same as above.
3. Step 3 depends on how the instances will be connected in the Federate discovering them. If each instance is connected to a different input port, then use input `port` (black icon); if more than one instance will be connected to a same input port (e.g., attributes `x` and `y` are connected to a `Display` inside the actor), then use an input `multiport` (white icon).
   FIXME: check if it works for a composite actor having ports and multiports.
4. Step 4 is the same as above.
5. Step 5 above is modified in this way: Instead of creating output ports in the (Ptolemy) class as in figure 3.c, the `HlaSubscriber` actors are connected to `Publisher` actors from (tick parameter `global` and name the channel, e.g., `V1` and `V2`) as depicted in figure 5.a. For each object to be discovered, one instance of the (Ptolemy) class must be created for each (object) instance that will be registered int the whole Federation.
6. Step 6 is modified in this way: Populate the Federate model with as many `Subscriber` actors as the `HlaSubscriber` actors inside the `CompositeClassDefinition` (see figure 5.a). Name each `Subscriber` actor as the name in the channel of the `Publisher` actor and connect to the corresponding input port of the actor (`C2 (consumer)` in this case) as represented in figure 5.b.



a) Inside Ptolemy class b) Ptolemy model

Figure 5: Building Ptolemy federates (consumer) with Publisher and Subscriber actors instead of output ports.

2.3 Configuring a federate that publishes and subscribes to attributes

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

- If Step 1 of section 2.1 is done, skip it. Do Step 2 of section 2.1 (duplicate the centralized model removing all composite actors except `PC1(producer-consumer)`) and Step 3 (drag `HlaManager` and `HlaPublisher`).
- Configure the `HlaManager` (step 5 of section 2.1). Pay attention to name the federate as `prodcons` in the configuration window of HlaManager (figure 2.d).

- Configure the `HlaPublisher` for `val2` as in step 4 of section 2.1.
- If you want to discover the object published by Federate `producer` but also the object published by Federate `otherProd`, replace the input port of `PC1` by an input multiport (Step 3 of section 2.2.1).
- Do Step 4 of of section 2.2.1 for just the attribute `val1` (since this Federate subscribes only to `val1`) [3].
- Do Step 5 of of section 2.2.1.
- Connect the output of `myObjectClass val1` to `PC1` actor.

Now you can run two Federations: F1 with Federates `producer`, `consumer` and `prodcons` and F2 with this Federates and `otherProd`.

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.

3 Running a federation

1. Open a terminal and run `cfgCerti` as said in the beginning of this manual or execute the shell `source $CERTI_HOME/share/scripts/myCERTI_env.sh`
2. Go to the folder where the 3 federate models are (or give the absolute address) and open the models: `$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 register?" ticked. Run the other models in any order but the last to be run is the one that has the cited field ticked.

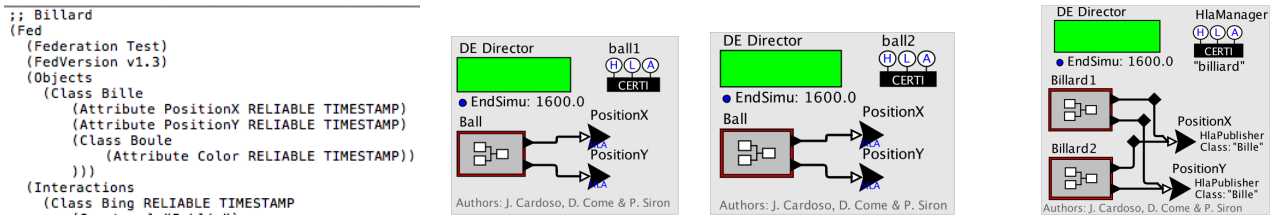
For being sure you have `myCERTI_env.sh` shell executed and the good path for `PTII` , there are two ways:

1. Put in your .bash_profile:
   `export PTII=your-path-to-ptII`
   `source your-CERTI_HOME/share/scripts/myCERTI_env.sh`
   You need to close the terminal and open a new one. Note that `your-path-to-ptII` and `your-CERTI_HOME` are absolute addresses.
2. Put aliases in your .bash_profile :
   `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.

4 Registering multiple instances of an attribute

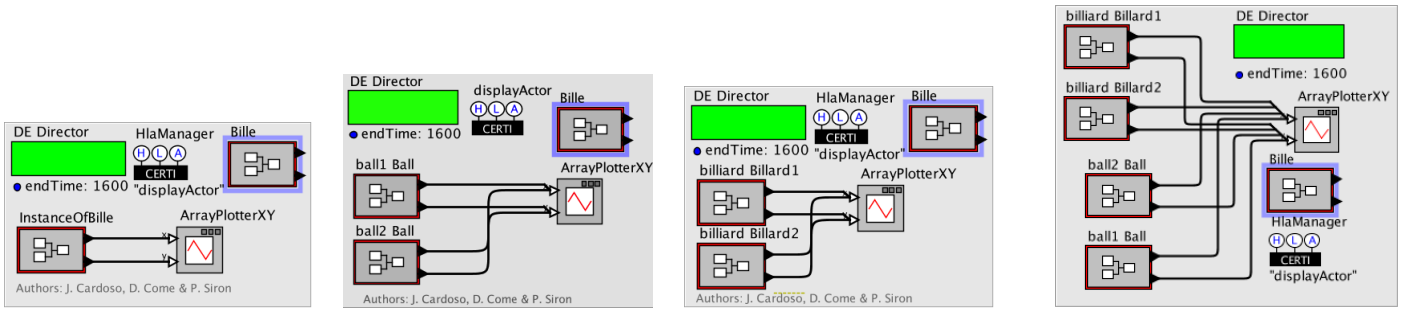
Let us consider a general case where several instances of object are published, as in the example of a billiard game, a demo of CERTI. For example, let us consider that we want to represent in a billiard table (federate `display` in figure 7) two instances of class `Bille` with attributes `positionX` and `positionY` as indicated in the FOM on figure 6.a. There are two ways of modeling:

- create two federates, each one registering one instance of class `Bille` as in figure 6.b and c, or
- create one federate that register two instances of class `Bille` as in figure 6.d. Notice that this is possible because the input port of `HlaPublish` actor is an input multiport (white arrow).



a) Test.fed (FOM) b) `ball1` federate c) `ball2` federate d) Two instances of `Bille`

Figure 6: Two different Ptolemy federates sending attributes `positionX, positionY` of class `Bille`



a) The Federate model b) Run (fig. 6.b + 6.c) c) Run with fig. 6.d d) Run with fig. 6.b,6.c and 6.d

Figure 7: `DisplayActor` registering multiple instances of attributes `PositionX` and `PositionY` from class `Bille`.

And what about the discovering of multiple instances?
As indicated in the HLA standard, there is no difference if the objects come from one or more federates. The simulation of billiard balls is a good example for the use of dynamic instances. Figure 7.a depicts the (initial) model of Federate `DisplayActor` used for displaying the movement of the simulated balls (attributes `positionX, positionY` of (HLA) class `Bille`): an instance of (Ptolemy) class `Bille` is connected to actor `ArrayPlotterXY` that has input multiports (white icons).

Let us consider a Federation with the Federates `ball1`, `ball2` and `DisplayActor` (figures 6.b, 6.c and 7.a respectively). After running this distributed simulation, the model of Federate `DisplayActor` is the one represented on figure 7.b. Note that a second instance of `bille` was inserted in the model, connected to same (following) actor, and each one is named properly with the (composed) name "federate + actor".

Figure 7.c depicts the model of Federate `DisplayActor` after running a distributed simulation with only Federates `billard` and `DisplayActor`. Pay attention to the name of the instances of class `bille`: they are now `billiard Billiard1` and `billiard Billiard2` (name of the Federate publishing + name of the composite actor inside it).

Figure 7.d depicts the model of Federate `DisplayActor` after running a distributed simulation with Federates `billard`, `ball1`, `ball2` and `DisplayActor`, which means four objects of class `bille` (coming from three federates).

This federation can run also C++ federates from CERTI demo (see section 7).

5 Important revisions

- Up to revision 71843: `hlacerti` was in `$PTII/ptolemy/apps/`; it was necessary to install CERTI and JCERTI; only one instance of a class could be registered; only NER time management was implemented. Code by Gilles Lasnier. On revision 71867 PTJCERTI_JAR was added to the CLASSPATH by C. Brooks (no need to install JCERTI anymore).

- Revision 71890: `hlacerti` was moved to `$PTII/org`; multiple instances of a class can now be registered thanks to David Come. Some demos were updated at r71919.

[3]You can copy the class `myObjectClass` from federate `consumer` and remove the actors related to `val2`. In this case, the following step is already done (connect to output ports).

- Up to revision 71935: some fixes as: fixed displayed name for HLASubscriber (r71932); encoding and decoding messages are in a separated class (r71924); an automatic translator for Ptolemy models was created (oneModelUpdate.py) on r71923; new demos by Janette Cardoso.
- Revision r72005: HLASubscriber must be in a composite actor for allowing dynamic (multiple) instances; Billard demo updated (r72045), fixed potential race condition due to non thread safe static objects (r72131), new (HLASubscribers) actors are added on run time if necessary (r72165), opaqueIdentifier renamed to objectName (r72187); new (HLASubscribers) actors are connected to existing actors in the model (r72204); manual and demo updated (r72244); HLA time unit created (r72431). All changes made by D. Come.
- Revision 72943: TAR time management was implemented by Yanxuan LI; new demo SimpleProducerMultipleConsu TAR added; variables are renamed for clarity (r73341).
- Revision r73687: Some cleanup of demo layout and minor tuning of icons.

6 Installing Ptolemy

For having Ptolemy and CERTI in a same root, you can create a folder $HOME/pthla and install Ptolemy inside.

```
mkdir $HOME/pthla
cd $HOME/pthla
svn co https://repo.eecs.berkeley.edu/svn-anon/projects/eal/ptII/trunk ptII
export PTII=$HOME/pthla/ptII
cd $PTII
./configure (there is a build.xml in $PTII)
ant
cd $PTII/bin
make
```

For open the graphical interface `vergil` in a terminal:

```
vergil $PTII/org/hlacerti/demo/Billard/FederationBillard.xml &
```

This demo is a federation with a billiard ball sending its location to a display.

7 Installing CERTI

```
-- Author: Gilles Lasnier (gilles.lasnier@isae.fr)
-- @version: $Id: INSTALL_CERTI.txt 73687 2015-10-20 00:58:57Z eal $

Install certi (an open source RTI following the hla standard) latest version.  As of 10/17/15, that
Download the gzipped tar file from:
http://download.savannah.gnu.org/releases/certi/

Here a receipe for installing certi:
   mkdir $HOME/pthla (only if you did not yet created this folder)
   cd $HOME/pthla
   tar xvfz CERTI-3.5.1-Source.tar.gz # (or double click to expand)
   mv CERTI-3.5.1-Source certi # (or create a symbolic link: ln -s CERTI-3.5.1-Source certi)
   mkdir $HOME/pthla/certi-tools
   cd $HOME/pthla/certi
   mkdir $HOME/pthla/certi/build-certi
   cd $HOME/pthla/certi/build-certi
   cmake -DCMAKE_INSTALL_PREFIX=$HOME/pthla/certi-tools  ../
```

```
   make
   make install

CERTI has been compiled and installed in the $HOME/pthla/certi-tools folder defined as $CERTI\_HOME
folder. CERTI provides a script to set the environment (global variables, binaries, etc) to allow th
correct launch of RTIG process, RTIA process and federates. To set the CERTI environment properly
in a terminal run the command:
  source $HOME/pthla/certi-tools/share/scripts/myCERTI_env.sh

or put this command in your ~/.bash_profile file:

To test the installation:

1. open 3 terminals (make sure you open new terminals or source ~/.bash_profile in already open term
2. go to the first terminal and execute the command "rtig"
3. go to second terminal and call a billard federate "1" (-n name)
      billard -n1 -fTest -FTest.fed -t10
   DO NOT HIT ENTER YET.
4. go to the third terminal and call a billard federate "2" (-n name)
      billard -n2 -fTest -FTest.fed -t10
   DO NOT HIT ENTER YET.
5. go back to second terminal (of step 3) and press "ENTER"

The C++ billard demo that has been run is in $HOME/pthla/certi/test/Billard/
The Ptolemy billiard demo is in $PTII/org/hlacerti/demo/Billard
```

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] Giles, L. *Toward a Distributed and Deterministic Framework to Design Cyber-PhysicalSystems*, ISAE Report 2013.

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

[6] Yanxuan LI. A Distributed Simulation Environment for Cyber-physical systems. Report ISAE-SUPAERO, October 2015.

[7] E. Noulard, J.-Y. Rousselot, and P. Siron, CERTI, an open source RTI, why and how? Spring Simulation Interoperability Workshop, 2009.

Appendices

A Check list for creating Federates using `hlacerti`

1. Have CERTI installed and a .fed file with the FOM.
2. The top level director must be DE (Discrete Event). There must be only one Composite actor for each instance of object.
3. Add a HlaManager decorator from `MoreLibrairies->Co-Simulation->HLA` and configure it: name the Federate (must be unique in the Federation) and the Federation (the same for all federates), browse the .fed file, choose the time management NER or TAR (if TAR, choose also the time step), put values for Lookahead and HLA time unit. If federates have a synchronization point, tick the field and choose a same name for all federates in this federation. Choose a federate to be the last one to be launched, and tick the field "Is synchronization point register"?
4. If the Federate will send values (of an attribute) for other federates, add a `HLAPublisher` for each attribute (in the FOM) to be Published to the Federation. If a same Federate has several object instances to be sent (e.g. figure 6.d), just connect each output port corresponding to the attribute to a same HlaPublisher. Name it with the attribute, put the class name, choose the good data type in its input port.
5. If the Federate will receive values (of an attribute) from other federate(s), there is two steps:
   (a) Drag a `CompositeClassDefinition` from `Utilities`, rename it with the class (in the FOM), populate it with a DE director and `stopTime`> 0 and a `HLASubscriber` to each attribute the Federate will subscribe; name the `HLASubscriber` with the attribute name. If the Federate will discover multiple instances of an attribute, connect an output port to each `HLASubscriber`. Otherwise, see section 2.2.2. Choose the good data type.
   (b) Create at least one instance of each (Ptolemy) class the Federate will subscribe and connect its outputs to the actors in the Federate.

   Remark: In the case of multiple instances to be discovered, the output port of the (Ptolemy) class must be connected to an (input) multiport of the composite actor (if there are more than one actor in the model) or an atomic actor.

B Error messages

The error messages bellow were obtained running the models of figures 1.c, 4.d and 4.b.

B.1 RTIinternalError "Connection to RTIA failed"

When: trying to run the first federate.

```
ptolemy.kernel.util.IllegalActionException: RTIinternalError. If the error is "Connection to RTIA
failed", then the problem is likely that the rtig binary could not be started by CertRtig. One way
to debug this is to set the various environment variables by sourcing certi/share/scripts/myCERTI_en
then invoking rtig on the .fed file then rerunning the model.
  in .producerP1.HlaManager
Because:
Connection to RTIA failed. libRTI: Network Read Error
at org.hlacerti.lib.HlaManager.initialize(HlaManager.java:580)
(...)
```

Raison: $CERTI_HOME is not set.
Solution: source the file myCERTI_env.sh. If you used the recipe presented in section 7, do:
source $HOME/pthla/certi/share/scripts/myCERTI_env.sh

B.2 RTIinternalError 4

When: after running a Federation. The error message comes from the Federate `name-of-federate` that subscribes some `attribute`. The HlaManager become red and the message is:

```
ptolemy.kernel.util.IllegalActionException: RTIinternalError
  in .name-of-federate.HlaManager
Because:
4
            at org.hlacerti.lib.HlaManager.proposeTime(HlaManager.java:768)
            (...)
            Caused by: hla.rti.RTIinternalError: 4 serial:0
```

Raison: The input port of the HLAPublisher (named `attribute`) of some federate publishing this attribute has data type "unknown", or the wrong data type.
Solution: Check if the data type of the input port of all HlaPublisher of all federates are correctly set.

B.3 RTIinternalError

When: trying to run a Federation.
Message:

```
ptolemy.kernel.util.IllegalActionException: RTIinternalError
  in .consumerC3Singleport.HlaManager
Because: Received a RAV but could not put in any object. Means the ChangeRequest has not been
processed yet
   at org.hlacerti.lib.HlaManager.proposeTime(HlaManager.java:768)
   (...)
```

Raison: the number of discovered instances does not correspond to the number registered instances. One reason is that the number of instances of a class (declared in some federate or script) in the Federation does not correspond to the instances really created.
Solution: Check if the instances declared are the one really created and act accordling: change the declaration or launch the one(s) declared.
Example: a Federation with Ptolemy federates P_i-fed (where each one registers an instance I_i) and a Morse federate M-fed that will discover these instances. The error occurs if only one instance is declared (eg I_1) in the .py script but two Ptolemy federates P_i-fed are launched creating two instances (or vice-versa).

B.4 Ptolemy message "Cannot find class:"

When: after successfully running a Federation, trying to save a Federate `consumerC3Multiport` that subscribes to class "myObjectClass" from the FOM, with a `CompositeClassDefinition` named `myObjectClass`. `otherProd P2(producer)` is the name of the instance of `myObjectClass` discovered by the Federate (after running the Federation).
Message :

```
Error encountered in XML element:
<entity name="otherProd P2(producer)" class= ".consumerC3Multiport.myObjectClas...
ptolemy.kernel.util.IllegalActionException:
Cannot find class: .consumerC3Multiport.myObjectClass. In Ptolemy, classes are typically Java
.class files. Entities like actors may instead be defined within a .xml file.  In any case, the
class was not found. If the class uses a third party package, then the class would be present
only if the third party package was found at compile time.  It may be necessary to upgrade
Java or install the third party package, reconfigure and recompile.
Because:
-- /consumerC3Multiport/myObjectClass.xml (No such file or directory)
```

```
-- XML file not found relative to classpath.
-- /Users/j.cardoso/PtolemyHLA/manual/modelsR73687//consumerC3Multiport/myObjectClass.xml
/consumerC3Multiport/myObjectClass.xml (No such file or directory)
 in file:/Users/j.cardoso/PtolemyHLA/manual/modelsR73687/consumerC3Multiportxx.xml at line 222 and c
```

Raison: The class is not defined in the library (??)
Solution: do not use "File/Save as"; duplicate the model before save then use "File/Save". Change the name of the original model or the saved model.

B.5 Nothing is received from the Federate that subscribes and there is no error message

When: running a Federation with one producer and one consumer.
Message: No error message but no attribute value is received. Raison: the output ports of a `CompositeClassDefinition` containing `HLASubscriber` must be connected to a multiport.
Solution: replace the single (input) port by a multipart (white icon) on the side of the Consumer federate.

B.6 Ptolemy message "Attempt to link more than one relation to a single port."

When: trying to run a Federation with two producers and one consumer. A class instance is automatically created but its output is not connected to the expected actor.
Message :

```
ptolemy.kernel.util.IllegalActionException: Attempt to link more than one relation to a single port.
  in .consumerC3Singleport.CompositeActor.in and .consumerC3Singleport.prod2 prod val
```

Raison: the output ports of a `CompositeClassDefinition` containing `HLASubscriber` must be connected to a multiport.
Solution: replace the single (input) port by a multipart (white icon) on the side of the Consumer federate.

B.7 Ptolemy message "Error encountered in XML element:"

When: Opening a Federate model (that subscribes to some attribute). If you skip all elements, it should work anyway. But is advised to correct the model.
Message :

```
Error encountered in XML element:
<param name="opaqueIdentifier" value="fed"/>
ptolemy.kernel.util.IllegalActionException: Error evaluating expression:
fed
  in .PoolTable2OneInstance.InstanceOfBille.HlaSubscriber.opaqueIdentifier
Because:
The ID fed is undefined.
  in .PoolTable2OneInstance.InstanceOfBille.HlaSubscriber.opaqueIdentifier
  (...)
```

Raison: You opened an model from a previous revision (that used `fed` for discovering multiple instances).
Solution: Note the name of the attribute and data type of the `HlaSubscriber`, remove the `HlaSubscriber`, drag a new one and configure it.