User Manual for Ptolemy-HLA federates

Janette Cardoso, April 2018

Contents

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 [12]), and is a easy way to produce a HLA federate in a Federation using CERTI. Ptolemy and so the HLA-PTII co-simulation framework is available for the Linux, Windows XP and Mac OS X operating systems. The HLA-PTII framework (called `hlacerti` in Ptolemy tree) is an on-going work[1]. For more information about the HLA-PTII co-simulation framework read [4, 5, 6, 7, 8, 9].

This user guide contains the following sections:

- A brief presentation of a Ptolemy-HLA federation. This allows to understand the main features of the HLA-PTII co-simulation framework.
- Getting Started: First is presented how to execute a demo federation. Then, the instructions for creating a new federation with new federates is presented. HLA-PTII demos are in `$PTII/org/hlacerti/demo`.

[1] Contributors implied in this framework from the beginning in 2013 up to now, March, 2018 (in alphabetical order): Vandita Banka, Christopher Brooks, Tarciana Cabral de Brito Guerra, David Come, Patricia Derler, Maxim Ivanov, Sebastien Jaillant, Gilles Lasnier, Edward Lee, Yanxuan Li, Clement Michel, Claire Pagetti.

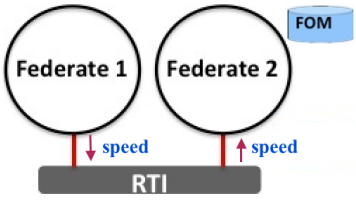


Figure 1: HLA architecture..

```
;;FOM with 1 object class and 1 attribute.
(Fed
 (Federation Test)
 (FedVersion v1.3)
 (Spaces)
 (Objects
  (Class ObjectRoot
   (Attribute privilegeToDelete reliable timestamp)
   (Class RTIprivate)
     (Class Signal
       (Attribute speed reliable timestamp))))
```

Figure 2: FOM.

- Installing the HLA-PTII co-simulation: which softwares you need and how to install Ptolemy and CERTI.
- There are also FAQ, Error Messages and previous versions of this framework.

2 Ptolemy-HLA federation

2.1 Some basics about HLA standard

The IEEE High-Level Architecture (HLA) standard [2] targets distributed simulation. A CPS can be seen as a federation grouping several federates which communicate via publish/subscribe patterns. This decomposition into federates allows to combine different types of components such as simulation models, executable code (in C++, Java, etc.), and hardware equipment. The key benefits of HLA are interoperability and reuse [4].

A simulation entity performing a sequence of computations is called a *federate*, and the set of federates simulating the entire system is called a *federation*. Federates are connected via the Run-Time Infrastructure (RTI), the underlying middleware functioning as the simulation kernel.

The HLA specification defines [5]:

1. An interface specification for a set of services required to manage the federates and their interactions. For instance, it describes how a federate can join or create a federation.
2. An object model template which provides a common framework for the communication between HLA simulations. For each federation, a Federation Object Model (FOM) describes the shared objects and their attribute. This federation object model is usually specified in a Federation Execution Data (FED) file.
3. A set of rules describing the responsibilities of federations and the federates. An example is the rule that *all data exchange among federates shall occur via the RTI.*

2.1.1 Data Exchange in HLA

Let us say that the *attribute* `speed` belongs to a *class* called `Signal`. Let us consider the federation in fig. 1 with two federates: Federate **2** uses the data `speed` provided by Federate **1**, i.e., Federate **1** publishes the class `Signal` and Federate **2** subscribes to attribute `speed` of this class. There are two steps concerning the *object management* [4]:
1) When federate **1** is launched, it registers an object instance of `Signal` class. When federate **2** is launched, it discovers object instances `Signal` related to the attribute `speed` it subscribed.
2) During the simulation, **1** sends through the RTI a new value of `Signal.speed` using the service `updateAttributeValues` (UAV). The RTI sends this value to **2** using the callback `reflectAttributeValues` (RAV).

2.1.2 Time Advance in HLA

The time advance phase in HLA is a two-step process: 1) a federate sends a time advance request service, and 2) waits for the time to be granted, provided by `timeAdvanceGrant` (TAG) service. There are two services for a time advance request: the `timeAdvanceRequest` service (TAR), used to implement time-stepped federates; and the `nextEventRequest` service (NER), used to implement event-based federates. For more information about time advance, a very important point, see [4].

2.2 Ptolemy-HLA framework

The Ptolemy-HLA co-simulation framework must comply with both, HLA and Ptolemy rules, in particular when dealing with data exchange and time advance. See see [4] for more technical information.

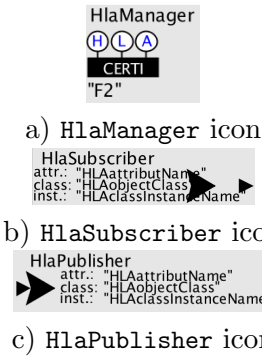


Figure 3: Icons.

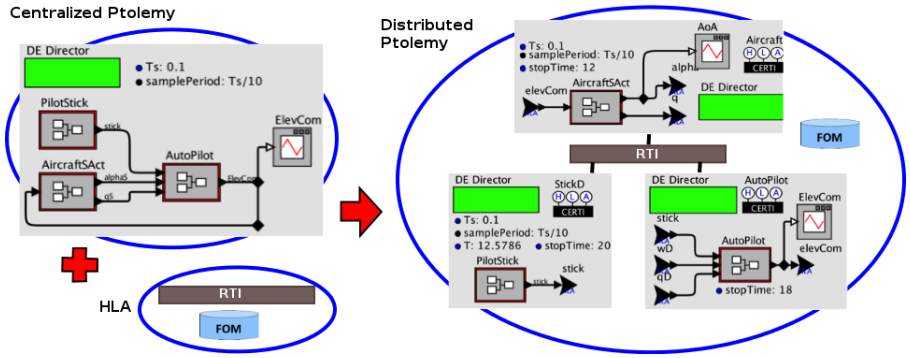


Figure 4: A Ptolemy-HLA federation from a centralized Ptolemy model.

Three new components were added to Ptolemy whose icons are shown in fig. 3:

- `HlaManager`: This interface handles: 1) the time coordination between the the Ptolemy logical time and HLA logical time, 2) the data exchange between federates (with `HlaSubscriber` and `HlaPublisher`).
- `HlaPublisher`: Registers the object instance and sends the data through the RTI.
- `HlaSubscriber`: Discovers the object instance and receives the data from the RTI.

Figure 4 shows a (centralized) Ptolemy model split in three federates. You can notice the icons `HlaManager`, `HlaPublisher` and `HlaSubscriber` on the left size.

3 Getting Started

3.1 Creating federates of a federation Test

If you have already installed Ptolemy, follow the steps bellow. Otherwise, see section 5.1.

Let us consider the (centralized) Ptolemy model depicted in figure 5. We want to create a federation called **Test** with two federates. The composite actor **A1** will be implemented in Federate **1** and **A2** will be implemented in Federate **2**. Now the data `speed` in fig. 5 will be sent through the RTI (as represented in fig. 1). The splinting of the centralized model into two federates is done following the steps below. The three icons of the framework can be found in `MoreLibrairies->Co-Simulation->HLA` [2].

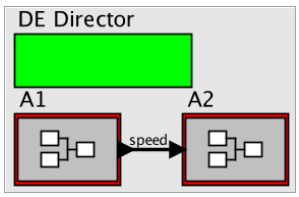


Figure 5: A Ptolemy model.

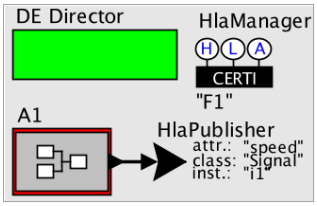


Figure 6: Federate **1**.

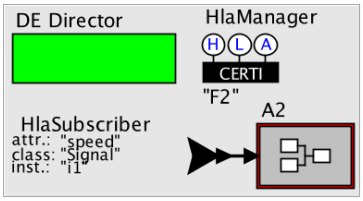


Figure 7: Federate **2**.

3.1.1 Creating federates

Create a folder that will be populated with the federates and a `.fed` file describing the FOM. We will use for this example the FOM depicted in fig. 2.

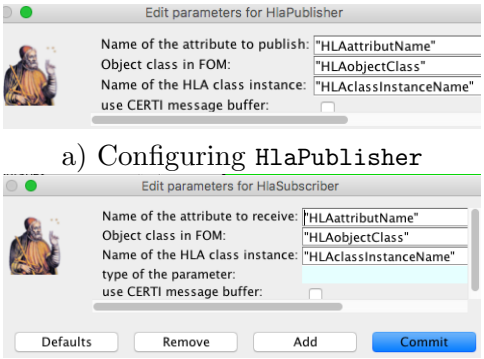
[2] For create Ptolemy models, see chapter *Building Graphical Models* in [1].

Federate **1**:

1. Create a new model[2] in the folder, populate with a DE director (mandatory); save it, e.g., as `Federate1.xml`,
2. Copy the composite actor **A1** from the centralized model,
3. Check if the output port of **A1** has a type; if not, choose a type[2],
4. Drag an `HlaManager` icon and an `HlaPublisher` icon; connect the latter to the output port of **A1**. For configuring the icons, see sections 3.1.2 and 3.1.4. The final result is depicted in fig. 6.

Federate **2**:

1. Create a new model in the same folder, populate with a DE director (mandatory); save it, e.g., as `Federate2.xml`,
2. Copy the composite actor **A2** from the centralized model,
3. Drag an `HlaManager` icon and an `HlaSubscriber` icon; connect the latter to the input port of **A2**. For configuring the icons, see sections 3.1.3 and 3.1.4. The final result is depicted in fig. 7.



a) Configuring `HlaPublisher`

b) Configuring `HlaSubscriber`

Figure 8: Configuring actors.

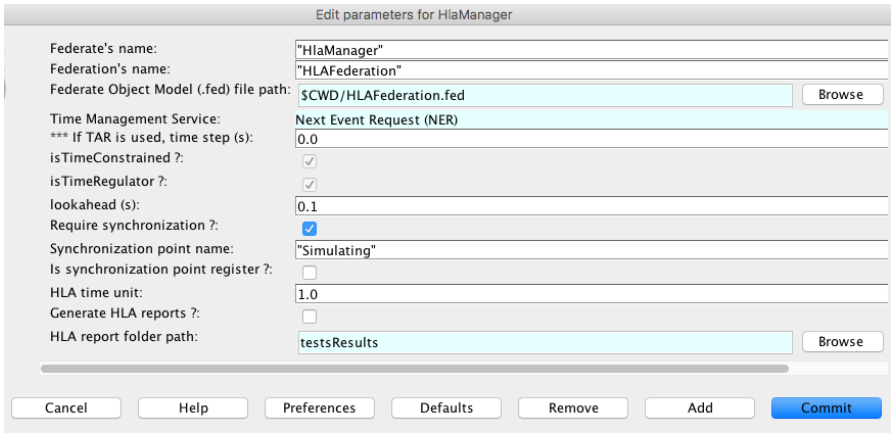


Figure 9: Configuring `HlaManager`

3.1.2 Configuring `HlaPublisher`

In the model `Federate1.xml` (fig. 6), double-click on the icon `HlaPublisher`; the window depicted in fig. 8.a pops out. Replace `HlaAttributeName` by `speed` and `HlaObjectClass` by `Signal`. As for the moment, put any name, e.g., `i1` in the field `class instance`. We will talk again about this parameter when presenting an example with multiple instances in section 3.2. Click `Commit`. Do not mind for now about field `use CERTI message buffer`.

3.1.3 Configuring `HlaSubscriber`

In the model `Federate2.xml` (fig. 7), double-click on the icon `HlaSubscriber`; the window depicted in fig. 9.b pops out.

Replace `HlaAttributeName` by `speed` and `HlaObjectClass` by `Signal`. In the field `class instance`, put the same name used in the `HlaPublisher`, `i1`. In the field `type of the parameter`, put the *same* type as the one in the input of the actor that publishes `Signal.speed` in fig. 6 (chosen in section 3.1.1, Federate **1**, step 3).

3.1.4 Configuring `HlaManager` of all federates in a Federation

Some important points to have in mind:

1. All federates must use the same Federation name that appears in the `.fed` file.
2. All federates must use the same `Synchronization Point Name`.

3. Each federate can choose its own time management, NER or TAR.
4. Each federate can choose to save its execution in .csv files in `$HOME/testsResults`
5. The last federate (Ptolemy model) to be launched must be the *register of the synchronization point*. FIXME: talk about synchronization point somewhere, or just cite a reference/code??

Federate **1**:

1. In the model `Federate1.xml` (fig. 6), double-click on the icon `HlaManager`; the window depicted in fig. 8.a pops out.
2. In the field `Federate's name`, replace the default `HlaManager` by `F1`.
3. In the field `Federation's name`, replace the default `HlaFederation` by `Test`.
4. Beside the field `Federate Object ...`, click on Browse button and select the `.fed` file. It *must* be in the same folder where the federate is. The federate is the Ptolemy model `Federate1.xml`.
5. In the field `Time Management Service`, choose `Next Event Request (NER)`. Do not mind about the value of the time step in the field below.
6. In the field `lookahead`, keep the default value or choose another one. FIXME: talk about lookahead somewhere, or just cite a reference??
7. Tick the field `Require synchronization?`; keep the default value the field `Synchro... point name` or choose another name.
8. Keep unticked the field `Is synchronization point register?`. This model will be the first to be launch.
9. You may tick `Generate HLA reports?`.

Federate **2**:

1. In the model `Federate2.xml` (fig. 7), double-click on the icon `HlaManager`; the window depicted in fig. 8.a pops out.
2. In the field `Federate's name`, replace the default `HlaManager` by `F2`.
3. In the field `Federation's name`, replace the default `HlaFederation` by the same name, `Test`.
4. Beside the field `Federate Object ...`, click on Browse button and select the `.fed` file. It *must* be in the same folder where the federate is. The federate is the Ptolemy model `Federate2.xml`.
5. In the field `Time Management Service`, choose `Next Event Request (NER)`. Do not mind about the value of the time step in the field below.
6. In the field `lookahead`, keep the default value or choose another one.
7. Tick the field `Require synchronization?`; keep the default value the field `Synchro... point name` or choose another name.
8. Tick the field `Is synchronization point register?`. This model will be the last to be launch.
9. You may tick `Generate HLA reports?`.

3.2 Using multiple instances of a class

FIXME: talk about multiple instances and give an example.

4 Running a Ptolemy-HLA Federation

If you have already installed CERTI, follow the steps bellow in *this* order. Otherwise, see section 5.2.

1. Open a terminal and execute the script
   `source $CERTI_HOME/share/scripts/myCERTI_env.sh`
2. In the same terminal, check if environnement variable PTII is set (echo $PTII). Otherwise, set this variable.
3. In the same terminal, open all the federates of the Federation. In this example, is `Federate1.xml` and `Federate2.xml`. Go to the folder where the 2 models are (or give the absolute address) and open the models:
   `$PTII/bin/vergil Federate1.xml Federate2.xml &`
4. Check there is no `rtig` process running (the first model to be run will automatically launch this process): `ps -ax | grep rtig`. If there is a `rtig` running, kill the process: `pkill rtig`
5. Check there is only one model that has the field "Is synchronization point register?" ticked. If the instructions were followed, `Federate2.xml` is the register. Run first `Federate1.xml` then run `Federate2.xml`.

5 Installing Ptolemy-HLA framework

5.1 Installing Ptolemy

For having Ptolemy and CERTI in a same root, you can create a folder $HOME/pthla and install Ptolemy inside. These instructions works well for Linux and Mac OS (10.8 Mountain Lion, El Capitan, 10.12 Sierra).

You need to have Java 1.8. You can use `make` if you do not have `ant` in step 7 below.

```
1.  mkdir $HOME/pthla
2. cd $HOME/pthla
3. git clone https://github.com/icyphy/ptII
4. export PTII=$HOME/pthla/ptII
5. cd $PTII
6. ./configure
7. ant
8. cd $PTII/bin
9. 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. If it does not work, you need to put the whole address `$PTII/bin/vergil` or add `$PTII/bin` to your PATH (in .bash_profile).

5.2 Installing CERTI

Since commit `5bcd48f1070` in `https://github.com/icyphy/ptII` there is a script that installs CERTI 4.0.0 in `$HOME/pthla/certi-tools`.

```
cd $PTII
./org/hlacerti/build-certi.sh
```

If for some reason the script does not work, you can install by yourself (see section A). You can check if there is any bug related to CERTI in https://savannah.nongnu.org/bugs/?group=certi.

FIXME: update the section A

FIXME: update the section B; is it useful?

6 FAQ

FIXME: Does any one has some suggestions?

7 Error Messages

FIXME: Talk about the test already made in org/hlacerti/test/auto ?

References

[1] Claudius Ptolemaeus, editor. *System Design, Modeling, and Simulation Using Ptolemy II*. Ptolemy.org, 2014.

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

[3] 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.

[4] J. Cardoso and P. Siron. *Ptolemy-HLA: a Cyber-Physical System Distributed Simulation Framework* in Principles of Modeling, Edward A. Lee Festschrift, P. Derler, M. Lohstroh, M. Sirjani, Eds, 2018.

[5] 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.*

[6] 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.

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

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

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

[10] Tarciana Cabral de Brito Guerra. *Performance Analysis of the Framework Ptolemy-HLA*. Technical Report ISAE-SUPAERO/DISC/RT2016/ 2.

[11] Clement Michel. *Distributed simulation of Cyber-Physical Systems.* Technical Report ISAE-SUPAERO/DISC/RT2016/ .

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

Appendices

A Installing CERTI

– Author: Gilles Lasnier (gilles.lasnier@isae.fr), Janette Cardoso (janette.cardoso@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 10/05/16, that was version 3.5.1. Download the gzipped tar file from: `http://download.savannah.gnu.org/releases/certi/`

Here a receipe for installing certi:

```
1.  mkdir $HOME/pthla (only if you did not yet created this folder)
2.  cd $HOME/pthla
3.  tar xvfz CERTI-3.5.1-Source.tar.gz # (or double click to expand)
4.   mv CERTI-3.5.1-Source certi
5.  mkdir $HOME/pthla/certi-tools
6.  cd $HOME/pthla/certi
7.  mkdir $HOME/pthla/certi/build-certi
8.  cd $HOME/pthla/certi/build-certi
9.  cmake -DCMAKE_INSTALL_PREFIX=$HOME/pthla/certi-tools  ../
10.  make
11.  make install
```

Remark: for having a faster simulation(using , replace line 9 by this command (in a same line):

```
 cmake -DCMAKE_INSTALL_PREFIX=$HOME/pthla/certi-tools
                                        -DCERTI_USE_NULL_PRIME_MESSAGE_PROTOCOL=ON ../
```

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 the 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 the above 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 ter
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.

Remark: At this date (May, 10, 2016), there is an issue with macos El Capitan. Even if CERTI and Ptolemy are successifuly installed, the demos in the folder `$PTII/org/hlacerti/demo` do not work, with models stuck with the message `initializing.` (in the left botton corner) and the following error message:

```
_read(): dyld: Library not loaded: libCERTId.3.dylib
  Referenced from: /Users/your-login/pthla/certi-tools/bin/rtig
  Reason: image not found
```

A fix was done on revision r74769 but please check out the (update) explaination in:
`https://chess.eecs.berkeley.edu/ptexternal/wiki/Main/HLA#ElCapitan`

```
1. Run the command:
ls -l $CERTI_HOME/lib/libCERTId.*
```

```
2. If you find a symbolinc link for file libCERTId.3.dylib, do the following commands:

a. cd $HOME/pthla/certi-tools/lib
b. mv libCERTId.3.dylib foo-libCERTId.3.dylib
c. cp libCERTId.3.5.1.dylib  libCERTId.3.dylib

3. run a demo again:
$PTII/bin/vergil $PTII/org/hlacerti/demo/2Billes2Fed/2Billes2Fed.xml &
```

B 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).
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. *fixme: change* If a same Federate has several object instances to be sent (e.g. figure **??**.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) *fixme: change*
   (b) 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 **??**. Choose the good data type.
   (c) Create at least one instance of each (Ptolemy) class the Federate will subscribe and connect its outputs to the actors in the Federate.