Ptolemy-HLA Framework
User Manual V0.1

Janette Cardoso, June 2018

Contents

1 Introduction

The Ptolemy-HLA 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 in this framework by CERTI [13]), and is a easy way to produce an HLA federate in a Federation. Ptolemy and so the Ptolemy-HLA co-simulation framework is available for the Linux, Windows XP and Mac OS X operating systems. The Ptolemy-HLA framework (called `hlacerti` in Ptolemy tree) is an on-going work[1]. For more information about the Ptolemy-HLA co-simulation framework read [4, 6, 7, 8, 9, 10].

This user guide contains the following sections:

- A brief presentation of a Ptolemy-HLA federation in section 2. This allows to understand the main features of the co-simulation framework. Section 3 presents how to execute a demo federation.
- Getting Started: The instructions for creating a federation with federates is presented in section 4.
- Installing the HLA-PTII co-simulation (section 5): which softwares you need and how to install Ptolemy and CERTI.
- There are also FAQ (section 6), Error Messages (section 7) and some information about previous versions of this framework (section D).

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

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. This is frequently represented by the lollipop view as in fig. 2.

The HLA specification defines [6]:

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 attributes (see sidebar on page 3).

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.*

```
;;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 1: FOM.

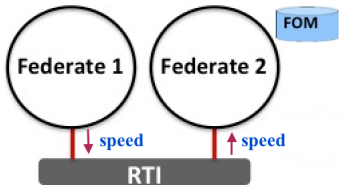


Figure 2: HLA architecture.

Table 1: Publish/Subscribe table.

Attribute	Fed. **1**	Fed. **2**
`Signal.speed`	Publishes	Subscribes

Sidebar: Federation Object Model (FOM)
The federation object model is usually specified in a Federation Execution Data (FED) file as the one in figure 1. How to specify this model? It depends how many federates the federation has. Or rather, which attribute is published by a federate. Even if HLA standard allows to change the ownership of a class, the Ptolemy-HLA framework do not do this. If a federate publishes a class instance, no other federate can publishes the same attribute of this class. But it can publishes an other attribute of the same class. See [14] for more details.

2.1.1 Data Exchange in HLA

Let us say that the *attribute* `speed` belongs to a *class* called `Signal`, as described in the FOM represented in fig. 1. Let us consider the federation in fig. 2 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. This information is usually provided in a table as the one in table 1. 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, federate **1** sends through the RTI a new value of `Signal.speed` using the service `updateAttributeValues` (UAV). The RTI sends this value to federate **2** using the callback `reflectAttributeValues` (RAV).

A UAV service has the following parameters: the object instance handle, the attribute value and a timestamp. The RAV service has a class handle, an attribute handle, a class instance name, an attribute value and a timestamp.

A federation can deal with several instances of a class, registered by different federates or a same federate. The Section 4.2 describes two different ways of implement a Billard federation.

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.

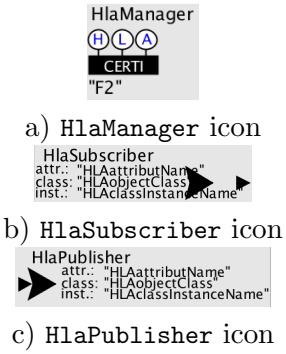


Figure 3: Icons.

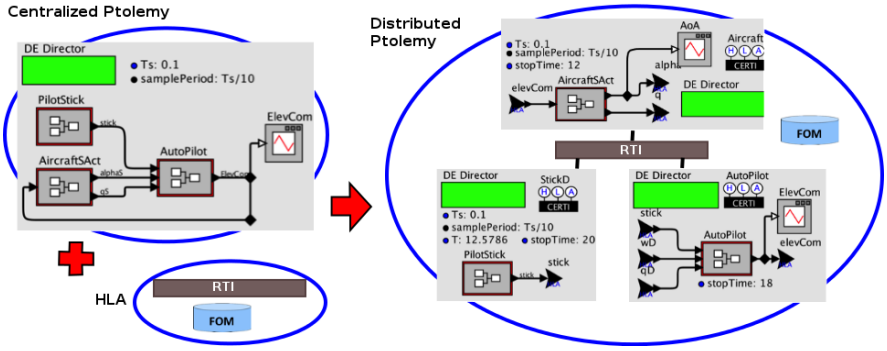


Figure 4: A Ptolemy-HLA federation from a centralized Ptolemy model.

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`: This actor implements a publisher in a federation; it relates a Ptolemy token to an object class attribute described in the FOM.
- `HlaSubscriber`: This actor implements a subscriber in a federation; it relates an object class attribute to a Ptolemy token.

Figure 4 shows on the left a (centralized) Ptolemy model split (on the right) in a federation with three federates [2]. You can notice the addition of the icons `HlaManager`, `HlaPublisher` and `HlaSubscriber` on the right side. You need also a FOM (see section 6), represented in fig. 5, construct a publish/subscribe table as the one in Table 2 and ... a RTI.

[2]This demo can be found in `$PTII/org/hlacerti/demo/f14HLA/AllFederatesNER/f14HlaNER.xml`

```
(Fed
  (Federation PRISE_V2)
  (FedVersion v1.3)
  (Objects
    (Class AIRCRAFT
        (Attribute alpha   RELIABLE TIMESTAMP)
        (Attribute q       RELIABLE TIMESTAMP))
    (Class ACTUATOR
        (Attribute elevCom RELIABLE TIMESTAMP))
    (Class JOYSTICK
        (Attribute ELEVATOR RELIABLE TIMESTAMP)
```

Figure 5: FOM for F14 federation.

Attribute	AutoPilot	Aircraft	PilotStick
AIRCRAFT.alpha	Subscribes	Publishes	–
AIRCRAFT.q	Subscribes	Publishes	–
ACTUATOR.elevCom	Publishes	Subscribes	–
JOYSTICK.ELEVATOR	Subscribes	–	Publishes

Table 2: Publish/Subscribe table for F14 federation.

3 Running a Ptolemy-HLA Federation Demo

If you have already installed CERTI and Ptolemy, follow the steps bellow in the *order* presented bellow. Otherwise, see section 5. The demos are in `$PTII/org/hlacerti/demo`.

1. Open a terminal and check if the environment variable CERTI_HOME is set (`echo $CERTI_HOME`). Otherwise, set this variable (`export CERTI_HOME=$HOME/pthla/certi-tools`).

2. In the terminal, execute the script
   `source $CERTI_HOME/share/scripts/myCERTI_env.sh`

3. In the same terminal, check if the environment variable PTII is set (`echo $PTII`). Otherwise, set this variable (`export PTII=/path/to/your/ptII`).

4. For avoiding errors[3], check if there is any `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. In the same terminal, go to `$PTII/org/hlacerti/demo` and choose a demo. All demos are in a folder `demo-name`; the instructions of how run the models are in the file `demo-name.xml` inside this folder. Remark: only one model must has the field "Is synchronization point register?" ticked; this one is the last one to be launched.
   ```
   cd $PTII/org/hlacerti/demo
   $PTII/bin/vergil demo-name/demo-name.xml &
   ```

There are four demos:

IntegrationTests There are three federations: TimeAdvancing1Federate, TimeAdvancing2Federates and TimeAdvancing2FederatesIntervalEvents. There is no data exchange; the goal is to show how the time is advanced using HLA time management services TAR and NER [4].
`$PTII/bin/vergil  IntegrationTests/IntegrationTests.xml &`

SynchronizeToRealTime This federation has two federates exchanging data, and a third federate that does not send neither receive data but has its logical time synchronized with real time. So the other 2 federates advance their logical time coordinated with it.
`$PTII/bin/vergil  SynchronizeToRealTime/SynchronizeToRealTime.xml &`

f14HLA A distributed simulation of the longitudinal control of a F14 aircraft. There are two federations, one using TAR and another using NER.
`$PTII/bin/vergil  f14HLA/f14HLA.xml &`

Billard A distributed simulation of two billiard balls in a pool table. The models are simplified: a ball does not change its direction when it hits another ball. There are two versions:

[3]For example, if you run another federation before, and, because of some exception, the federation was not properly destroyed, the current execution can raise other exception as *Federate already registered*, see section 7.

– 2Billes2Fed: two federates, each one registers and publishes a unique ball;
– 2Billes1Fed: one federate that register two balls (two instances of class Bille).
Both federations have another federate that just displays the coordinates of each ball.
`$PTII/bin/vergil  Billard/Billard.xml &`

4 Getting Started

4.1 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 6. 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. 6 will be sent through the RTI (as represented in fig. 2). 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` [4].

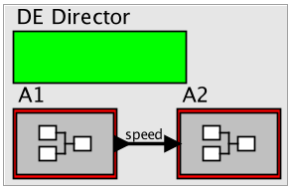


Figure 6: A Ptolemy model.

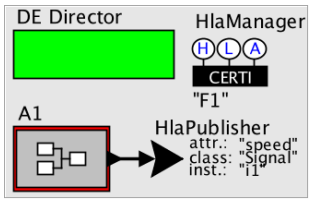


Figure 7: Federate **1**.

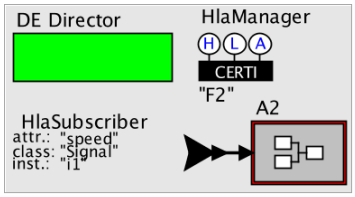


Figure 8: Federate **2**.

4.1.1 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. 1 and the information in table 1.

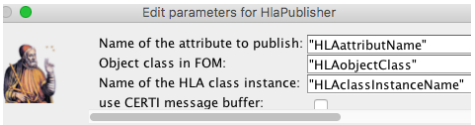
Federate **1** (publishes `Signal.speed`):

1. Create a new model[4] 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[4],
4. Drag an `HlaManager` icon and an `HlaPublisher` icon; connect the latter to the output port of **A1**. For configuring the icons, see sections 4.1.2 and 4.1.4. The final result is depicted in fig. 7.

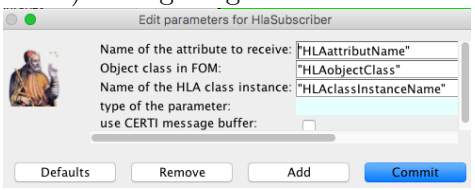
Federate **2** (subscribes `Signal.speed`):

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 4.1.3 and 4.1.4. The final result is depicted in fig. 8.

[4]For creating Ptolemy models, see chapter *Building Graphical Models* in [1].



a) Configuring `HlaPublisher`



b) Configuring `HlaSubscriber`

Figure 9: Configuring actors.

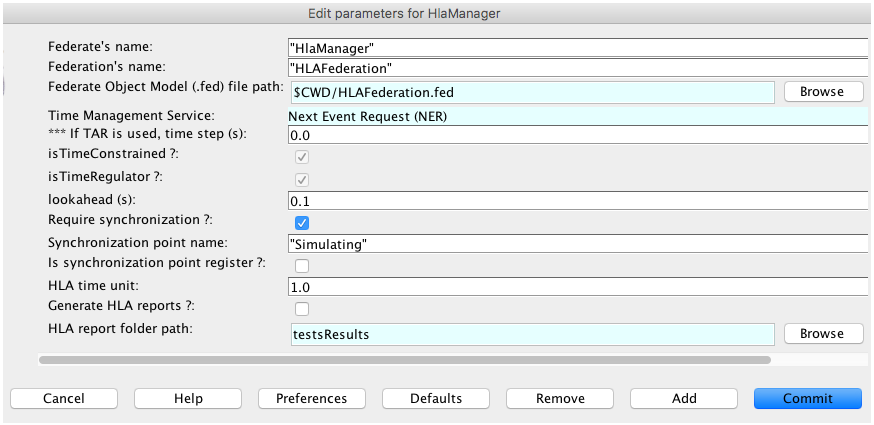


Figure 10: Configuring `HlaManager`

4.1.2 Configuring `HlaPublisher`

In the model `Federate1.xml` (fig. 7), double-click on the icon `HlaPublisher`; the window depicted in fig. 9.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 4.2. Check the data type of output port of `A1` (it is `double`); keep unset the field `use CERTI message buffer` (see sidebar on page 7). Click `Commit`.

Sidebar: data type of an attribute
For ensuring whether the data type of an attribute is the same in the `HlaSubscriber` and the corresponding `HlaPublisher`, check the following points:
– First of all, check if the class and the attribute are defined in the FOM (.fed file).
– The field `type of the parameter` in the `HlaSubscriber` actor, must have the *same* type as the one in the input of the `HlaPublisher` actor that publishes the corresponding class instance.
– Check if the type of the `HlaSubscriber` and the input port of the actor it is connected to are the same.
– The field `use CERTI message buffer` must be the same for both actors (set or unset for both). By default, they are unset, but you may need to set if another federate, e.g., a C++ federate has it set.
See errors 7.1.7 and 7.1.10 for actors in a same model, and errors 7.1.2 and 7.1.3 for actors in different federates (models).

4.1.3 Configuring `HlaSubscriber`

In the model `Federate2.xml` (fig. 8), double-click on the icon `HlaSubscriber`; the window depicted in fig. 10.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`. Put `double` in the field `type of the parameter`; keep the field `use CERTI message buffer` unset. Check if the data type of `A2` input is also `double` (see sidebar on page 7). Click `Commit`.

4.1.4 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.*

Federate **1**:

1. In the model `Federate1.xml` (fig. 7), double-click on the icon `HlaManager`; the window depicted in fig. 9.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.
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**:

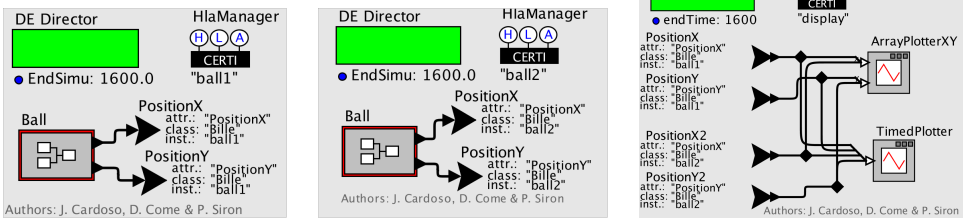
1. In the model `Federate2.xml` (fig. 8), double-click on the icon `HlaManager`; the window depicted in fig. 9.a pops out.
2. In the field `Federate's name`, replace the default `HlaManager` by `F2`.
3. In the field `Federation's name`, replace the default `HlaFederation` by the same name, `Test`.
4. 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?`; put the same name used in the `HlaManager` of `Federate1.xml`.
8. Tick the field `Is synchronization point register?`. This model will be the last to be launch.
9. You may tick `Generate HLA reports?`.

4.2 Using multiple instances of a class

The Ptolemy-HLA framework the ability to manage several instances of a class (e.g., several UAV flying in fleet). You just need to match the name of the instance in the federate that registers (and publishes) the instance and the one that discovers (and subscribes to) it. If the federate that subscribes does not need to know the name of instance, a joker (see section 6) can be used (instance name in `HlaSubscriber` must be `joker_i`).

Billiard Federation: `$PTII/org/hlacerti/demo/Billard/B/Billard.xml`
A quite simple example is a billard simulation. A first version, called `2Billes2Fed` is represented in fig. 11.

The federate `ball1`, depicted in fig. 11.a publishes `positionX` and positionY of class `Bille`; the instance is named also `ball1` (could be a different name). The federate `ball2`, depicted in fig. 11.b publishes `Bille.positionX` and Bille.positionY; the instance is named `ball2`. The federate `display` subscribes to `Bille.positionX` and Bille.positionY. For each attribute of an instance – i.e., for each `HlaPublisher` in each federate – there must be an `HlaSubscriber` as depicted in fig. 11.c. The FOM is represented in fig. 12. The (X,Y) coordinates of the two balls are represented in fig. 13. Table 3 shows the attributes of class `Bille` published and subscribed by `ball1` and `ball2` federates; `display` federate subscribes for all instances.



a) `ball1` federate b) `ball2` federate c) `display` federate

Figure 11: A federation exchanging attributes `positionX, positionY`.

Attrib.	Inst.	Fed. `ball1`	Fed. `ball2`
PosX	ball1	Publishes	Subscribes
PosY	ball1	Publishes	Subscribes
PosX	ball2	Subscribes	Publishes
PosY	ball2	Subscribes	Publishes

Table 3: Publish/Subscribe table.

A second version of a billard simulation is called `2Billes1Fed`. This federation has two federates: `display` (fig. 11.c) and `billiard`, represented in fig. 14. The latter publishes two instances of class `Bille`: `ball1` and `ball2`. The simulation is represented by the same figure 13.

```
;; 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
```

Figure 12: Test.fed (FOM).

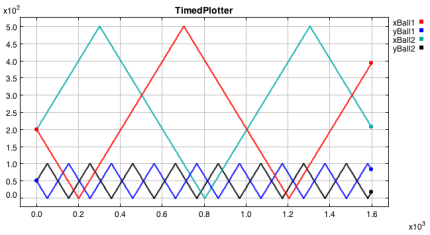


Figure 13: Time Plotter.

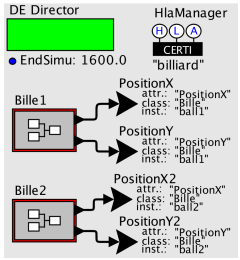


Figure 14: Federate `billiard`.

F14 Federation: `$PTII/org/hlacerti/demo/f14HLA/TwoF14AircraftsNER/TwoF14AircraftsNER.xml` Another demo provided in the framework using multi-instances is the one with two F14 aircrafts flying together. They do not exchange data, but it shows the HLA (and CERTI implementation) ability to correctly register/discover and send/receive data when there are multiple instances of a same class. In this example, each aircraft i is itself modeled by three federates `AutoPilot`, `Aircraft` and `PilotStick`. The classes and attributes used in this federation are described in the FOM represented in the fig. 5. When the federation simulates one aircraft, the publish/subscribe table represented in Table 2 is enough. But if there are two aircrafts, then it is necessary to add which instance is published/subscribed, as represented in Table 4.

Attribute	Type	Inst	`AutoPilot`	`Aircraft`	`PilotStick`	`AutoPilot2`	`Aircraft2`	`PilotStick2`
`AIRCRAFT.alpha`	double	ac1	Subscribes	Publishes	–	–	–	–
`AIRCRAFT.q`	double	ac1	Subscribes	Publishes	–	–	–	–
`ACTUATOR.elevCom`	double	ap	Publishes	Subscribes	–	–	–	–
`JOYSTICK.ELEVATOR`	int	elev	Subscribes	–	Publishes	–	–	–
`AIRCRAFT.alpha`	double	ac2	–	–	–	Subscribes	Publishes	–
`AIRCRAFT.q`	double	ac2	–	–	–	Subscribes	Publishes	–
`ACTUATOR.elevCom`	double	ap2	–	–	–	Publishes	Subscribes	–
`JOYSTICK.ELEVATOR`	int	elev2	–	–	–	Subscribes	–	Publishes

Table 4: Publish/Subscribe table for F14 federation with two aircrafts.

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 follow the steps in section A.

6 FAQ

6.1 What is a joker when using multi-instances?

A federate instantiates only the object instances it wants to use/receive, by using an HlaSubscriber. The order in which the instances will be discovered is not known before the run. If different object instances must be connected to different actors in the model, then the name of the instance must be the same in the HlaSubscriber and the (corresponding) HlaPublisher. For example, if the instance `UAV1` is the guide, and the instance `UAV2` is the follower, then this information must be known by the federate that calculates the control of both instances.

But when this information is not needed, the federate that subscribes to these instances needs, at least, to do not mix the attributes of both instances. For example, the (x,y,z) coordinates of each one UAV need to be "kept" together. In this case, we can use $\texttt{joker}_1$ and $\texttt{joker}_2$. We do not know which instance will be discovered by $\texttt{joker}_1$, and if the joker was chosen, is because this is not important! But $\texttt{joker}_2$ will discover the second instance.

7 Error Messages

Some commun errors are reported as tests that raise exception. As for the moment, this tests are in `org/hlacerti/test/testWithException`. Examples of the right way to use the Ptolemy-HLA environnement are in `org/hlacerti/test/auto`.

7.1 Known solutions

The errors bellow had appear in the cited situations. If an error appears in a new situation, please report to cardoso@isae.fr.

7.1.1 RTIinternalError

When: After the federate is launched
Actor highlighted: HlaManager icon of the federate
Message:

```
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 CertiRtig. One way to debug this
is to set the various environment variables by sourcing certi/share/scripts/myCERTI_env.sh, then
invoking rtig on the .fed file then (re)running the model.
  in .MixedSimulatorPlant.HlaManager
Because:
Connection to RTIA failed. null
```

Reason: Unknown.
Solution 1: Source $CERTI_HOME/share/scripts/myCERTI_env.sh as suggested; see section A.1.
Solution 2: Just launch again the federate. You can kill the rtig before.

7.1.2 RTIinternalError: 4

When: After the last federate is launched (i.e., the one that is the Register of the Synchronization Point).
Actor highlighted: HlaManager of a federate (in the message bellow, FedPlant).
Message:

```
ptolemy.kernel.util.IllegalActionException: RTIinternalError: 4
  in .FedPlant.HlaManager
Because:
4 at org.hlacerti.lib.HlaManager.proposeTime(HlaManager.java:916)
```

Possible Reason: A problem with data type, as described in sidebar on page 7. It may appear when receiving a RAV callback (inside a time advance loop called by proposeTime).
Solution: Carefully check the data type of each HlaPublisher input and the data type of the corresponding HlaSubscribers.

7.1.3 RTIinternalError: null

When: After the last federate is launched (i.e., the one that is the Register of the Synchronization Point).
Actor highlighted: HlaManager of a federate (FedDisplay in the example below)
Message:

```
ptolemy.kernel.util.IllegalActionException: RTIinternalError: null  in .FedDisplay.HlaManager
Because:  hla.rti.RTIinternalError serial:0
at org.hlacerti.lib.HlaManager.proposeTime(HlaManager.java:916)
```

Possible Reason: All federates do not have the save value for "certi message buffer" in their HlaSubscriber and HlaPublisher (see sidebar on page 7). It may appear when receiving a RAV callback (inside a time advance loop called by proposeTime).
Solution: Tick (or untick) "certi message buffer" in all HlaSubscriber and HlaPublisher actors.

7.1.4 Federate name already in use

When: After the federate is launched
Actor highlighted: HlaManager of this federate
Message:

```
ptolemy.kernel.util.IllegalActionException: RTIexception: 4.9.5.b : Federate name already in use.
  in .MixedSimulatorControl.HlaManager
Because:
4.9.5.b : Federate name already in use.
at org.hlacerti.lib.HlaManager.initialize(HlaManager.java:697)
```

Possible reasons:
1) there is another federate with the same name (see the HlaManager). *Solution*: rename one of the federates
2) A previous federate was launched but the federation was not properly terminated. *Solution*: kill the rtig and launch again the federation with all federates.

7.1.5 CouldNotOpenFED

When: After the federate is launched
Actor highlighted: HlaManager icon of the federate
Message:

```
ptolemy.kernel.util.IllegalActionException: CouldNotOpenFED: Module not found
in .FedQuadControlTEST.HlaManager
Because:
Module not found
at org.hlacerti.lib.HlaManager.initialize(HlaManager.java:671)
at ptolemy.actor.CompositeActor.initialize(CompositeActor.java:912)
```

Reason: Wrong name of the .fed file, or wrong address.
Solution: Just browse the right .fed file in the HlaManager icon.
Remark: When creating a new model, put the .fed file in the same directory that the federate model.

7.1.6 ErrorReadingFED

When: After the federate is launched
Message:

```
ptolemy.kernel.util.IllegalActionException: ErrorReadingFED: fed parser found error in FED file
  in .MixedSimulator2.HlaManager
Because:
fed parser found error in FED file
at org.hlacerti.lib.HlaManager.initialize(HlaManager.java:673)
```

Reason: there is an error in the .fed file.
Solution: check carefully the .fed file. Commun error: number of open and closed parenthesis is not equal.

7.1.7 Types resolved to unacceptable types in... due to the following objects

When: After the federate is launched
Actor highlighted: HlaSubscriber and HlaManager
Message:

```
ptolemy.actor.TypeConflictException: Types resolved to unacceptable types in .FedQuadControl due to
the following objects:
(port .FedQuadControl.Control.PositionControl.ControlForZ.Scale2.input: unknown)
```

Reason: The data type of the HlaSubscriber actor and the corresponding HlaPublisher do not correspond (in fact, the data type of the output port of the actor connected to the HlaPublisher). See sidebar on page 7 of this manual.
Solution: Choose the same data type for a same attribute `Class.Attribute` (in the HlaSubscriber and for the output port of the actor connected to the HlaPublisher). A good practice is construct a table as the one in fig. 4.

7.1.8 AttributeNotOwned

When: the federation starts the execution but hangs, and nothing appears in the plotters
Actor highlighted red: HlaManager
Message:

```
ptolemy.kernel.util.IllegalActionException: AttributeNotOwned:
  in .BillardHitBall2.HlaManager
Because:
at org.hlacerti.lib.HlaManager.updateHlaAttribute(HlaManager.java:1060)
at org.hlacerti.lib.HlaPublisher.fire(HlaPublisher.java:235)
```

Where in the code: Exception raised when calling `updateAttributeValues`
Possible reason: The HlaPublisher has an incorrect parameter or the FOM is not coherent.
Solution: Put the correct values for class-attribute-instance in the HlaPublisher.
What also worked:
Possible reason: the federate was renamed but in the xml file the name is still the old one.
Solution: edit the xml file where (the red HlaManager is) with the desired name.

7.1.9 ObjectAlreadyRegistered

When: After the federate is launched
Message:

```
ptolemy.kernel.util.IllegalActionException: ObjectAlreadyRegistered:
Because:
at org.hlacerti.lib.HlaManager$PtolemyFederateAmbassadorInner._setupHlaPublishers(HlaManager.java:2977)}
```

Possible reasons:

1) A problem of ownership. A same instance of object is registered for two different federates. It can be just un error when editing the parameters of an HlaPublisher/HlaSubscriber, a bad choice of the instance names, or it can be a FOM that is wrongly defined. Example: A federate F1.xml publishes `class.attribute1.i1` (subscribed by another federate F2.xml), and F2.xml publishes another attribute of the same class, `class.attribute2.i2` (subscribed by another federate F1.xml). Then the instance names i1 and i2 must be different.

Solution: Check if the HlaSubscribers and HlaPublishers are coherent with the FOM. And/or check if the FOM itself is coherent.

2) A previous federate was launched but the federation was not properly terminated.
Solution: Kill the rtig and launch again the federation with all federates.

3) There is another federate with the same name (see the HlaManager).
Solution: rename one of the federates.

7.1.10 A HlaSubscriber X with the same HLA information specified by the HlaSubscriber Y is already registered

When: After a federate is launched
Actor highlighted red: HlaManager
Message:

```
ptolemy.kernel.util.IllegalActionException: A HlaSubscriber '.Fed1.HlaSubscriber4' with the same HLA
information specified by the HlaSubscriber '.Fed1.HlaSubscriber'  is already registered for subscription.
  in .Fed1
at org.hlacerti.lib.HlaManager._populateHlaAttributeTables(HlaManager.java:1713)
```

Reason: Two HlaSubscribers have the same value for (at least) one of the parameters `Class.Attribute.InstanceName`. *Solution*: One of the values `Class.Attribute.InstanceName` must be changed. Check the possible values in the FOM and also the data exchanged between the simulators (federates). Remark: HlaSubscribers `C1.att1.i1` and `C2.att2.i` must have i $\neq$ i1.

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] https://savannah.nongnu.org/bugs/?group=certi.

[6] 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.*

[7] 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.

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

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

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

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

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

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

[14] Kuhl, Frederick and Dahmann, Judith and Weatherly, Richard, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, Prentice Hall PTR, ISBN 978-0-13-022511-5, 2000.

[15] Clement Michel, Janette Cardoso, and Pierre Siron. *Time management of heterogeneous distributed simulation.* In 31st European Simulation and Modelling Conference, October 2017.

Appendices

A Installing CERTI

CERTI is an Open Source HLA RTI (GPL, libraries are LGPL) [13]. It supports HLA 1.3 specifications (C++ and Java) and partial IEEE 1516-v2000 and IEEE 1516-v2010 (C++). On April, 2018, the last released version is 3.5.1 and the beta version, that will be released soon, is the 4.0.0. You can check if there is any bug related to CERTI in https://savannah.nongnu.org/bugs/?group=certi.

You only need to read the following sections if you want to install another version of CERTI or if the script in $PTII/org/hlacerti/build-certi.sh did not worked.

For installing CERTI 3.5.1, you can find the instructions here:
$PTII/org/hlacerti/models/legacy/DynamicMultiInstance/manual-ptii-hla.pdf

For installing CERTI 4.0.0, you can follow the steps bellow (if the script for automatically launching CERTI presented in section 5.2 did not work).
– You need the following softwares: cmake (or ccmake), flex and bison
– If you already created the folder `$HOME/pthla`, skip step 1 below.
– You can use the Null Prime Message Protocol for speed-up the simulation by setting `$CMAKE_FLAGS` before step 8 bellow:

```
export CMAKE_FLAGS =  -DCERTI_USE_NULL_PRIME_MESSAGE_PROTOCOL=ON
```

A.1 Installing CERTI on Linux

```
 1.  mkdir $HOME/pthla
 2.  cd $HOME/pthla
 3.  git clone -b br_jbch_4.0.0 https://git.savannah.nongnu.org/git/certi.git
 4.  mkdir $HOME/pthla/certi-tools
 5.  cd $HOME/pthla/certi
 6.  mkdir $HOME/pthla/certi/build-certi
 7.  cd $HOME/pthla/certi/build-certi
 8.  cmake -DCMAKE_INSTALL_PREFIX=$HOME/pthla/certi-tools  $CMAKE_FLAGS ../
 9.  make
10.  make install
```

For setting all the environment variables you need for running an HLA federation, choose one of these ways:

1. Put in your $HOME/.bash_profile file the following command:
   `source $HOME/pthla/certi-tools/share/scripts/myCERTI_env.sh`
   In a terminal, execute the command: `source $HOME/.bash_profile`.

2. Put in your $HOME/.bash_profile file the following command:
   `alias certiConfig ="$HOME/pthla/certi-tools/share/scripts/myCERTI_env.sh"`
   Each time that you need to run an HLA federation, type `certiConfig` in each terminal where you will call `$PTII/bin/vergil`.

3. In each terminal where you will call `$PTII/bin/vergil`, execute the command:
   `source $HOME/pthla/certi-tools/share/scripts/myCERTI_env.sh`

Your `$CERTI_HOME` is now `source $HOME/pthla/certi-tools`.

A.2 Installing CERTI on Windows

TBD

A.3 Installing CERTI on MacOS

Same instructions than for Linux (section A.1) but there is an issue with MacOS since El Capitan. Even if CERTI and Ptolemy are successfully 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) after the last federate has been launched and the following error message:

```
_read(): dyld: Library not loaded: libCERTId.4.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 symbolic link for file libCERTId.4.dylib, do the following commands:

```
cd $HOME/pthla/certi-tools/lib
mv libCERTId.4.dylib foo-libCERTId.4.dylib
cp libCERTId.4.0.0.dylib  libCERTId.4.dylib
```

3. Then, because of problem with shared libraries, to the following:

```
sudo -i
cd /usr/local/lib
ln -s /Users/.../pthla/certi-tools/lib/*.    .
```

4. Open a new terminal, and run a demo again:

```
$PTII/bin/vergil $PTII/org/hlacerti/demo/2Billes2Fed/2Billes2Fed.xml &
```

B Testing CERTI

To test the installation, run the C++ billiard demo; the code can be found in $HOME/pthla/certi/test/Billard:

```
1. Open 3 terminals (make sure you open new terminals or source ~/.bash\_profile
     in the already open terminals)
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 Ptolemy billiard demo is in $PTII/org/hlacerti/demo/Billard/ can interact with the C++ billiard demo.

C 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 an 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 a values for Lookahead and HLA time unit (usually keep the default value 1). 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. as in figure 14), put a different instance name in the field `class instance`. Choose the good data type in its input port.
5. If the Federate will receive values (of an attribute) from other federate(s), add a `HLASubscriber` for each attribute (in the FOM) and for each instance the Federate will subscribe. Carefully put the same instance name used in the (corresponding) `HLAPublisher` of the Federate publishing this instance. Choose the same data type as in the input of this `HLAPublisher`. If the name of the instance is not useful in the model where the data is received, you can use a joker. Important: for each instance registered by some federate, use a different joker.

D Previous revisions

Some important changes were made at revision 71890 (for allowing multiple instances of an object), 72233 (where instances can be dynamically discovered), 72943 (TAR mechanism for time management) and r74969 (print of information about the execution in .csv files). Revisions 71890 and 72233 broke backward compatibility, because they changed the way instances are considered: mono-instances, static multi-instances discovering and dynamic multi-instance discovering. All these revisions have demos in `$PTII/org/hlacert/models/legacy`.

- 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.
- 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 [9].
- Revision 72943: TAR time management was implemented by Yanxuan LI [10]; new demo SimpleProducerMultipleC TAR added; variables are renamed for clarity (r73341).

- Revision r73687: Some cleanup of demo layout and minor tuning of icons.
- r74969 (2016-07-25): Performance measures and simulation validation are added to the framework by Tarciana Guerra [11]: simulation data, simulation results (e.g., events in the Ptolemy calendar queue and events coming from the RTI); and simulation statistics (e.g., number of TARs/NERs). These information appear in .csv text files generated during the simulation.
- r75771 (2017-03-08): Function _roundDoubles removed in org.hlacerti.lib.HlaManager when used for rounding Ptolemy time (currentTime) and Certi time (CertiLogicalTime) in time advance requests. This was the raison of the error described on section B.11. See [12].
- r75720 (2017-02-17): Updating jcerti.lib. Now federates can be launched in another computer without problem.
- r75773 (2017-03-16): Some errors fixed in org.hlacerti.lib.HlaManager (related to .csv files). Unfortunately several errors were introduced in previous versions.
- r76350 (2017-07-08): demos are added in $PTII/ptolemy/org/hlacerti/demo/MicrostepReset for showing a non-expected behavior of microstep for some particular cases (output events of DiscreteClock actor and HlaSubscriber have the same timestamp and microstep but are not added). Explanation of the issue in Clement Michel report [12].