Ptolemy-HLA Framework User Manual V0.1

Edward Lee & Janette Cardoso, April 2019

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 [16]), 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 [7, 9, 10, 11, 12, 13].

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.

[1]Contributors implied in this framework from the beginning in 2013 up to now, April, 2019 (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.

Class	Attribute	Semantics	Unit
Aircraft	speed	Measure of the aerodynamic speed	m/s

Table 1: FOM for federation Fleet

- Installing the HLA-PTII co-simulation (section 6): which softwares you need and how to install Ptolemy and CERTI.
- There are also FAQ (section 5), Error Messages (section 7) and some information about previous versions of this framework (section F).

2 Ptolemy-HLA federation

2.1 Some basics about HLA standard

The IEEE High-Level Architecture (HLA) standard [4] 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 [7].

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 [9]:

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 Fleet)
  (fedVersion v1.3)
  (spaces)
  (objects
    (class ObjectRoot
      (attribute privilegeToDelete reliable timestamp)
      (class RTIprivate)
      (class Aircraft
        (attribute speed reliable timestamp)  )))
  (interactions)
)
```

Figure 1: FED file.

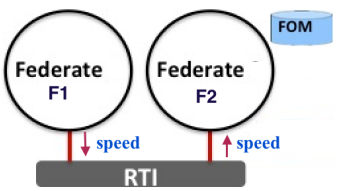


Figure 2: HLA architecture.

Sidebar: Federation Object Model (FOM) and Federation Execution Data (FED)
Every HLA Federation has a Federation Object Model (FOM) that introduces all shared information

Attribute	Fed. **F1**	Fed. **F2**
`Aircraft.speed`	Publishes	Subscribes

Table 2: Publish/Subscribe table.

and contemplates inter-federate issues as data encoding schemes [3], semantics of the attribute and its unit. The FOM in table 1 defines a class named "Aircraft" with a single attribute "speed" which unit is m/s. An instance of the class Aircraft contains data about that aircraft, in this case only its speed, that is shared among federates. For example, in figure 2, one federate updates the speed, and other observes these updates.

A subset of a FOM along with the specification of some default values for Ordering and Transport properties of data [3] is put in a Federation Execution Data (FED) file (norm US DoD 1.3, with extension .fed) or FDD file (a xml file in norm IEEE 1516). The FED file is used by the RTI during the simulation. Nothing in the FED file is case sensitive. The example FED file in figure 1, in addition to boilerplate text that is required to be in every FED file, exhibes the information of the FOM document, without the units or data types: it names the federation "Fleet" and the class "Aircraft" with its attribute "speed". The keyword **timestamp** indicates that the delivery of events must be ordered in accordance with logical time, and **reliable** indicates that quality of service is using TCP. The keyword **fedVersion v1.3** indicates that it uses the norm US DoD 1.3.

HLA supports two distinct mechanisms for federates to communicate, shared **objects** and **interactions**. Ptolemy II uses only shared objects, each of which is an instance of a class defined in the FOM. Each communication sent by an HlaPublisher actor and received by an HlaSubscriber actor is an **attribute** of such an object. The name of the attribute, as well its units and type is defined in the FOM.

Figure 2 defines a federation named "Fleet". Class definitions appear within the "objects" part of the FED file. This federation uses only one class, Aircraft, with only one attribute, speed. All classes are subclasses of **ObjectRoot**, and hence are nested within ObjectRoot in the FED file. ObjectRoot has an attribute **privilegeToDelete** which indicates ownership of an object. In Ptolemy-HLA, the HlaPublisher will assert ownership of any objects it creates.

A valid FED file is required to contain several items that are actually irrelevant to the usage in Ptolemy II. The "spaces" section allows to define multiple *routing spaces* defined by some *dimension*, e.g., longitude, latitude, altitude [3, 17]. The **RTIprivate** class is required in every FED file. The interactions section at the bottom defines communications between federates that are not via objects. Ptolemy does not use any such interactions, so this section will be ignored by any federate implemented in Ptolemy.

See [17] for more details.

2.1.1 Data Exchange in HLA

The data in HLA is described by the FOM with also the types of attributes and parameters, but the FED file does not, because the RTI has no notion of their type [17]. If the delivery of events must be ordered in accordance with logical time, the keyword `timestamp` must be used (see fig. 1). Otherwise, events are delivered in receive order. The quality of service can also be chosen in HLA standard, best-effort (UDP or IP) or reliable (TCP). We are using `reliable`.

The *attribute* `speed` belongs to a *class* called `Aircraft`, and is `reliable timestamp`, as described in the FOM represented in table 1. Let us consider the federation in fig. 2 with two federates: Federate **F2** uses the data `speed` provided by Federate **F1**, i.e., Federate **F1** publishes the class `Aircraft` and Federate **F2** subscribes to attribute `speed` of this class. This information is usually provided in a table as the one in table 2. There are two steps concerning the *object management* [7]:

1. When federate **F1** is launched, it registers an object instance of `Aircraft` class. When federate **F2** is

launched, it discovers object instances `Aircraft` related to the attribute `speed` it subscribed.

2. During the simulation, federate **F1** sends through the RTI a new value of `Aircraft.speed` using the service `updateAttributeValues` (UAV). The RTI sends this value to federate **F2** 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 4.3 describes two different ways of implement a Billard federation.

2.1.2 Time Advance in HLA

The Ptolemy-HLA framework is intended to model and simulate cyber-physical systems where the coordination of time is an important issue. Even if the HLA standard allows a distributed simulation where time can not be taken into account, Ptolemy federates are both time constrained and time regulating [7]. The framework can be easily extended to consider more simple federates.

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:

1. the `timeAdvanceRequest` service (TAR), used to implement time-stepped federates; and

2. the `nextEventRequest` service (NER), used to implement event-based federates.

For more information about time advance, a very important point, see [7].

2.1.3 Execution of a HLA federation

The RTI services allow to execute a distributed simulation in a coordinated way. To make it short, for guaranteeing the reproducibility of a federation execution, before a federate begins to advance its time, it must be guaranteed that all federates have joined and have set their time switches. The *synchronization point* allows to synchronize the federation execution at the beginning of a phase, using the appropriated RTI services managed by the HlaManager.java code. In the Ptolemy-HLA framework, the choice was to set the register of the synchronization point in the last federate launched. This parameter is set in the `HlaManager` icon. Other designers use a federate just for deal with the synchronization (see appendix B).

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 [9, 7] for more technical information.

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

- `HlaManager`: This interface handles: 1) The time coordination between the Ptolemy logical time and HLA logical time, 2) The data exchange between federates (with `HlaSubscriber` and `HlaPublisher`). The architecture of the HlaManager is partially depicted in figure 26, appendix A.

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

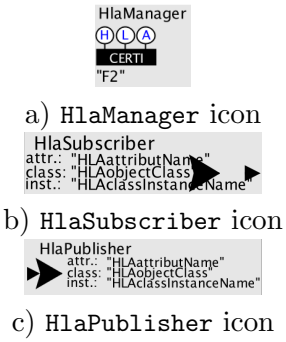


Figure 3: Icons.

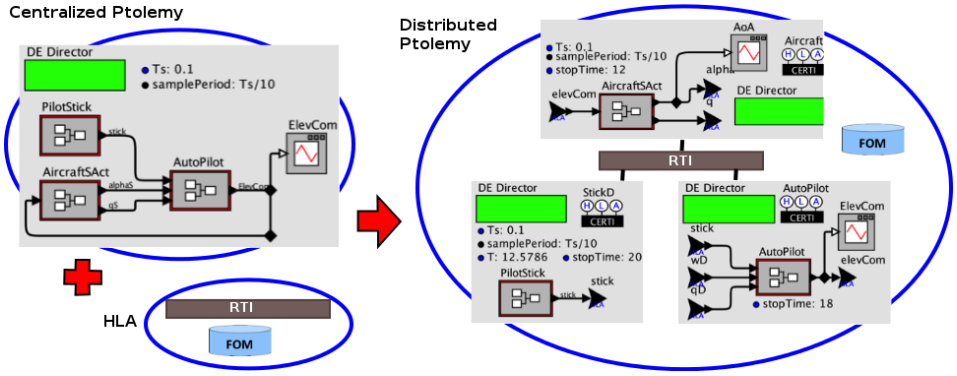


Figure 4: A Ptolemy-HLA federation from a centralized Ptolemy model.

```
(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: FED for F14 federation.

- `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 sidebar 2.1) whose FED file is represented in fig. 5, construct a publish/subscribe table as the one in Table 3 and ... a RTI.

3 Running a Ptolemy-HLA Federation Demo

3.1 Ptolemy and the RTIG

There are two ways to run a federation in the Ptolemy-HLA environnement:

1. Open a terminal, and launch the rtig (*by hand*) in the directory where the Ptolemy federation is located. Open another terminal in the same directory, and launch vergil. You may need to kill the rtig before launch another federation (with a different FED file).

2. If the federation models are in your local file system, and all environnement variables are set (PTII, CERTI_HOME) the first launched federate launches the rtig *automatically*. This federate also kill the rtig after all federates have resigned the federation.

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

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

Table 3: Publish/Subscribe table for F14 federation.

If you have already installed CERTI and Ptolemy, follow the steps bellow in the *order* presented bellow. Otherwise, see section 6. The demos are in `$PTII/org/hlacerti/demo`.

3.2 Setting environnement variables

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`

3.3 Running a demo

There are several demos in `$PTII/org/hlacerti/demo`. A particular demo is 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 is the register of the synchronization point. This one is the last one to be launched. Read the HlaManager javadoc.

```
cd PTII/org/hlacerti/demo
PTII/bin/vergil demo-name/demo-name.xml  &
```

Important remark: The RTIG creates a log file in the directory where it is launched. If you do not have the writing rights in this directory, then copy the demo folder in your file system.

These are the demos in the current framework:

Fleet This federation has two federates F1 and F2 exchanging data. See section 2.1.1 and 4.2.

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 [7].
`$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 &`

[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.

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:
– 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 &`
See also **BillardJS** where the balls are simulated using JavaScript code (Utilities/JavaScript/JavaScript).

4 Getting Started

4.1 Ptolemy-HLA library

The three new components of the framework can be found in `MoreLibrairies->Co-Simulation->HLA`. [4].

Two new actors were created, HlaAttributeUpdater and HlaAttributeReflector that will extend the current HlaPublisher and HlaSubscriber. They are used already in some models, as the Fleet federation in section 4.2. If they are not in the library, open the Fleet federates and copy these actors in your model. You can also save them in your own library, see `https://ptolemy.berkeley.edu/ptolemyII/ptII10.0/ptII10.0.1/doc/coding/addinganactor.htm`.

Read the documentation by doing right-click on the actor, then Documentation/Get documentation.

4.2 Creating federates of federation Fleet

If you have already installed Ptolemy, follow the steps bellow. Otherwise, see section 6.1.

Let us consider the (centralized) Ptolemy model depicted in figure 6. We want to create a federation called **Fleet** with two federates. The composite actor **A1** will be implemented in Federate **F1** and **A2** will be implemented in Federate **F2**. 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.

This federation can be executed by doing:

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

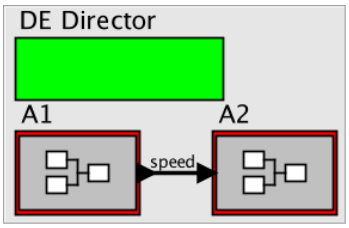


Figure 6: A Ptolemy model.

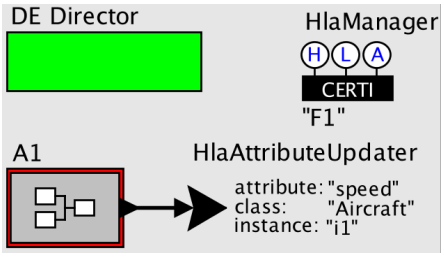


Figure 7: Federate **F1**.

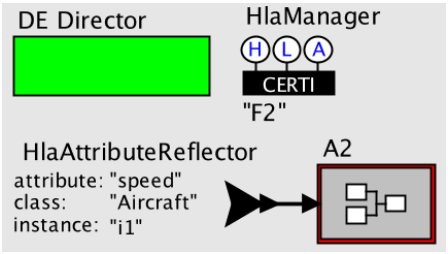


Figure 8: Federate **F2**.

4.2.1 Defining a FOM

Before create a federation, you need to define the federation execution data, or FED [17], that contains the extract of the FOM (Federation Object Model). We will use for this example the FOM defined in table 1.

4.2.2 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 FED depicted in fig. 1 and the information in table 2.

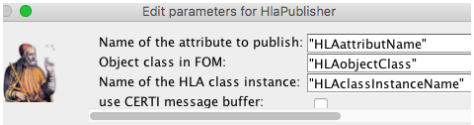
[4] For creating Ptolemy models, see chapter *Building Graphical Models* in [2].

Federate **F1** (publishes `Aircraft.speed`):

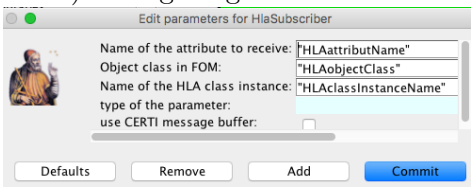
1. Create a new model[4] in the folder, populate with a DE director (mandatory); save it, e.g., as `Federate1.xml`,
2. Copy the composite actor **A1** from the centralized model,
3. Check if the output port of **A1** has a type; if not, choose a type[4],
4. Drag an `HlaManager` icon and an `HlaPublisher` icon; connect the latter to the output port of **A1**. For configuring the icons, see sections 4.2.3 and 4.2.5. The final result is depicted in fig. 7.

Federate **F2** (subscribes `Aircraft.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 4.2.4 and 4.2.5. The final result is depicted in fig. 8.



a) Configuring `HlaPublisher`



b) Configuring `HlaSubscriber`

Figure 9: Configuring actors.

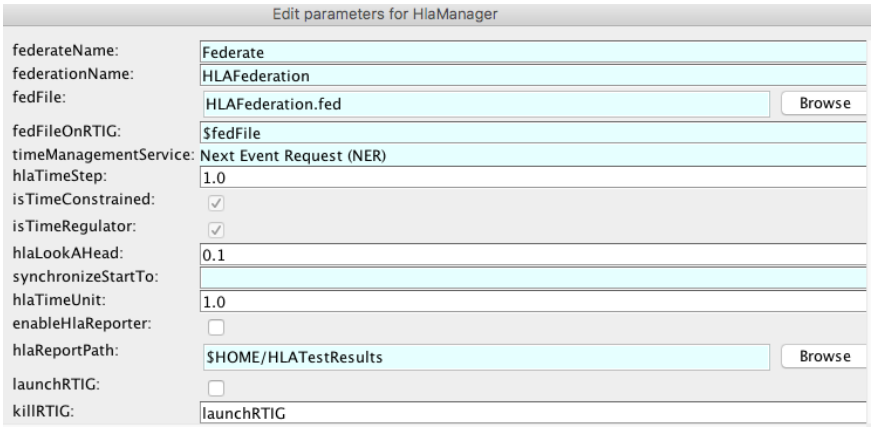


Figure 10: Configuring `HlaManager`

4.2.3 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 `Aircraft`. 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 4.3. Check the data type of output port of `A1` (it is `double`); keep unset the field `use CERTI message buffer` (see sidebar on page 9). 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 7.1.8 and 7.1.11 for actors in a same model, and errors 7.1.3 and 7.1.4 for actors in different federates (models).

4.2.4 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 `Aircraft`. 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 9). Click `Commit`.

4.2.5 Configuring `HlaManager` of all federates in a Federation

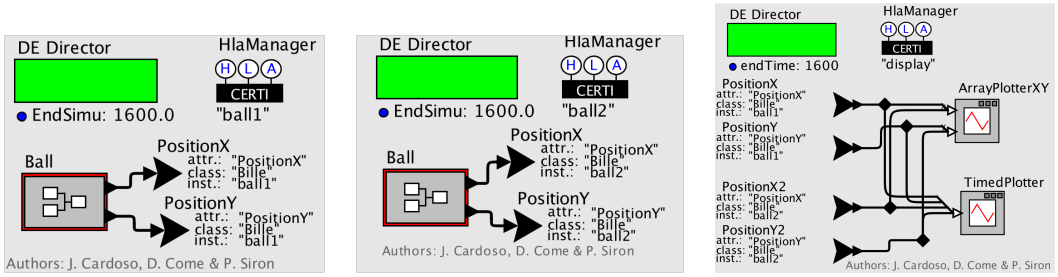
Some important points to have in mind:

1. All federates must use the same Federation name. Some `.fed` files have a federation name, but this name is not checked anywhere.
2. If the RTIG must be launched automatically, then the first federate to be launched must set the parameter launchRTIG.
3. The last federate (Ptolemy model) to be launched must be the *register of the synchronization point*. The parameter launchRTIG must be unset.
4. All federates must use the same `Synchronization Point Name`. This name is the name of the last federate to be launched.
5. Each federate can choose its own time management, NER or TAR.
6. Each federate can choose to save its execution in .csv files in `$HOME/HLATestsResults` by ticking the parameter enableHlaReporter. Another folder can be browsed.

Federate **F1**: the first one to be launched

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 `Fleet`.
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. This federate was chosen to be the first one to be launched: tick the field launchRTIG.
8. Put the name of the federate F2 in the field SynchronizeStartTo.
9. You may tick `enableHlaReporter`.

Federate **F2**: the last one to be launched



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

Figure 11: A federation exchanging attributes `positionX, positionY` of class `Bille`

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, `Fleet`.
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. Put the name of the federate F2 in the field SynchronizeStartTo.
8. You may tick `enableHlaReporter`.

4.3 Using different 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 5) 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 4 shows the attributes of class `Bille` published and subscribed by `ball1` and `ball2` federates; `display` federate subscribes for all instances.

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.
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

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

Table 4: Publish/Subscribe table.

```
;; 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.

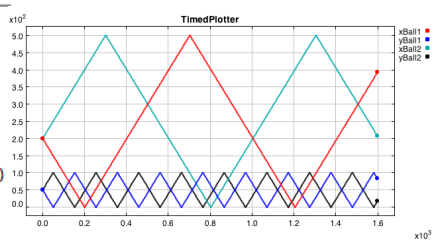


Figure 13: Time Plotter.

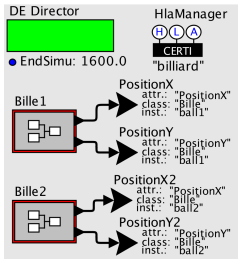


Figure 14: Federate `billiard`.

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 3 is enough. But if there are two aircrafts, then it is necessary to add which instance is published/subscribed, as represented in Table 5.

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

4.4 Using wildcard with different instances of a class

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 `joker_1`and `joker_2`. We do not know which instance will be discovered by `joker_1`, and if the joker was chosen, is because this is not important! But `joker_2`will discover the second instance.

However, the wildcard must be used with caution. Figure 17 shows the federate Display. The FOM has a unique class, `Vehicle`, and three attributes `x, y,` and `z`. Federate Display shows the coordinates of two instances of class `Vehicle` in the two plotters called Avehicle and AnotherVehicle. It is not important to

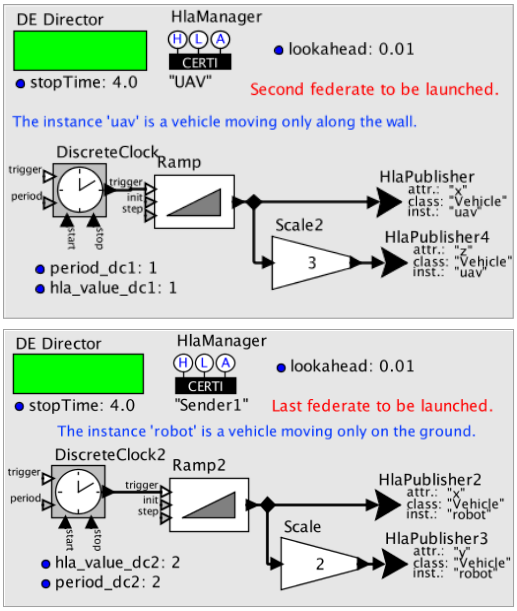


Figure 15: Two federates: UAV and Sender1.

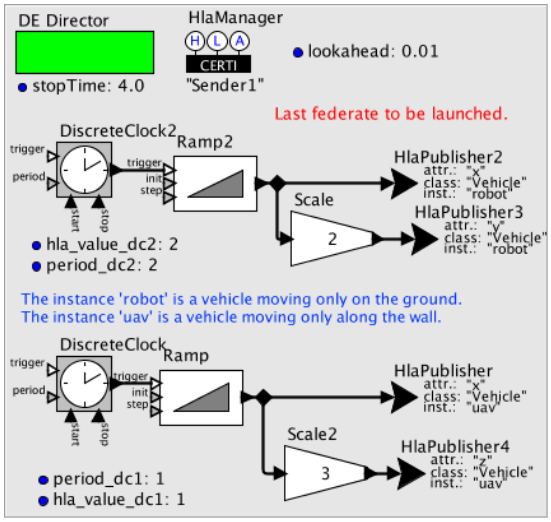


Figure 16: A federate with two instances.

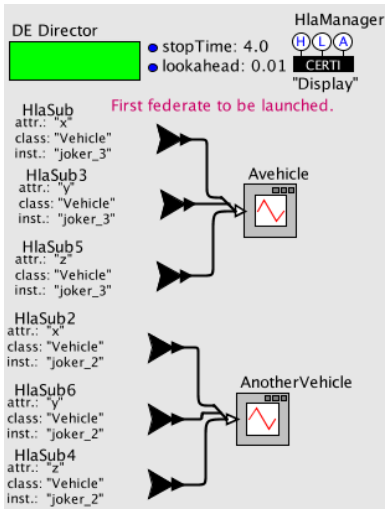


Figure 17: Display.

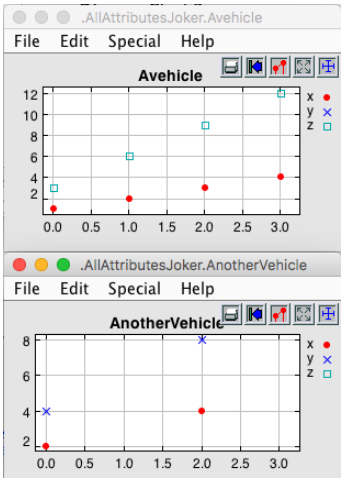


Figure 18: Plotter using Fig. 17 and Fig. 15.

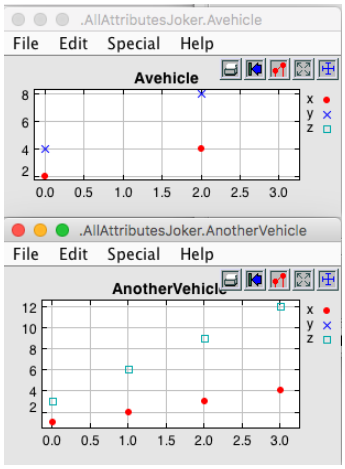


Figure 19: Plotter using Fig. 17 and Fig. 16.

know which instance will be displayed in each plotter, so two wilcards, joker_1 and joker_2, are used in the HlaSubscribers.

The two instances of class Vehicle (robot and uav) can be updated into two different ways: using two federates, each one updating a different instance, as depicted in Figure 15, or by a unique federate, updating the two instances, as depicted in Figure 16.

Let us call `DifferentFederates` a federation with federates represented in Figure 17 and Figure 15. The result of this simulation is represented in Figure 18. And let us call `SameFederate` a federation with federates represented in Figure 17 and Figure 16. The result of this simulation is represented in Figure 19.

Both results are correct, since the user is not interested in which instance will be displayed in a particular Display (Avehicle and AnotherVehicle). It can be noted that joker_3, connected to Avehicle, was associated to UAV instance (with coordinates x and z) in Figure 18, but was associated to the Robot instance (with coordinates x and y) in Figure 19.

If the federate Display is replaced by the DisplaySome, depicted in Figure 20, the simulation results, represented on Figure 21, are correct when using this new federate with the two federates depicted on

Figure 15. However, the results are incorrect when using the federate on Figure 16: only the x coordinate of robot and uav are displayed. This indicates that the model of federate DisplaySome is not a good model: the simulation cannot be dependent on the way the instances are updated. In the model on Figure 20, the computation with the values of the attributes is dependent on the instance. The wildcards cannot be used in this federate: joker_1 must be replaced by robot (since the ground coordinates are used), and joker_2 must be replaced by uav (since the wall coordinates are used).

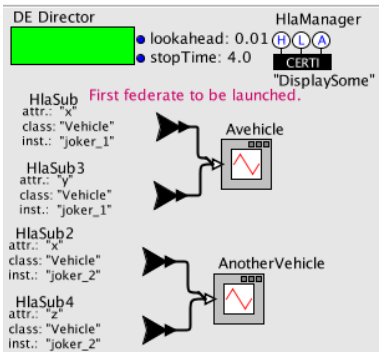


Figure 20: Only some attributes are reflected.

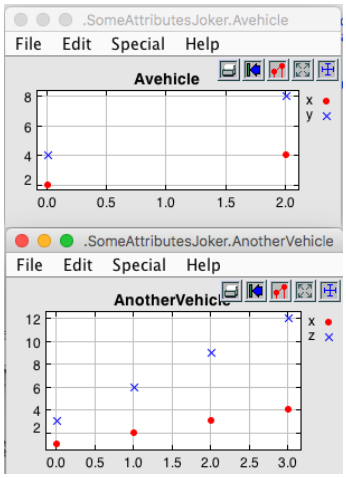


Figure 21: Plotter using Fig. 20 and 16.

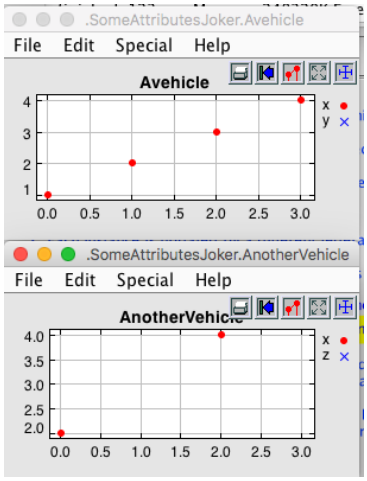


Figure 22: Plotter using Fig. 20 and 15.

5 FAQ about HLA

5.1 What is a class handle, attribute handle and instance handle?

Figure 23 depicts the file OneClassVehicle.fed used in the federation MyVehicles. The federation is created by the first launched federate, using the service createFederationExecution(MyVehicles, OneClassVehicle.fed).

If the rtig is launched by hand in a terminal, the log message represented on Figure 24 appears on the terminal. Notice that beside each class name and attribute name a new information appears: (id k), with k=1..4. These are the handles provided by the rtig to each class name and attribute name. All services related to classes and attributes will refer to these handles. They are not known before the federation execution. This log is the same, whichever the federate Display on Figure 17 or DisplaySome on Figure 20 is used, since they are using the same fed file.

The discoverObjectInstance(objectInstanceId, classHandle, instanceName) callback "informs the federate that a new object instance has come into existence" [3]. This method provides the handle called objectInstanceId, that can be different in other federates that subscribed to the attributes of the same class.

The discoverObjectInstance (DOI) callback is the counterpart of the service registerObjectInstance(classHandle, instanceName). A handle is provided for his instance; the RTI provides the services getObjectInstance-Handle() and getObjectInstance-Name(). The instanceName is a symbolic name associated with the new instance and is not interpreted by the RTI. However, this name must be unique to the federation [3].

```
;; Federation with one class and three attributes
(Fed
 (Federation MyVehicles)
 (FedVersion v1.3)
 (Spaces)
 (Objects
  (Class ObjectRoot
   (Attribute privilegeToDelete reliable timestamp)
   (Class RTIprivate)
   (Class Vehicle
       (Attribute x RELIABLE TIMESTAMP)
       (Attribute y RELIABLE TIMESTAMP)
       (Attribute z RELIABLE TIMESTAMP)))
```

Figure 23: FED file.

```
(FED
    (Federation "MyVehicles")
    (FEDversion "v1.3")
    (spaces
    )
 (objects
   (class "ObjectRoot" (id 1)
      (attribute "privilegeToDelete" (id 1) reliable timestamp)
      (class "RTIprivate" (id 2)
      )
       (class "Vehicle" (id 3)
           (attribute "x" (id 2) reliable timestamp)
           (attribute "y" (id 3) reliable timestamp)
           (attribute "z" (id 4) reliable timestamp)
       )
```

Figure 24: rtig log on terminal.

6 Installing Ptolemy-HLA framework

6.1 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).

6.2 Installing CERTI

CERTI is an Open Source HLA RTI (runtime infrastructure) [16]. As the for the moment (April, 2019) the stable version is CERTI 4.0.0 (git commit c2146229adc0 (March, 1, 2019). An RTI manages the communication between federates. CERTI has a GPL license, so it cannot be distributed along with Ptolemy. You have to separately install it. First, check if you have an version of CERTI in `$HOME/pthla` older then 4.0.0, you may have certi and certi-tools:

```
ls ~/pthla
```

If there is an old version, move them to another folder.

```
mv ~/pthla/certi ~/pthla/certi.old ;  mv ~/pthla/certi-tools ~/pthla/certi-tools.old
```

Then install the latest version:

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

If you are installing on MacOS with any version since El Capitan, there is one additional step. The dynamically loaded libraries need to be in `/usr/local/lib`. To put them there, do the following:

```
sudo -i
(enter your root password)
cd /usr/local/lib
ln -s /Users/YOURUSERNAME/pthla/certi-tools/lib/* .
exit
```

Replace YOURUSERNAME above with your user name, assuming CERTI is installed in your home directory, which is the default.

If for some reason the script does not work, you can follow the steps in section C for your operating system of choice.

7 Error Messages

Some common errors are reported as tests that raise exception. Examples of the right way to use the Ptolemy-HLA environment are in `org/hlacerti/test/auto`.

7.1 Known solutions

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

7.1.1 Initializing

When: After the last federate is launched and it has the "Register Synchronization Point" set
Message: The last federate stucks with the message `initializing` (in the left bottom corner of the model). The following message may also appear:

```
_read(): dyld: Library not loaded: libCERTId.4.dylib
  Referenced from: /Users/your-login/pthla/certi-tools/bin/rtig
  Reason: image not found
```

Reason: This error appears only in MacOS after El Capitan, related with dynamic libraries and the SIP issues. See `https://chess.eecs.berkeley.edu/ptexternal/wiki/Main/HLA#ElCapitan` for more details.

Solution: It does not occur if the steps presented in 6.2 are followed. However, if you have already a CERTI version installed, you may need to remove the links in /usr/local/lib before execute the symbolic link command. Indeed, only one version of CERTI can be installed at a time in the new MacOS, what was not required before and is still not required in Linux.

7.1.2 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 C.1.
Solution 2: Just launch again the federate. You can kill the rtig before.

7.1.3 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 9. 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.

7.1.4 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 9). 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.

7.1.5 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.

7.1.6 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.

7.1.7 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.

7.1.8 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 9 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. 5.

7.1.9 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.

7.1.10 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.

7.1.11 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] Okan Topcu and Halit Oguztuzun. *High Level Architecture* in *Guide to Distributed Simulation with HLA*, Simulation Foundations, Methods and Applications Series, Springer International Publishing, ISBN 978-3-319-61266-9, DOI 10.1007/978-3-319-61267-6, eBook ISBN 978-3-319-61267-6, 2017.

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

[3] Department of Defense, High Level Architecture Run-Time Infrastructure: Programmer's Guide. RTI 1.3 Version 6, 12 March 1999.

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

[5] IEEE Standard for Modeling and Simulation (M & S) High Level Architecture (HLA)– Federate Interface Specification, IEEE, pages 1–378, Aug. 2010.

[6] 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.

[7] 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.

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

[9] 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.*

[10] 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.

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

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

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

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

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

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

[17] 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.

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

Appendices

A HLA Management areas and services

HLA services are grouped into six management areas related to the federate life cycle. Figure 2 describes the subset of HLA services used in the Ptolemy-HLA framework. See [4] and [5] for a complete description of all HLA services.

Areas	Services	Description (non-formal)
Federation	createFederationExecution()	create a federation
	joinFederationExecution()	join a federation
	resignFederationExecution()	quit a federation
	destroyFederationExecution()	destroy a federation
	registerFed...Sync...Point()	register a synchronization point
	sync...PointReg...Succeeded()⋆	register synchro point succeeded
	announceSynchro...Point() ⋆	wait a synchronization point
	synchronizationPointAchieved()	release from a synchro. point
	federationSynchronized() ⋆	announce synchronization
	tick()	allow to get callbacks from RTI
Declaration	publishObjectClass()	declare publication of a class
	subscribeObj..ClassAttributes()	subscribe to a class
	unsubscribeObjectClass()	unsubcribe to a class
	unpublishObjectClass()	unpublish a class
Object	registerObjectInstance()	register an object instance
	discoverObjectInstance() ⋆	for object instances discovering
	updateAttributeValues(), UAV	send & update value
	reflectAttributeValues() RAV ⋆	receive updated value
Time	enableTimeRegulation()	declare federate is regulator
	timeRegulationEnabled() ⋆	federate as regulator succeeded
	enableTimeConstrained()	declare federate constrained
	timeConstrainedEnabled() ⋆	federate as contrained succeeded
	timeAdvanceRequest(), TAR	ask to advance federate's time
	timeAdvanceGrant() TAG ⋆	notify time advancement granted
	nextEventRequest(), NER	ask to advance federate's time

Figure 25: HLA services with a ⋆ are sent from RTI to Federates (callbacks); all other services are from Federates to RTI. Credits: table from [9].

Figure 26 depicts the relationship between HLA services and Ptolemy components. The reader can see [9] and [11] for more details.

B Synchronization Point

Figure 27 shows the time diagram with the services related to the synchronization point for a federation with 3 federates. Federate 1, the first federate to be launched; it creates the federation and join it. This federate is a kind of manager in this scenario. The designer of this federation choose to make this first federate: i) the one that registers the synchronization point sending to the RTI the service `registerFederationSynchronizationPoint`, and ii) the one that sends the callback `synchronizationPointAchieved` *after the user decided all federates have joined* by pressing the Enter key (CR or carriage return character).

In the meanwhile, each (non-register) federate launched joins the federation sending to the RTI the service `joinFederationExecution()` and receive from the RTI the callback `announceSynchronizationPoint()`. At some point in its code, when it is ready for starting the time advancing, it sends to the RTI the service `synchronizationPointAchieved()`.

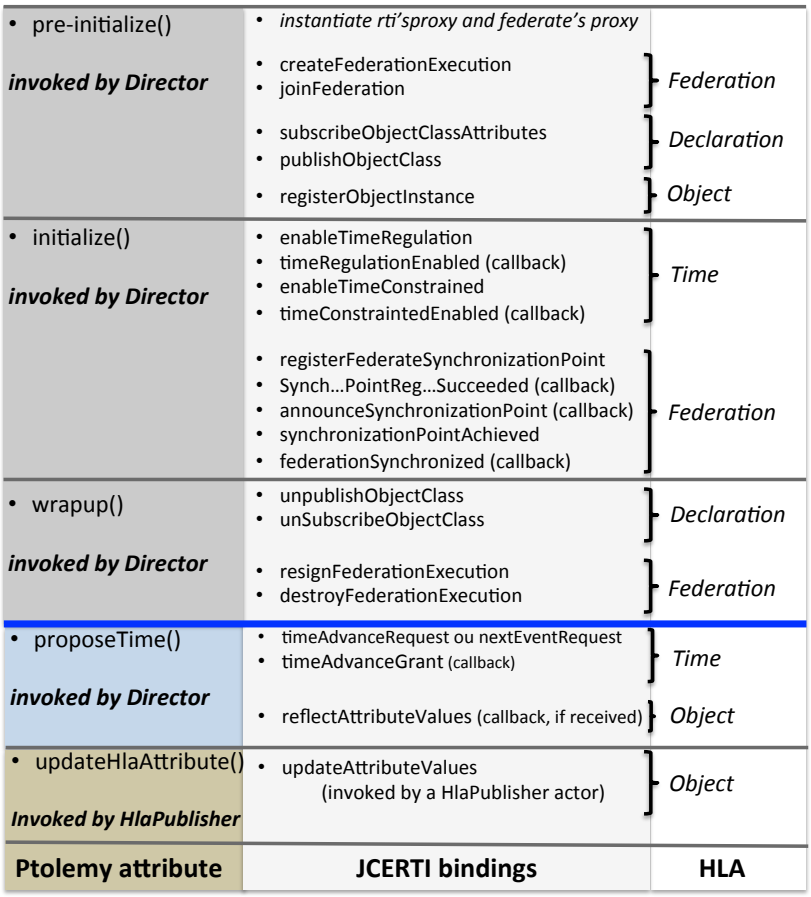


Figure 26: Architecture of the *HlaManager* attribute (from [9]).

When all federates belonging to the federation send the callback `synchronizationPointAchieved()`, then the RTI sends for all federates the callback `federationSynchronized()` and the simulation starts. Notice that the RTI is able to deal with federates that leave a federation and join a federation during the federation execution, because a synchronization point has a *synchronization set*. For more details about the the goals and how deal with the synchronization point, see [17] and the HLA standard [4].

The choice made in the Ptolemy-HLA framework was to not use a manager federate, but guarantee that the user knows when all federates has been launched – and are waiting for the synchronization point – by making the last federate to be launched the one that register the synchronization point. The user set this parameter when configuring the HlaManager, as depicted in figure 10. Figure 28 depicts two Ptolemy federates: the Consumer (first to be launched) and the Producer (last to be launched). It does not matter which federate is the last, but this one is the only federate to have this parameter set.

C Installing CERTI

CERTI is an Open Source HLA RTI (GPL, libraries are LGPL) [16]. 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:

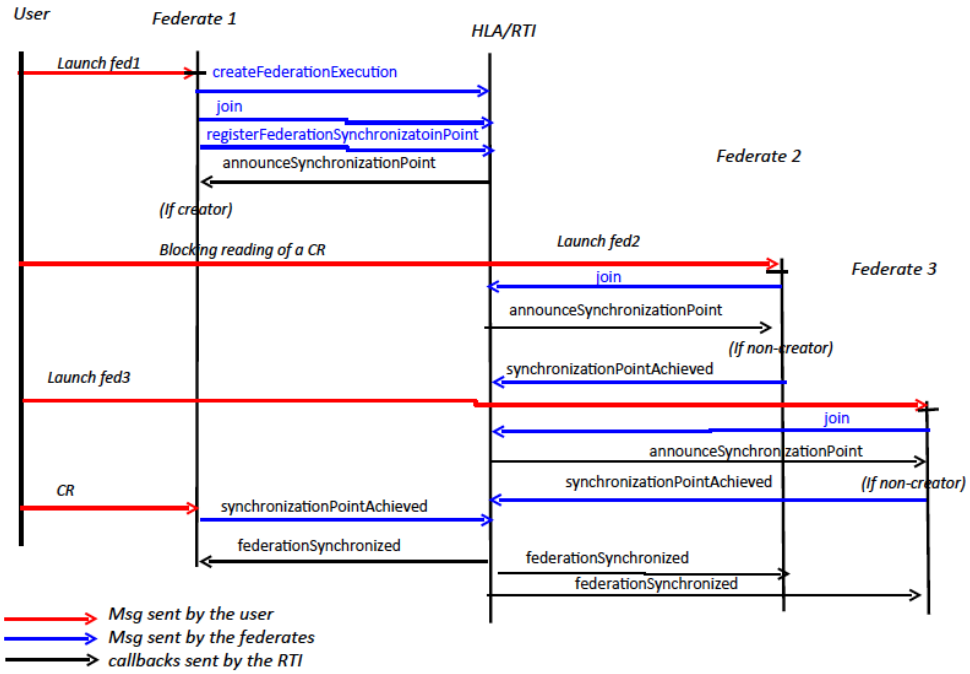


Figure 27: Scenario where the *first* federate registers the synchronization point.

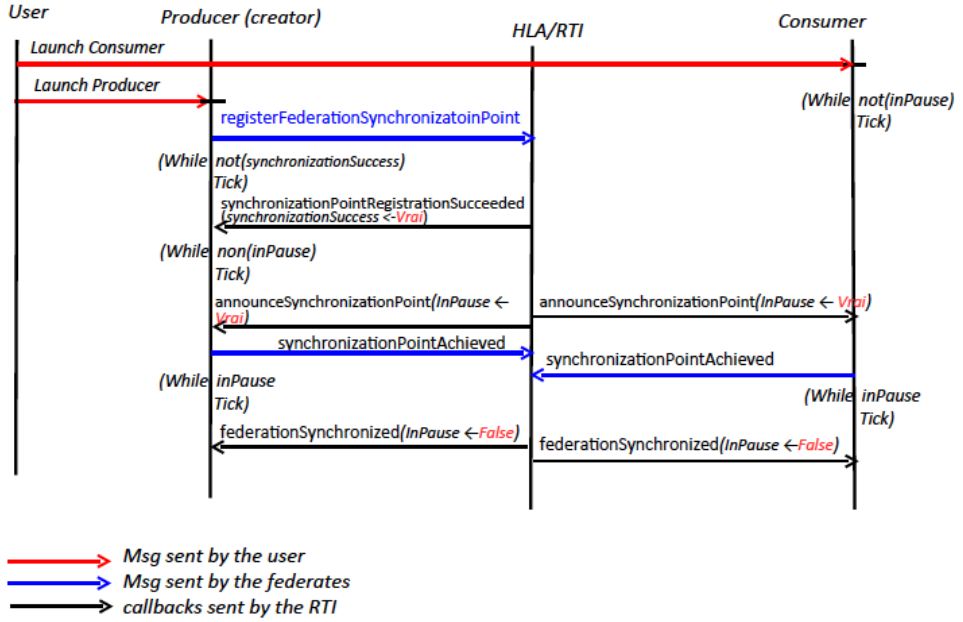


Figure 28: Scenario where the *last* federate registers the synchronization point..

$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 6.2 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
```

C.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`.

C.2 Installing CERTI on Windows

TBD

C.3 Installing CERTI on MacOS

Follow the same instructions as for Linux (section C.1), but with one additional step that is required for MacOS since El Capitan. The dynamically loaded libraries need to be in `/usr/local/lib`. To put them there, do the following:

```
sudo -i
(enter your root password)
cd /usr/local/lib
ln -s /Users/YOURUSERNAME/pthla/certi-tools/lib/* .
exit
```

Replace YOURUSERNAME above with your user name, assuming CERTI is installed in your home directory, which is the default.
4. Open a new terminal, and run a demo again:

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

D 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.

E 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.

F 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 [12].

- Revision 72943: TAR time management was implemented by Yanxuan LI [13]; 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 [14]: 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 [15].

- 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 [15].