Ptolemy-HLA: A Cyber-Physical System Distributed Simulation Framework

Janette Cardoso(✉) and Pierre Siron

ISAE-SUPAERO, Université de Toulouse, Toulouse, France
{janette.cardoso,pierre.siron}@isae-supaero.fr

Abstract. The Ptolemy-HLA distributed co-simulation framework leverages two open source tools, Ptolemy II and HLA/CERTI, for the simulation of Cyber-Physical Systems (CPS). This framework enables dealing with three important issues: (1) Distribution of a simulation, allowing to scale up models and performance; (2) Interoperability of tools, allowing reusability and interfacing with other simulators or real devices/systems; (3) Heterogeneous simulations (discrete events, continuous time).

The framework extends Ptolemy both, by coordinating the time advance of various Ptolemy instances, and by allowing data communication between them with the help of HLA management services.

These additions enable the creation of HLA federates (i.e., simulators) in a Federation (i.e., a distributed simulation) in an easy way, since the user does not need to be an HLA specialist in order to design a Federate. The paper presents the new components added to Ptolemy, some semantic issues, an application example and performance analysis.

Keywords: Distributed simulation · HLA · Cyber-physical systems

1 Introduction

There are many advantages to a distributed simulation. A first aspect comes from the nature of the systems to be simulated, which are nowadays more and more distributed and complex. It can be more appropriate to build a distributed simulation of a distributed system, as, for example, a fleet of drones, than a monolithic simulation. The distributed simulation is more representative and it mimics the real system without simplifications better. The second aspect is the complexity of the system that is translated in an integration of complex and heterogeneous models. Distributed simulation is often associated with the notion of simulation interoperability offering the possibility of integrating different simulators, such as specific domain simulators. The reuse of a simulator can offer a significant reduction of design and development time as well as improve quality of the simulation. Distributed simulation is also relevant for non-functional requirements. It can reduce the simulation time (parallelization speedup) or enable larger simulations (scalability) [1]. Finally, models can be treated as black-boxes or executed on remote processors, and we can deal with IP issues [2]. For all these reasons,

https://doi.org/10.1007/978-3-319-95246-8_8

distributed simulations are adapted for the challenging study of Cyber-Physical Systems, that are complex, heterogeneous and distributed. But performing distributed simulations is difficult, and we propose in this paper a principled and friendly way to build these simulations.

Ptolemy is an open source modeling and simulation tool for heterogeneous systems, developed at the University of California Berkeley. This tool is well suited for modeling CPS [9] by providing different models of computation (MoC), such as continuous time for describing physical systems or discrete event for describing software and control.

The IEEE High-Level Architecture (HLA) standard [15] 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.

PTIDES [26], a framework implemented in Ptolemy, is used to design event-triggered, distributed, real-time systems. It leverages network time synchronization to provide a coherent global meaning for timestamps in distributed systems. Moreover, it has the nice characteristic that it carefully relates multiple timelines (physical time, logical time, *oracle* time). However, even if it allows the simulation and execution of a distributed system, the entire system is modeled in only one model.

The Functional Mock-up Interface (FMI) standard for co-simulation allows the exchange and interoperation of model components or subsystems designed with different modeling tools. However, it is up to the user to guarantee a coherent time representation when the simulation is distributed. There are works proposing an integration of HLA and FMI [13,23,24]. An HLA-FMI wrapper that turns a FMU into a full featured HLA federate exists [14], but it seems to deal only with data. FIDE, which stands for FMI Integrated Development Environment [8], is an IDE for building applications using Functional Mock-up Units (FMUs) that implement the FMI standard in the Ptolemy framework. Their work focuses on a master algorithm that deterministically combines discrete and continuous-time dynamics. However, it does not deal with distributed simulation. A detailed analysis of the time representation in the FMI framework is done in [7]. It proposes the superdense model of time using integers (implemented by the class `Time` in Ptolemy) for solving many problems of time representation. In particular their paper discusses the choice of resolution to be used when the FMUs (components of a co-simulation) have different resolutions. The coordination of different notions of time is an issue that also comes up in cyber-physical systems [29].

FORWARDSIM [10] is a proprietary software toolbox that allows for distributed simulation using the HLA standard. It provides the HLA Blockset for Simulink and the HLA Toolbox for Matlab. The user must know the entire standard well and it is up to the user to call each service by adding the corresponding block to the model.

In this paper, we will present a framework called Ptolemy-HLA: it brings together the heterogeneity provided by Ptolemy (i.e. the possibility to mix continuous, discrete or other MoCs) and interoperability provided by HLA (i.e. the possibility to mix simulation models, pieces of code and physical equipment). We consider that, in relation to ForwardSim, our framework provides a friendly, open-source interface to the user which requires minimal knowledge of the HLA standard. Similar to PTIDES, we carefully tackle time coordination between HLA and Ptolemy timelines. Our approach allows the distributed simulation of models over a network. As of this moment, the Ptolemy-HLA framework does not allow to build applications using FMUs as per the FMI standard.

HLA users can benefit from already existing Ptolemy models that can be easily translated into the Ptolemy-HLA framework. For Ptolemy users, the Ptolemy-HLA framework can be useful if a large model already exists. Splitting the components of such a model into distributed models, running on a same computer or different computers, allows the user to use a good *granularity* of the model. This can improve handling and may be more representative, since each (distributed) model can be more detailed or extended, for example, modeling two aircraft engines separately and/or using a more complex model for the engine. From a single model, e.g., a quad-rotor model, a model of a fleet of quad-rotors can be easily obtained. The interoperability of the HLA standard allows to use models or code, as well as real devices with an interface compliant with this standard.

This paper is organized as follows. An overview of HLA and Ptolemy is presented in Sect. 2. Section 3 describes the co-simulation framework: how time is advanced and how data is exchanged considering the rules of both, Ptolemy and HLA. Section 4 illustrates the results of our approach applied to a concrete case-study: a flight control system of a F14 aircraft. Finally, Sect. 5 presents concluding remarks and our future work.

2 Tools for Distributed Simulation and Heterogenous Simulation

The simulation of Cyber-Physical systems needs to deal with both, heterogeneity and distribution of simulators. We choose two open source tools for taking advantage of each one of these needs: Ptolemy II and HLA/CERTI.

2.1 Ptolemy

Quoting [26] "Ptolemy II is an open-source simulation and modeling tool intended for *experimenting with system design techniques*, particularly those that involve combinations of different types of models". Being interested in cyber-physical system (CPS) modeling and simulation, Ptolemy's ability to represent heterogeneous system, offered by its different Models of Computations (MoC), is a very important feature. A MoC is specified by a component called a Director, represented by a green block as shown in Fig. 1. In this paper, we will deal with two directors: Discrete Event (DE) and Continuous (CT). They will be used for

modeling the cyber and physical part of a CPS. For the sake of simplicity, a model with a DE director will be called a DE model. The same is done for the CT director.

Another important feature in Ptolemy is its model time known as superdense time, which allows two distinct ordered events to occur in the same signal without time elapsing between them [7]. A superdense time value can be represented as a pair (t, n), called a timestamp, where t is the model time and n is a microstep (also called an index). The model time represents the time at which an event occurs, and the microstep represents the sequencing of events that occur at the same model time [26]. The initial (default) value for the microstep is 1 when using a DE director in a model and 0 when using a CT director. In this paper, for the sake of simplicity, a timestamp $(t, 1)$ for DE and $(t, 0)$ for CT will be represented only by t. The time t is represented as mr, where m is an integer and time resolution r is a double-precision floating point number. Therefore, the (model) time resolution is the same throughout its execution [26] which is not the case when IEEE-754 double is used. The Ptolemy time representation is implemented by a Java class called `Time`.

Time Advance in Ptolemy [4,26]. Every director in Ptolemy has a local clock. If the director is at the top level of the model, i.e., if there is no enclosing director, then the advance of the clock is entirely controlled by this director. An event in the Ptolemy calendar queue is represented as $e(v, (t, n), A_j)$, where A_j is the j^{th} input port of destination actor A. All events are generated locally, and the director will always advance time to the smallest timestamp of unprocessed events. In a DE model, this timestamp is that of a given event and only the destination actor of this event is executed. In a CT model, the timestamp is computed by a solver, and all actors are executed. If there is more than one event with the same timestamp, the destination actors are fired in the order given by a topological sort of the actors, which is a list of the actors in data-precedence order. This behavior ensures determinism.

Ptolemy also provides a so-called TimeRegulator interface with a `proposeTime` method. This interface is implemented by attributes that wish to be consulted when a director advances time. The director will call the `proposedTime` method, passing it a proposed time to advance to, and the method will return either the same proposed time or a smaller time. This method has a key role in the Ptolemy-HLA framework.

Data Exchange in Ptolemy. Actors in Ptolemy have input and/or output ports. Actors with only input ports are called sink actors (e.g., a `TimeDisplay` as in Fig. 4c) and actors with only output ports are called source actors (e.g., a `DiscreteClock` as in Fig. 4c). A *token* is the unit of data (with a type), such as the numerical value of an *aircraft vertical speed*. It is communicated between two actors via ports: created by one actor A_1, *sent through* an output port i, and *received by* (the input port j) of a destination actor A_2, as represented in Fig. 1. This token can be received by several actors.

Link Between Time Advance and Data in Ptolemy. There is a *production-consumption* phase related to the time advance (if the models are timed). A simplified view is the following: When a token is produced by A_1 (as in Fig. 1) at output port A_{1_i} at current time t, an event $e(v, (t', n'), A_{2_j})$, $t' \geq t, n' \geq n$, is put in the calendar queue. When this event is the earliest one in the calendar queue, the DE director will advance its (current) time to (t', n'), fire (or execute) A_2 and consume the token in the input port A_{2_j}. Most actors, such as `AddSubtract`, `CurrentTime`, `Integrator`, have $t' = t$ and $n = n'$. Some actors provide mechanisms for delaying events, e.g., `TimeDelay` $(t' > t, n' = 1)$ or `MicrostepDelay` $(t' = t, n' = n+1)$ [26]. A token will be referred to as an event e, $e(t)$, or $e(t, n)$.

2.2 High Level Architecture (HLA) Standard

The High-Level Architecture (HLA) [15,16] is a standard for distributed discrete-event simulations, generally used to support analysis, engineering and training. The approach promotes reusability and interoperability. A simulation entity performing a sequence of computations is called a *federate*, and the set of federates simulating the entire system is called a *federation*. Federates are connected via the Run-Time Infrastructure (RTI), the underlying middleware functioning as the simulation kernel. The lollipop architecture of an HLA federation is depicted in Fig. 2.

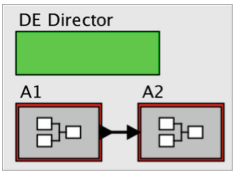


Fig. 1. Ptolemy model.

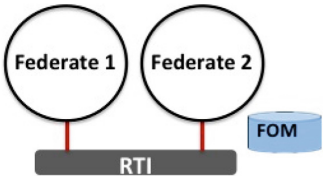


Fig. 2. HLA architecture.

The HLA standard defines a set of rules describing the responsibilities of federations and the federates, e.g., *all data exchange among federates shall occur via the RTI*. Among the rules, an important one concerns the time advance: *A federate delegates its time advance to the RTI*. Another one concerns the sending of data: *A federate cannot send an event earlier than* $t+lah$, where t is its current logical time and *lah* its the lookahead [15].

The standard also defines an interface specification for a set of services required to manage the federates and their interactions. In this paper, we will present the services related to Time Management and Data Management.

Data Exchange in HLA. For each federation, a Federation Object Model (FOM) describes the shared objects, interaction classes and their attributes. The object management services allow message exchange between federates. Let us consider two federates F_1 and F_2: F_1 sends the signal *aircraft vertical speed* to F_2. In HLA terms, F_1 publishes the class `Aircraft_speed` and F_2 subscribes to attribute `v_speed` of this class. The HLA services used are, respectively,

`publishObjectClass` and `subscribeObjectClassAttributes`. There are two steps concerning the *object management*:

(1) When federate F_1 is launched, it registers an object instance of `Aircraft_speed` class (service `registerObjectInstance`). When federate F_2 is launched, it discovers object instances `Aircraft_speed` related to the attribute `v_speed` it subscribed (callback `discoverObjectInstance`);
(2) During the simulation, F_1 sends through the RTI a new value of `Aircraft_speed.v_speed` using the service `updateAttributeValues` (UAV). The RTI sends this value to F_2 using the callback `reflectAttributeValues` (RAV).

Time Advance in HLA. HLA time management services enable deterministic and reproducible distributed simulations [5]. Each federate manages its own logical time and communicates this time to the RTI that ensures that federates observe events in the same order [12].

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. The time step between successive TAR service calls can change during a simulation, but it is frequently chosen as a fixed time step TS. There is a trade-off between the performance and the precision of the simulation according to the time step used. The user needs to carefully make this choice. Such a choice is not required for NER, since the time advance request has the timestamp of the next event. According to the HLA standard, a federate can switch from TAR to NER and NER to TAR during a simulation. However, in our framework, a federate can use one of these services but the user must make the choice before the simulation. The HLA standard does not impose a time representation. In general, the HLA standard proposes IEEE-754 double-precision floating point numbers.

Is There a Link Between Data Exchange and Time Management? By default, the RAV callbacks are received during the time advance phases and they are delivered in the order messages are received. This is the one and only link between data exchange and time advance for so-called HLA real time simulations. For the sake of repeatability and determinism, the data exchanges are in timestamp order. This order can also reflect causality relations.

When dealing with timed systems as CPS, the messages must be timestamped and the federates are time-constrained and time-regulating[1]. Besides the value

[1] A federate can only advance its time if it is granted by the RTI. When this federate is *time-constrained*, this grant is computed by the RTI with knowledge of the time advancements of the *time-regulating* federates, so that the conservative property of the distributed simulation is guaranteed between regulating and constrained federates.

of the attribute of a class instance, the UAV service has a timestamp. When the simulator is at current date t, it computes a new value of the attribute for a date $t*$ in the *future*, $t < t* \leq t+$ lookahead. This lookahead (a value associated with a federate) establishes a lower bound on the timestamps that can be sent. In a distributed simulation, strictly positive lookahead values allow the use of well known, deterministic and efficient distributed algorithms for the time management in the RTI. The lookahead can be equal to zero, and in this case may cause a deadlock. A first alternative is to rely on new algorithms in the RTI, for example the use of the Null Message Prime protocol [6] or the computation of a distributed snapshot [19]. A second alternative is that the user resolves the (possible) deadlock by using the TARA and NERA services[2] instead of sending a message with a zero lookahead [11].

Besides the attribute value of a class instance, the RAV service has a timestamp. The delivery of these callbacks is done in chronological order of the timestamp values. At current time t, during the time advance phase starting by TAR(t') and ending by TAG(t'), all RAV callbacks have a timestamp t'' that respects $t < t'' \leq t'$. These callbacks can concern the same instances of a class or different instances (of the same class or of different classes). For a time advance phase starting with NER(t'), if any callback with timestamp t'' is received, this phase will end with a TAG(t''). If there are any RAV callbacks with timestamps within t'' and t', they will be delivered during the following time advance phase(s). This is the main difference between TAR and NER concerning the way time is advanced. Section 4 shows that the execution time of a federation is also different between TAR and NER. We could have an equality between the timestamps of two RAVs of different object instances. HLA does not allow to specify an order in this case (FIFO order between messages coming from different federates). For the sake of determinism, the user code must produce the same result for the different execution cases. This is not difficult because, generally, the new state computation (and, in general, the sending of a data) in a simulation follows the time advance phase when all the required data is received.

3 Putting Ptolemy and HLA Together

The distributed co-simulation framework must comply with both, HLA and Ptolemy rules, in particular concerning data exchange and time advance ones. As of now, only NER and TAR are implemented in the framework and the lookahead cannot be zero. The way the coupling is designed is discussed next.

[2] NERA stands for Next Event Request Available and TARA for Time Advance Request Available. A TARA(t) (respectively, NERA(t)) that ends with a TAG(t) can be followed with the production and the reception of new events timestamped with t. If federates exchange data at the same time in a loop, the loop must be broken by calling TAR(t) (respectively, NER(t)). Then no additional event will be delivered to the federate with timestamp t and time can be advanced.

For the sake of simplicity, a time-stepped federate will be called a TAR federate, and an event-based federate will be called a NER federate. In this work, the RTI compliant with HLA is CERTI, an open source RTI written in C++ [25]. However, another RTI could be used.

3.1 How Time Is Advanced in the Ptolemy-HLA Framework

The first thing to point out is the existence of two timelines in a federate: the Ptolemy timeline t and the HLA timeline h. Both timelines use the same global unit (e.g., second or millisecond). Ptolemy (local) logical time t must be compliant with HLA logical time h. It means that the time must be advanced using HLA services NER or TAR, and a new interface called `HlaManager` was designed. Concerning the time, it has a method called `proposeTime` implementing a `TimeRegulator` interface (presented in Sect. 2.1). When Ptolemy wants to advance to the timestamp t' of the earliest event in its calendar queue, the DE director will check if this is possible with the `proposeTime` method. In this section, it is considered that no data exchange exists in the federation, in order to focus on time advance. According to the federate time management NER or TAR, the time will be advanced by calling Algorithms 1 or 2.

As time representation in CERTI and Ptolemy are different, a conversion is needed in both algorithms: f converts `double` to `Time`, and g converts `Time` to `double` [18,21]. To minimize the comparisons, the time step TS (see Sect. 2.2) in Algorithm 2 is represented as `double` in Ptolemy model.

An important difference can be noticed between Algorithms 1 and 2 when Ptolemy wants to advance to t', the timestamp of the earliest event in its queue:

- at least one NER($g(t')$) is called in Algorithm 1, but more NER($g(t')$) can be called according to the number of TAG messages received. Each time a TAG($h'' < g(t')$) is received, Ptolemy advances to $f(h'')$. When TAG($h'' = g(t')$) is received, Ptolemy advances to t';
- k TARs will be called in Algorithm 2, $k \geq 0$, with

$$k = (\lfloor g(t') - h)/TS \rfloor - 1). \tag{1}$$

  When the last TAG is received, Ptolemy advances to t', with the guarantee that $k * TS < g(t') < (k + 1) * TS$.

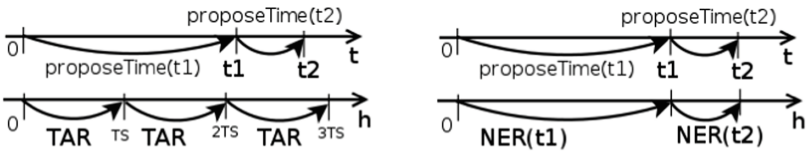
It is worth mentioning that, after asking to advance to t', Ptolemy time eventually advances to t' and has the same time history, independent of the time management (NER or TAR) used. But HLA time can have a different time history according to the time management, as presented in Fig. 3 and some examples shown next. For the sake of readability, time conversions f and g are not represented in Fig. 3.

Algorithm 1. NER ProposeTime(t')

1: NER($g(t')$)
2: **while** not granted **do**
3: TICK() ▷ Wait $TAG(g(t'))$
4: **end while**
5: $h \leftarrow g(t')$ ▷ Update `HLA` time
6: **return** t' ▷ Update `PtII` time

Algorithm 2. TAR ProposeTime(t')

1: **while** $g(t') > h + TS$ **do**
2: TAR($h + TS$)
3: **while** not granted **do**
4: TICK() ▷ Wait $TAG(h + TS)$
5: **end while**
6: $h \leftarrow h + TS$ ▷ Update `HLA` time
7: **end while**
8: **return** t' ▷ Update `Ptolemy` time



a) TAR federate b) NER federate

Fig. 3. Time advance using TAR or NER without consider time conversion.

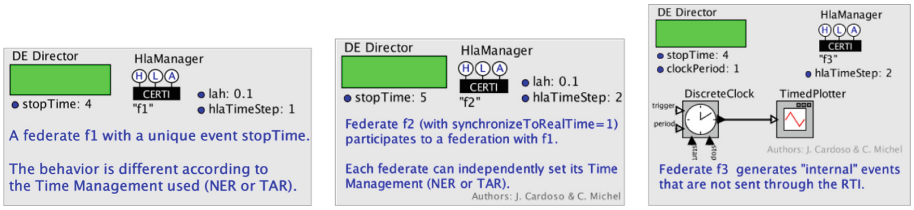
Let us consider federations `F`$_a$ `= {f1}`, `F`$_b$ `= {f1,f2}` and `F`$_c$ `= {f1,f3}`; federates `f1`, `f2` and `f3` are depicted in Fig. 4a and b and c. Each federation is used to explain a particular point in the time advance. No federate sends any data through the RTI. To keep track of the time representation in Ptolemy and HLA, an index will be added to the time value v in each timeline: v_T and v_d for Time and double.

Federation F_a: A unique federate advances its time with the RTI.

The `f1` model has a (current) Ptolemy start time $t = 0_T$ and HLA start time $h = 0_d$. The unique next event e in the `f1` calendar queue is the *stop time* event $e(t' = 4_T)$. Table 1 depicts the services called and its callbacks in this federation, as well the final Ptolemy and HLA time when using NER or TAR (with time step $\text{TS}_{f1} = 1_d$) time management. As discussed above in the presentation of `proposeTime` Algorithms 1 and 2, the Ptolemy final time is the same using NER or TAR as well its time history: $\{0, 4_T\}$. But the HLA time history is different: $\{0, g(4_T)\}$ when using NER, and $\{0_d, 1_d, 2_d, 3_d, 4_d\}$ when using TAR.

Table 1. Time advance of `f1` using NER or TAR.

Type	Service call	Callback	Final h	Final t
NER	NER ($g(4_T)$)	TAG($g(4_T)$)	$g(4_T)$	4_T
TAR	TAR(1_d), TAR(2_d), ... TAR(4_d)	TAG(1_d), TAG(2_d), ... TAR(4_d)	4_d	4_T



(a) Federate f1. (b) Federate f2. (c) Federate f3.

Fig. 4. Federates used in $\texttt{F}_a$ = {f1}, $\texttt{F}_b$ = {f1,f2} and $\texttt{F}_c$ = {f1,f3}.

Federation F_b*:* Two federates advance their time in coordination with the RTI. Federates `f1` and `f2` have the same parameters except that `f2` has (HLA) time step $\text{TS}_{f2} = 2_d$ (needed when TAR is used) and *stop time* is 5_T. Whatever `f1` uses NER or TAR, its results (final time and time history) are the same, as depicted in Table 1. For `f2`, its final Ptolemy time is $t_{f2} = 5_T$ and Ptolemy time history is $\{0, 5_T\}$. The `f2` HLA final time 4_d and its time history is $\{0_d, 2_d, 4_d\}$, since $g(5_T) \not> 4_d + 2_d$ and so noTAR(6_d) is executed (Algorithm 2, line 1).

Federation $\texttt{F}_c$*:* Time coordination with a federate that produces internal events. Federate `f3` has two internal events, $e(1_T)$ and $e(3_T)$, produced by `DiscreteClock`, and has *stop time* $e(4_T)$. Its (HLA) time step is $\text{TS}_{f3} = 2$ (needed when TAR is used). Now, besides the event *stop time*, its internal events will be added to the calendar queue. The rule is the same: the Ptolemy model needs to check with HLA if it can advance to the time of the event. The Ptolemy time history of `f3` is the same using NER or TAR: $\{0_T, 1_T, 3_T, 4_T\}$. Concerning HLA time history, it is $\{0_d, g(1_T), g(3_T), g(4_T)\}$ when using NER, and $\{0_d, 2_d, 4_d\}$ when using TAR. Ptolemy and HLA time stories of `f1` are the same as the ones in federation $\texttt{F}_a$.

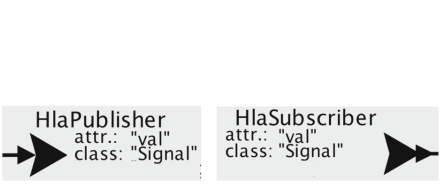
The distribution of a simulation is necessary and/or appropriate, but it comes at a price. Beside the complexity of the implementation, the timestamp of a message can change according to the simulator tool, as presented in [21]. A federate may have two kinds of events: (i) events that are only internal to the model (as `f3` participating in Federation $\texttt{F}_c$ above); (ii) events that are sent and received through the RTI. The latter will be discussed in the following.

3.2 How Data Is Exchanged in the Ptolemy-HLA Framework

The unit of data in Ptolemy is a *token*, and in HLA it is the *attribute* of an *object class* described by the FOM. As seen in Sects. 2.1 and 2.2: (i) both are timestamped and have a value with a type; (ii) both have a production-consumtion behavior. The user gives – in a (classical) Ptolemy model or in an HLA federate – the (static) information about who produces and who consumes. In a Ptolemy model, the communication via ports is represented by a link between two actors

A_1 and A_2 (Fig. 1): The A_1 output port sends the data and the A_2 input port receives the data. In an HLA federation, a federate F_1 (as in Fig. 2) must indicate that it publishes an attribute of a class and federate F_2 must indicate it subscribes to this attribute. Besides this static information, let us recall that an HLA federate has two more steps: (a) *After* launching, each object instance is *registered* once by the producer and *discovered* by the consumer; (b) During the simulation, the attribute of an object instance is *updated* by the producer and *reflected* by the consumer; this step occurs each time there is a new sent value. These steps are provided by the Ptolemy-HLA framework – and hidden from the user – making it easier for the user to distribute a simulation.

To establish a relationship between a token and an object class attribute, two new actors, `HlaPublisher` and `HlaSubscriber`, are added to the Ptolemy-HLA framework. They are depicted in Fig. 5a and b with parameters `Class` (Signal) and `Attribute` (val) according to the FOM (`.fed` file in Fig. 5c). The type of the ports corresponds to the type of the attribute. As events have a timestamp, time is involved and so the `HlaManager` interface needs to interact with them. Each actor has two roles: The `HlaPublisher` registers the object instance and sends the data through the RTI; The `HlaSubscriber` discovers the object instance and receives the data from the RTI. This is transparent to the user, that must connect the input port of an actor (receiving data from the RTI) to an `HlaSubscriber` actor, and connect the output port of an actor (sending data through the RTI) to an `HlaPublisher` actor (see federation F14 in Fig. 8).



a) `HlaPublisher` b) `HlaSubscriber`

```
;;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 val reliable timestamp))))
```

c) `.fed` file (FOM).

Fig. 5. `HlaPublisher` and `HlaSubscriber` icons in accordance with a FOM.

Data Sent by Ptolemy Through the RTI: When the earliest event $e(t)$ in the Ptolemy queue is the input of an `HlaPublisher` actor, the DE director first advances its time to t (as explained in Sect. 3.1), and then executes the `HlaPublisher` actor. Its execution consists of sending an update to the federation *at t or as soon as possible*, by calling the `HLA` service `UAV`. It means that the `HlaPublisher` actor provides a mechanism for (possibly) delaying an event (as, e.g., `TimeDelay` does) when necessary. But when is this necessary? Remember that a federate with lookahead *lah* and current time t cannot send any message before $t+lah$, which delimits a *forbidden* zone (see Sect. 2.2). So, if an `HlaPublisher` actor is fired at current time t, if $t < f(h + lah)$ (inside the forbidden zone), it will

send a UAV($g(t) + lah$); otherwise, it will send a UAV(t). It is worth mentioning the values of the Ptolemy and HLA timelines (t, h) can be different during a simulation, as seen in Sect. 3.1. This can happen in particular at the firing of an HlaPublisher at t, and depends on the time management used:
– TAR: they can be different as represented in Fig. 3a: $(t, h = n * TS \leq g(t))$, $n = 1..k$, k given by Eq. 1, with UAV given by Eq. (3) in Table 2;
– NER: they are the *same*, modulo the time conversion, as represented in Fig. 3b: $(t, h = g(t))$ if no RAV is received or $(t = f(h), h)$ otherwise, with UAV given by Eq. (2) in Table 2.

Table 2. UAV sent by an HlaPublisher.

NER	$UAV(g(t) + lah)$		(2)
TAR	$UAV(g(t) + lah)$	if $g(t) < h + lah$	(3)
	$UAV(g(t))$	otherwise	

Table 3. Event received by an HlaSubscriber.

NER	$e(f(h''))$	(4)
TAR	$e(f(h + TS))$	(5)

Data Received by Ptolemy from the RTI: The data *reception* is started by the arrival of an RAV callback *during* the advance time phase (see Sect. 2.2). Algorithms 1 and 2 are extended to the Algorithms 3 and 4 to take into account the data arrival.

In the Ptolemy-HLA framework, the activation of an HlaSubscriber actor at t follows the reception of a RAV(h'') event received from the RTI (corresponding to a UAV(h'') sent by another federate). The HlaSubscriber activation date depends on the time management used: Eqs. 4 and 5 in Table 3 describe how an HlaSubscriber adds an event from an RAV callback when using NER or TAR respectively.

There is no delay added in the reception in a NER federate (Algorithm 3), but a delay up to an HLA time step can be added in a TAR federate (Algorithm 4), as can be seen in Federation F_d in the sequel. Why is, in a TAR federate, an RAV(h'') callback not translated into an event in the calendar queue at time f(h'') by an HLASubscriber actor, as in a NER federate? The reason is the following: an RAV(h''), $h'' \leq h + TS$, is received, when a TAR federate is waiting for a TAG($h + TS$) (after it asked to advance its time with TAR($h + TS$)). So, at the RAV(h'') reception, the federate is still at h. But if an event $e(f(h''),$ HlaSubs) is put in the queue, and if this actor is directly or indirectly connected to an HLAPublisher, an UAV(g(f(h'')) would be sent through the RTI. This breaks another HLA rule, saying that a federate that did a TAR(h^*) cannot send any UAV message before $h^* + TS$. This is why, in our framework, an RAV(h'') is translated into an event timestamped $e(f(h + TS),$ HlaSubscriber).

Algorithm 3. NER `proposeTime`(t') taking RAVs into account

```
1: if g(t') > h then
2:     NER(g(t'))
3:     while not granted do
4:         TICK()                    ▷ Wait TAG(h'')
5:     end while
6:     h ← h''                       ▷ Update HLA time
7:     if receivedRAV then
8:         t'' ← f(h'')
9:         if t'' > t then           ▷ General case
10:            t' ← t''
11:        else
12:            t' ← t + r
13:        end if
14:        putRAVonHlaSubs(t')
15:    end if
16: end if
17: return t'                        ▷ Update PtII time
```

Algorithm 4. TAR `proposeTime`(t') taking RAVs into account

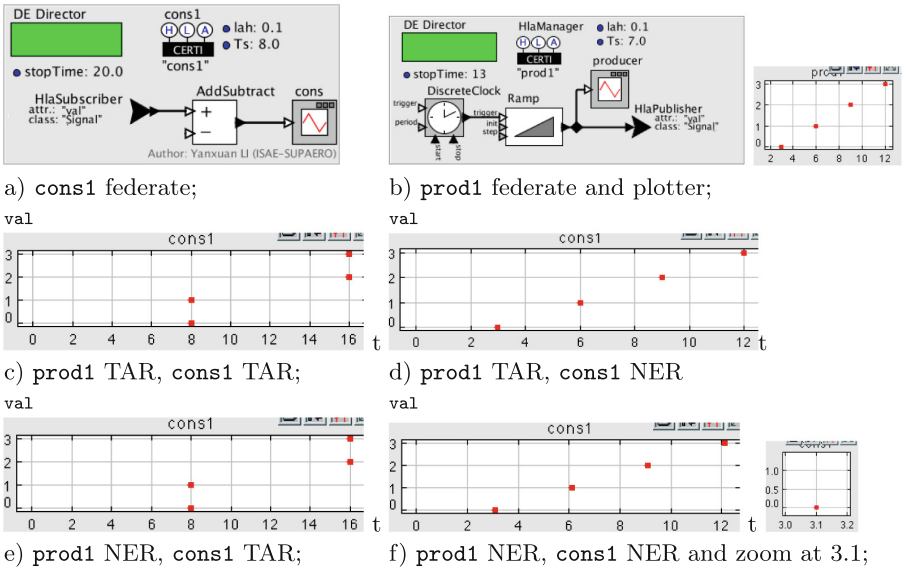
```
1: while g(t') > h + TS do
2:     TAR(h + TS)
3:     while not granted do
4:         TICK()                    ▷ Wait TAG
5:     end while
6:     h ← h + TS                    ▷ Update HLA time
7:     if receivedRAV then
8:         t'' ← f(h)
9:         if t'' < t' then
10:            t' ← t''
11:        end if
12:        putRAVonHlaSubs(t')
13:        return t'                 ▷ Update PtII time
14:    end if
15: end while
16: return t'                        ▷ Update to asked PtII t'
```

Assume there is a Federation $\mathtt{F}_d$ with two federates `cons1` and `prod1`, as depicted in Fig. 6a and b respectively. Their HLA time steps are different ($\mathrm{TS}_{cons1} = 8$ and $\mathrm{TS}_{prod1} = 7$) and so are their end of simulation times (20.0 for `cons1` and 13.0 for `prod1`). Both have $t_0 = 0_T$ and $h_0 = 0_d$ and the same lookahed $lah = 0.1$. Federate `prod1` publishes `val`: the input of `HlaPublisher(Signal.val)` is connected to the Ramp actor that produces events $e(t)$ with timestamps $3_T, 6_T, 9_T, 12_T$ depicted in Fig. 6b. As seen in Sect. 3.2, the timestamp of the UAV sent to the RTI depends on the time management. Federate `cons1` subscribes to attribute `val` of class `Signal` using the `HlaSubscriber(Signal.val)` actor. Figures 6c–f show the plotter in the `cons1` federate when the federates use different combinations of time management.

- *Both* `prod1` *and* `cons1` *use TAR* (*the result is depicted in* Fig. 6c)

The earliest event of `cons1` is $e(20_T)$, the end of simulation; from Algorithm 3, a `TAR`(8) is called, since $g(20_T) > h + 8_d$. The earliest event of `prod1` is $e(0; 3_T)$; from Algorithm 3, no TAR is called and its time is advanced to $t = 3_T$, since $g(3_T) < h + 7_d$. The `HlaPublisher` is executed, and according to Eq. (3) in Table 2, a UAV$(0, g(3_T))$ is sent.

The same behavior appears for the next event in `prod1`, $e(1; 6_T)$, and UAV$(0, g(6_T))$ is sent. But a `TAR`(7_d) will be called for event $e(2, 9_T)$ and UAV$(0, g(9_T))$ is sent. When `prod1` reaches the end of simulation, `cons1` is the only federate and the RTI sends a `TAG`(8_d). As indicated in Eq. 5 in Table 3, the RAV$(0, g(3_T))$ and RAV$(1, g(6_T))$ are then put in the queue as events $e(0, (f(8_d), 1)$, `HlaSubscriber`) and $e(1, (f(8_d), 2)$, `HlaSubscriber`). Notice that they have the same timestamp $f(8_d)$ and different microsteps. After `TAR`(16_d), `prod1` will receive `RAV`$(2, g(9_T))$ and `RAV`$(3, g(12_T))$, the events $e(2, (f(16_d), 1)$, `HlaSubs`) and $e(3, (f(16_d), 2)$, `HlaSubsc`) are generated. The entire exchange is presented in Fig. 7. This figure can be compared to Fig. 3. Notice that the left



a) `cons1` federate; b) `prod1` federate and plotter;

c) `prod1` TAR, `cons1` TAR; d) `prod1` TAR, `cons1` NER

e) `prod1` NER, `cons1` TAR; f) `prod1` NER, `cons1` NER and zoom at 3.1;

Fig. 6. Federation F_d (a,b); plotter at `cons1` (c to f).

side of Fig. 7 is similar to Fig. 3a, because `prod1` has internal events and Ptolemy wants to advance its time to values others than $k * TS$.

- `prod1`=*TAR and* `cons1`=*NER* (*the result is depicted in* Fig. 6d)

The behavior of `prod1` is the same as above. At `cons1`, according to Eq. 4 in Table 3, no delay is added to the received RAVs and the translated Ptolemy events are: $e(0, g(3_T))$, $e(1, g(6_T))$, $e(2, g(9_T))$ and $e(3, g(12_T))$.

- `prod1`=*NER and* `cons1`=*TAR* (*the result is depicted in* Fig. 6e)

When `prod1` advances to $t = 3_T$, a $UAV(0, g(3_T) + lah) = UAV(0, g(3.1_T))$ is sent according to Eq. (2) in Table 2 ($lah = 0.1$). As `cons1` uses TAR, the corresponding $RAV(0, g(3_T + lah))$ generates an event $e(0, (f(8_d), 1)$, `HlaSubscriber`), since $g(3_T) + lah < TS = 8_d$. As $g(6_T) + lah < TS = 8_d$, $RAV(1, g(6_T) + lah)$ generates $e(1, (f(8_T), 2)$, `HlaSubscriber`). Notice that Fig. 6e and c are the same.

- *Both* `prod1` *and* `cons1` *use NER* (*the result is depicted in* Fig. 6f)

According to Eq. 4 in Table 3, no delay is added in the received RAV; `prod1` sent a $UAV(0, 3.1)$, and `cons1` will receive a $RAV(0, 3.1)$ that is translated to $e(0, f(g(3.1)))$ as can be seen in Fig. 6f. The other events are $e(1, f(g(6.1)))$, $e(2, f(g(9.1)))$ and $e(3, f(g(12.1)))$.

These examples point out that the user needs to carefully analyze the semantics of the models. This will be discussed in the following.

3.3 Zooming in on the Boundaries

During the waiting phase of a TAG, many RAVs can be received by a federate (see Algorithms 3 and 4, lines 3–5). These RAVs are memorized in a FIFO.

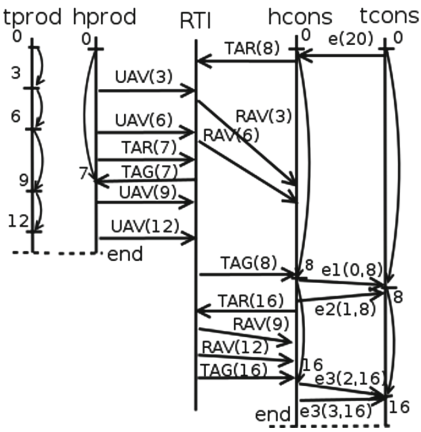


Fig. 7. F_d: `cons1`/TAR (TS = 8) + `prod1`/TAR (TS = 7).

After the while loop, a unique Ptolemy time t' is calculated for the firing of the corresponding HLASubscriber actor. The function `putRAVonHlaSubscribers`(t') empties this FIFO and adds event $e_j(t'$, `HlaSubs`) to the Ptolemy calendar queue.

How many RAVs can be received? Without lack of generality, let us consider the RAVs from the same instance of a class.

In a TAR federate, k RAVs can be received with different timestamps h'', all $h'' \leq h + TS$, and all will be translated to events $e_j(f(h + TS), n_j)$, `HlaSubs`), $j = 1..k$, with increasing microsteps. The order in which the RAVs are received is maintained using microsteps. This occurs, for example, in Federation F_d, Fig. 6c and e. This can happen when the federates have different rhythms in the data exchanges: different HLA time steps when using only TAR, or a NER federate sending data to a TAR federate.

Let us consider, for example, the reception of different values from the same sensor. In this case, the freshest value is more useful, and the solution could be to insert only one event $e(f(h + TS, 1)$, `HlaSubs`) corresponding to the freshest k^{th} RAV. Moreover, the calculation of a new state and, in general, a new output (e.g., the control of an actuator in a CPS) can be meaningful for just one value at a time t.

However, when designing a model, there are always elements where the user should use good software design patterns to ensure the model has the right semantics. One way to tackle the reception of k RAVs(h'') is to add a clock to the federate (subscribing to these attributes) that will dictate the wanted rhythm for the calculation [20]. Another way to tackle the RAV reception is to focus on the sending of the UAV. From the HLA point of view, it is not relevant to send several UAVs for the *same object instance at the same date h*. For example, before sending a UAV of an attribute that is subscribed to by an actuator, a `MostRecent` actor can be inserted in the input of the `HlaPublisher` such that only the freshest event will be sent.

Some tests have been added to Algorithms 3 and 4 for taking into account that the time representation of Ptolemy (t) and HLA (h) are different.

The first one is in Algorithm 3 of `proposeTime`, line 12 (NER federates): Because of the needed conversions, it can happen that $f(h'') = t$, e.g., $f(10_d + \epsilon) = 10_T$. In this case, inserting a new event at $e(t,$`HlaSubs`$)$ can be a problem to the director, since all existing events $e(t,$`HlaSubs`$)$ have already been executed. Our choice was, in this particular case, to add the time resolution r to t, since it is the shortest value that we can add for advancing the time beyond t. Let us point out that the microstep is used and correctly taken into account in the other cases as depicted in Fig. 6c and e.

The second one is in Algorithm 4 `proposeTime` (TAR federates), and makes the algorithm robust when dealing with a very particular case. This case can happen when t' and $f(h+TS)$ have the *same* (mathematical) values, e.g., $t' = 10_T$ and $h+TS = 10_d$, but $f(h+TS) > t'$. Let us recall two points: (1) The `proposeTime` method in the TimeRegulator interface of a Ptolemy model must return the same proposed time or a smaller time; (2) HLA guarantees that the timestamp of an RAV will never be smaller than the current time h (or larger than $h+TS$). So, if $f(h+TS)$ is bigger than the initial t', caused by the conversion, then the returned time must be t'. Otherwise, the Ptolemy time would advance to a time larger than timestamp t', when there is still an event $e(t')$ in the queue, and this event would be in the past.

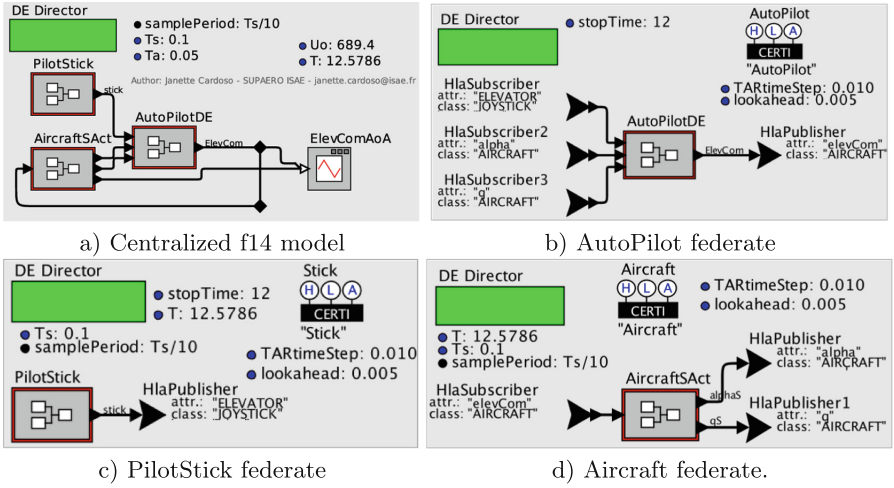
Another issue is the following: Is it possible to produce an *internal* event in a Ptolemy federate `f` with the same timestamp as an RAV-event received from another federate? Let us consider two federates `f1` and `f2` sending a data update: except for cases where these federates have exactly the same code (and same time representation), their UAVs rarely have the same timestamp, because $h_{\texttt{f1}}$ can be different from $h_{\texttt{f2}}$. In the general case where Ptolemy federates can interoperate with, e.g., C^{++} federates, this can be very difficult or even impossible to achieve, because of the different time representations. But even in a pure Ptolemy federation, because of the RTI time representation, it can still be impossible to have an internal event $t_{\texttt{f1}} = f(h_{\texttt{f1}})$.

Ptolemy can be downloaded from website [27]. The Ptolemy-HLA framework can be found at `$PTII/org/hlacerti`. The F14 demo presented in the sequel and others demos are provided, as well as a user guide. Some practical information can be found in the wiki [28].

4 Case Study: F14

Simulation is a very powerful way to perform validation, but one needs confidence in the results. The Ptolemy-HLA framework provides useful information, allowing for performance measures and simulation validation: simulation data (parameters of the federate e.g., name and time management); 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 and number of TAGs,

simulation execution time, execution time between services calls). This information appears in *.csv* text files generated during the simulation (if the user chooses this option) [3].



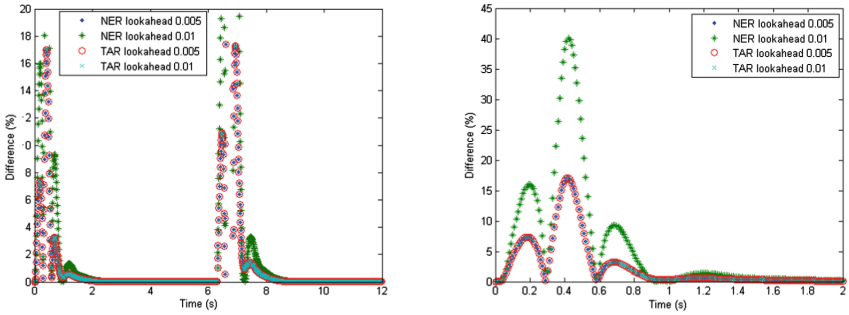
a) Centralized f14 model b) AutoPilot federate

c) PilotStick federate d) Aircraft federate.

Fig. 8. The F14 federation of a centralized model.

An aircraft is a very good example of a CPS. Figure 8a depicts a Ptolemy model of a F14 aircraft derived from a Simulink demo: `PilotStick` and `Aircraft` have a Continuous director and `AutoPilot` uses a DE MoC [17]. Both continuous models have a Sampler actor with sampling time 10ms, that provides the input for `AutoPilot`, and `Aircraft` has a ZeroOrderHold actor in its input. This model was split up into the three federates represented in Fig. 8b, c and d. Taking advantage of the interoperability of HLA, the pilot stick simulated in Fig. 8c was later successfully replaced with a real pilot stick. The results presented in this section are for the federation with the simulated pilot stick. The first step is checking the simulation results of the distributed model against those of the centralized model. Figure 9a shows the simulation results for federate `AutoPilot` and attribute `elevCom`, comparing the *centralized* and the *distributed* simulation results of the F14 Federation depicted in Fig. 8. The results were obtained using NER and TAR (with HLA time step `TS` = 0.010) and two values of the lookahead (0.005 and 0.010). The error is almost zero in steady state and smaller than 17% using TAR or NER with the smaller lookahead (0.005).

Concerning the performance related to the time management used by the federates: Fig. 10 shows the number of time advancement requests in the federate Aircraft using TAR and NER as a function of the HLA time step `TS`. As expected, NER is constant, since it does not depend on the HLA time step. Concerning TAR, the number of time advance requests is the same as NER when the HLA time step `TS` is equal to the sampling time of Sampler actor (10 ms). For values



(a) `elevCom` [0, 12s] (b) Zoom on `elevCom` over [0, 2s]

Fig. 9. Relative difference between *centralized* and *distributed* simulation.

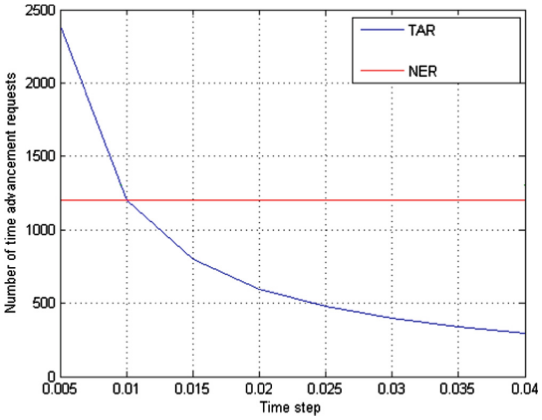


Fig. 10. Time advance requests in the federate Aircraft using TAR and NER.

of `TS` <10 ms, the performance of the TAR simulation is worse than in the NER case, but the accuracy of the simulation is better. The opposite occurs when `TS` >10 ms: better performance for TAR as opposed to NER, but worst accuracy, as expected.

5 Conclusion

In this paper, we present Ptolemy-HLA, a distributed simulation framework for complex and heterogeneous systems such as encountered in CPS. We have combined Ptolemy – which allows for heterogeneous systems simulation with a clean time representation – and the use of the HLA standard – which allows for Distributed Discrete Event Simulation (DDES) and interoperability of simulations. We hope this framework gathers the benefits of both. Moreover, Ptolemy and CERTI, an HLA-compliant RTI, are open-source, and so is Ptolemy-HLA.

A key feature in Ptolemy-HLA is that it can be easily installed and experimented with, without requiring in depth knowledge of HLA or the (quite complex) DDES domain. Collaborative contributions by other researches would be welcome.

Topics addressed in this work deal mainly with the existence of different timelines across distributed components and their coordination. The way time is advanced in the framework is carefully discussed and algorithms are presented: first, without data exchange, and then, the more general case that includes data exchange. We also present the way Ptolemy tokens and HLA attributes are translated into one another, taking into account the time advance and the conversion between the two timelines. Other features are implemented in the framework but are not discussed in this paper, such as the use of an initial synchronization point that makes it easier to launch the federation, and the ability to manage several instances of a class (e.g., several f14 aircrafts flying in formation).

The framework presented here allows Ptolemy to be compliant with the HLA standard. Moreover, we think the issues discussed can be re-used for other software needing to be compliant with this standard. We have applied this framework to the study of some CPS. In this paper, we have presented the F14 distributed simulation and some results. We also implemented a federation simulating a fleet of quad-rotors using Ptolemy and MORSE [22], a generic simulator for academic robotics.

Future work include new applications and extensions to this framework. Section 3.3 discusses issues related to data exchange. The last version of HLA provides services for negotiating the rhythm of data exchange between federates. This could be implemented and may simplify the work for the user as well as optimize the performance. Other HLA features not yet used, are the notion of interactions, the ownership management of objects, and the optimized data distributed management (with the introduction of subscribing and publishing regions).

We hope that this research, finalized with a tool, will be useful to tackle the problem of coupling different simulations, and the problem of coupling and distributing real systems. HLA-FMI is a very promising coupling technology. FIDE, a Ptolemy-FMI framework [8] could be combined with the Ptolemy-HLA framework and provide an HLA-FMI coupling.

Since the beginning of this work in 2013, many contributors have been involved, and we want to thank them warmly in this alphabetical and not exhaustive list: Vandita Banka, Christopher Brooks, Tarciana Cabral de Brito Guerra, David Come, Patricia Derler, Maxim Ivanov, Sébastien Jaillant, Gilles Lasnier, Edward Lee, Yanxuan Li, Clément Michel, Claire Pagetti.

Acknowledgements. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

References

1. Adelantado, M., Bussenot, J.L., Rousselot, J.Y., Siron, P., Betoule, M.: HP-CERTI: towards a high performance, high availability open source RTI for composable simulations. In: Fall Simulation Interoperability Workshop, September 2004
2. Bieber, P., Siron, P.: Design and implementation of a distributed interactive simulation security architecture. In: 3rd IEEE International Workshop on Distributed Interactive Simulation and Real-Time Applications, October 1999
3. Cabral De Brito Guerra, T.: Performance analysis of the framework Ptolemy-HLA. Technical report, ISAE/DISC/RT2016/2, September 2016
4. Cardoso, J., Derler, P., Eidson, J.C., Lee, E.A., Matic, S., Yang Zhao, J.Z.: Modeling timed systems. In: Ptolemaeus, C. (ed.) System Design, Modeling, and Simulation using Ptolemy II. Ptolemy.org (2014). http://ptolemy.eecs.berkeley.edu/books/Systems/chapters/ModelingTimedSystems.pdf
5. Chandy, K.M., Misra, J.: Distributed simulation: a case study in design and verification of distributed programs. IEEE Trans. Softw. Eng. **SE–5**(5), 440–452 (1979)
6. Chaudron, J.B., Noulard, E., Siron, P.: Design and model-checking techniques applied to real-time RTI time management. In: Spring Simulation Interoperability Workshop, April 2011
7. Cremona, F., Lohstroh, M., Broman, D., Lee, E.A., Masin, M., Tripakis, S.: Hybrid co-simulation: it's about time. Softw. Syst. Model., 1–25 (2017)
8. Cremona, F., Lohstroh, M., Tripakis, S., Brooks, C., Lee, E.A.: FIDE - an FMI integrated development environment. In: Symposium on Applied Computing, April 2016. http://chess.eecs.berkeley.edu/pubs/1158.html
9. Derler, P., Lee, E.A., Vincentelli, A.S.: Modeling cyber-physical systems. Proc. IEEE **100**(1), 13–28 (2012)
10. Forwardsim (2017). http://www.forwardsim.com/products/hla-toolbox/
11. Fujimoto, R.M.: Zero lookahead and repeatability in the High Level Architecture. In: Spring Simulation Interoperability Workshop, March 1997
12. Fujimoto, R.M.: Time management in the high level architecture. SIMULATION **71**(6), 388–400 (1998)
13. Garro, A., Falcone, A.: On the integration of HLA and FMI for supporting interoperability and reusability in distributed simulation. In: Proceedings of the Symposium on Theory of Modeling and Simulation: DEVS Integrative MS Symposium, vol. 47, pp. 9–16, 04 2015
14. HLA for FMI. https://www.ds.tools/products/hla-and-dis-for-fmi/
15. IEEE: IEEE standard for modeling and simulation (M&S) High Level Architecture (HLA) - Framework and rules. IEEE Std 1516–2010 (Revision of IEEE Std 1516–2000), pp. 1–38, August 2010
16. Kuhl, F., Dahmann, J., Weatherly, R.: Creating Computer Simulation Systems: An Introduction to the High Level Architecture. Prentice Hall PTR, Upper Saddle River (2000)
17. Lasnier, G., Cardoso, J., Siron, P., Pagetti, C., Derler, P.: Distributed simulation of heterogeneous and real-time systems. In: Proceedings of the 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications, pp. 55–62. IEEE Computer Society (2013)
18. Li, Y.: A distributed simulation environment for cyber-physical systems. Technical report, ISAE-Supaero, September 2015
19. Mattern, F.: Efficient algorithms for distributed snapshots and global virtual time approximation. J. Parallel Distrib. Comput. **18**(4), 423–434 (1993)

20. Michel, C.: Distributed simulation of cyber-physical systems. Technical report, ISAE-Supaero, April 2017
21. Michel, C., Cardoso, J., Siron, P.: Time management of heterogeneous distributed simulation. In: 31st European Simulation and Modelling Conference, October 2017
22. MORSE. https://www.openrobots.org/morse/doc/latest/morse.html
23. Nägele, T., Hooman, J.: Co-simulation of cyber-physical systems using HLA. In: 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC), pp. 1–6, January 2017
24. Neema, H., Gohl, J., Lattmann, Z., Sztipanovits, J., Karsai, G., Neema, S., Bapty, T., Batteh, J., Tummescheit, H., Sureshkumar, C.: Model-based integration platform for FMI co-simulation and heterogeneous simulations of cyber-physical systems. In: Proceedings of the 10th International Modelica Conference, pp. 235–245, March 2014. https://modelica.org/events/modelica2014/proceedings/html/ProceedingsOfThe10thModelicaConference.pdf
25. Noulard, E., Rousselot, J.Y., Siron, P.: CERTI: an open source RTI, why and how. In: Spring Simulation Interoperability Workshop, March 2009
26. Cardoso, J., Derler, P., Eidson, J.C., Lee, E.A., Matic, S., Zhao, Y., Zou, J.: Modeling timed systems. In: Ptolemaeus, C. (ed.) System Design, Modeling, and Simulation Using Ptolemy II. Ptolemy.org (2014). http://ptolemy.eecs.berkeley.edu/books/Systems/chapters/Dataflow.pdf
27. Ptolemy source. http://ptolemy.eecs.berkeley.edu/ptolemyII/
28. Ptolemy-HLA. https://www.icyphy.org/hla/wiki/Main/PtII-hlacerti
29. Shrivastava, A., Derler, P., Baboudr, Y.S.L., Stanton, K., Khayatian, M., Andrade, H.A., Weiss, M., Eidson, J., Chandhoke, S.: Time in cyber-physical systems. In: 2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 1–10, October 2016