Time Management of Heterogeneous Distributed Simulation

Clément Michel, Janette Cardoso, Pierre Siron

TIME MANAGEMENT OF HETEROGENEOUS DISTRIBUTED SIMULATION

Clément Michel
Janette Cardoso
Pierre Siron
ISAE-SUPAERO, University of Toulouse
10 avenue Édouard Belin
BP 54032 - 31055 Toulouse CEDEX 4, France
Email: {firstname.lastname}@isae-supaero.fr

KEYWORDS
Aerospace, Distributed Processors, Model design, Discrete simulation, Simulation interfaces, Cyber-Physical Systems, Heterogeneous Systems, HLA, Distributed Simulation

ABSTRACT

Cyber-physical systems (CPS), by their very nature, mix continuous and discrete behavior and are modeled by heterogeneous components. Formal analysis cannot always handle such complex systems and simulation is a necessary step. In particular, distributed simulation is very useful for the validation of CPS for two main reasons: either the CPS itself is distributed (e.g., a fleet of UAVs) or the CPS is too complex and/or has too much models (e.g., an aircraft). We discuss in this paper the impact of distributing the simulation of a system: which are the rules that must be applied to guarantee a correct behavior between the different simulators? If a centralized simulation already exists (using Discrete Event simulation), which hypothesis must be made for the Distributed Discrete Event simulation? The co-simulation framework used and discussed in this work is Ptolemy-HLA. It allows a Ptolemy model to be distributed using the high-level architecture (HLA) standard.

INTRODUCTION

The analysis of cyber-physical systems (CPS) is a complex task due to the heterogeneity of the parts involved, as they integrate different methodologies and tools.
Simulating different parts of a CPS requires different abstraction and tool supports, and the lack of interoperability between tools poses a major challenge. Because of the nature of the CPS, or because of its complexity, distribution can be necessary.
Distributing a simulation brings its own challenges, as it requires all the simulation elements to conform to a collection of rules in order for the elements to communicate between them.

In this work, the Ptolemy framework is used for modeling the CPS system, and HLA is the standard chosen for the distribution. The IEEE High-Level Architecture (HLA) is a standard for distributed discrete-event (DDE) simulation. CERTI is a HLA-compliant RTI (Run Time Infrastructure). Ptolemy II is a modeling and simulation tool for heterogeneous systems, well suited for modeling CPS since it provides different models of computation (MoC). In this paper, the simulation entities, called **federates** are Ptolemy models, but the approach presented in this paper can be easily used for other simulators. We focus on the time representation issues introduced in a distributed simulation, using HLA as a standard for the co-simulation and Ptolemy as a simulator component. Interoperability and reuse are important targets, so the coupling between simulators is an important issue in the design. The first step in a distributed simulation is to find the partition of a simulation model consisting of a set of sub-models. Which sub-models can (or must) be put together in a *simulator* belonging to the distributed simulation? How to guarantee that the distributed simulation will be valid (according to some criteria)?
The purpose of this work is twofold: Providing properties that ensure a correct time coordination between a federate and HLA, and studying the impact of the distribution in the Ptolemy-HLA framework in order to build models requiring timely input. By studying this specific coupling, we intend to find general problems as well their solutions concerning the time management of heterogeneous distributed simulations.
We will start by presenting the characteristics of both Ptolemy and HLA. Then, we will discuss the impact of an HLA-based distribution, before introducing elements needed for the Ptolemy-HLA simulation to be conservative.

PTOLEMY

Ptolemy II is a Java open-source simulation and modeling tool intended for experimenting with system design techniques, particularly those that involve combinations of different types of models (Ptolemaeus 2014). A Ptolemy simulation, called model, is com-

posed of a Director and software components called actors, that execute concurrently and communicate through messages (called events) carrying values, sent via interconnected ports. An actor that is executed is said to be fired.

The collection of rules that governs concurrent execution of the actors and the communication between them is called a Model of Computation (MoC). The MoC for each actor is implemented as a Director, a software component that dictates how actors should be fired. In this paper, we focus on the Discrete Event (DE) and Continuous MoCs.
Ptolemy uses a model of time known as the superdense time, represented by a tuple (t, n), where t is called the model time and n is called the microstep. The model time represents the time at which some events occurs, and the microstep represents the sequencing of events that occur at the same model time (Manna and Pnueli 1993, Ptolemaeus 2014). The events are ordered in the event queue first by t, then by n.
An event e is noted $e((t, n), v)$ with (t, n) the timestamp and v the value of the event. To ensure determinism, the order in which actors are fired when multiple events are queued for the same timestamp (t, n) is given by the actor rank, a topological sort that lists the actors in data-precedence order.
In the DE MoC, a model advances its logical time to the timestamp of the next event in the queue, and the actor to whom this event is destined for is executed. In the Continuous MoC, the model advances its logical time to a timestamp computed by the solver (Runge-Kutta 23 or Runge-Kutta 45), and all actors are fired at once.
Ptolemy provides a so-called `TimeRegulator` interface with a `proposeTime` method (Ptolemaeus 2014). This interface is implemented by attributes that wish to be consulted when a Director advances time. The Director will call the `proposeTime` method, passing it a proposed time to advance to, and the method will return either the same proposed time or a smaller time.

HLA

The High-Level Architecture (HLA) is a standard for distributed discrete-event simulation. In HLA terminology, the entire system to be simulated is represented by a *federation*, which is a collection of *federates* (simulation entities or simulators).
The execution of a federation uses a middleware, called the *Runtime Infrastructure* (RTI). The federation have a *Federation Object Model* (FOM), a file that contains the definition of data structures (called objects) exchanged between the federates.
In a federation, messages are exchanged between the various federates through the RTI. Those messages, called events, can be timestamped. Messages that are time-stamped are said TSO for Time Stamp Order.

Among the services defined by the HLA standard (IEEE-SA Standards Board 2010), we will focus on object management and on time management.

The object management service allows for federates to exchange messages and values. We focus here on two functions: *Update attribute value (UAV)*, that sends a message to the federation, and *Reflect Attribute Value (RAV)*, that delivers a message from the federation.
A federate is said *regulating and constrained* when it both sends and receives TSO messages. When all the federates are regulating and constrained, the federation is said to be *conservatively synchronized* (Kuhl et al. 2000). Regulating federates generate TSO messages that must occur no earlier than $h_c + lah$, with h_c being the current HLA time for the federate, and lah being its lookahead, a value that establishes a lower bound on the timestamps that can be sent.
While a Discrete Event simulation simply advances its time to t when wanting to, a Distributed Discrete Event simulation first asks for the permission to advance to h, and only advances to h when granted its authorization.
A time constrained federate requests to move its logical time forward by first asking the RTI to do so, either through a *Next Event Request (NER)* or a *Time Advance Request (TAR)*. The RTI replies to this request by sending all TSO messages up to h to the federate, then sending back a *Time Advance Grant (TAG)*. The TAG is written $TAG(h)$ and is an "authorization" for the federate to advance to the logical time h.

TAR
Consider a federate is at time h and asks for a time h_1 through a $TAR(h_1)$; the next time value is always h_1 granted by $TAG(h_1)$. So $TAR(h_1) \rightarrow TAG(h_1)$. The figure 1 illustrates the reception of a $RAV(h')$ messages with $h < h' \leq h_1$ during the advancing phase to h_1: the logical time is not updated to h' and is eventually granted to h_1.

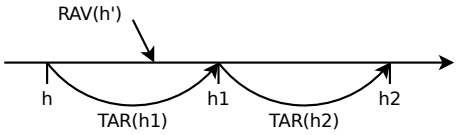


Figure 1: TAR Time Advance Policy Illustrated.

NER
Consider a federate is at time h and asks for a time h_1 through a $NER(h_1)$. If it receives a message $RAV(h')$ with $h < h' \leq h_1$, it advances its time to h':

$$NER(h_1) \rightarrow \begin{cases} TAG(h'), & \text{if } RAV(h') \\ TAG(h_1), & \text{if no } RAV(h') \end{cases}$$

Figure 2 illustrates this advance policy. If the federate is still willing to go to h_1, a new $NER(h_1)$ must be asked.

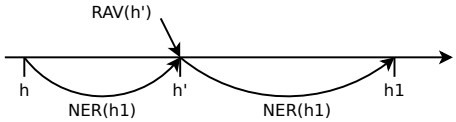


Figure 2: NER Time Advance Policy Illustrated.

IMPACT OF THE DISTRIBUTION AND COUPLING

To take part in a HLA federation, a simulator needs a HLA *coupling interface*, that handles the coupling between the simulator and HLA. Let us consider Figure 3 where two parts are depicted:

- Distribution: ruled by HLA standard that guarantees the time advancing is coherent for all federates (no simulator will advance to a time in the past, and TSO message are delivered in time-stamp order);
- Coupling: put in accordance the time advancing mechanism of the simulator with the HLA time advancing mechanism and so guarantee an ordered data exchange between the simulators in the federation.

The *object* management services (UAV, RAV) and the *time* management services (NER, TAR) are independent. However, the delivery of a RAV message is done during the time advancing phase.
Let be t be the simulation time and h the HLA time. The *distributed* simulation in Figure 3 is a Federation where Federate `f1` simulates a model `M1` and Federate `f2` simulates a model `M2`. Each federate has its own calendar queue, and events are sent/received through the RTI using HLA services UAV and RAV. The time is advanced using HLA services TAR (or NER).

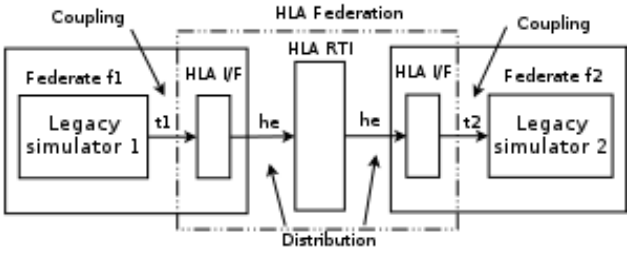


Figure 3: Distribution and coupling

Let be:

- t_1 the timestamp at which $f1$ wants to produce an event (to be sent through the RTI);
- h_e the timestamp of this corresponding event in the HLA service UAV sent through the RTI; it is also the timestamp of the callback RAV;
- t_2 the timestamp of an event put in `f2` calendar queue after the reception of the RAV from the RTI;
- h_c the current HLA time;
- lah the lookahead;
- TS the HLA time step, defined only when using TAR; $h_c + TS$ is called *next point in time*.

In such a distributed simulation, the event time stamp of a message can be altered twice, $t_1 \rightarrow h_e$ and $h_e \rightarrow t_2$, due to:

- Different time representations;
- the distribution itself and its rules, e.g. a federate cannot send an event earlier then h_c+ lookahead;
- the coupling *HLA* $\leftrightarrow$ *simulator* providing a coherent way for both time advancing mechanisms.

Indeed, a coupled distributed simulation does not come without a cost, and $t_2 > t_1$. In the next section, the Ptolemy-HLA coupling will be presented. Let us point out that in this paper we will focus on NER time management but TAR was also designed and implemented.

PTOLEMY-HLA

Ptolemy-HLA (Lasnier et al. 2013) aims to provide a distributed, conservative simulation. Both HLA and Ptolemy offer conservative time management options (Buck et al. 1994, Fujimoto 2003), then the interface binding them together must be conservative as well. In this section, we detail the different conditions needed for a Ptolemy-HLA conservative simulation.

Ptolemy/HLA Coupling Design

For a Ptolemy model to work as a HLA federate, the model's Director must be a DE Director. However, it can very well contain composite actors that themselves contain a Continuous Director each. Three Ptolemy components are added for Ptolemy and HLA to communicate:

- *HlaManager*: encompasses the high-level HLA rules and services such as time management and object management.
- *HlaSubscriber* actor: receives events from the HLA federation through RAVs (pictured on Figure 4b).
- *HlaPublisher* actor: sends Ptolemy events to the HLA federation through UAV service (pictured on Figure 4c).

Events circulating through the federation (even when not sent to another federate) must respect both HLA and Ptolemy time restrictions. In order to coordinate both Ptolemy time advancing and HLA time advancing, the `proposeTime` method was extended in order to allow Ptolemy to query the RTI for a time t. When a Ptolemy

federate wants to advance its time to the timestamp of the earliest event available, it first interrogates the RTI through the `proposeTime` algorithm, and wait for the RTI to grant it.
Algorithm 1 displays the `proposeTime` for the NER.

Algorithm 1 Extended `proposeTime` algorithm (NER)

```
1: NER(t_asked)
2: while TAG(h_received) not received do
3:     TICK()
4: end while
5: t_asked ← h_received
6: if RAV received then
7:     Schedule HlaSubscriber firing at t_asked
8: end if
9: return t_asked
```

The Ptolemy-HLA framework allows a federation (`f1`, `f2`) with any combination of federates time management: (NER,NER), (NER,TAR), (TAR, NER), (TAR,TAR). This is true also for a federation with more than two federates.
The events exchanged between the federates see their timestamp manipulated and shifted as depicted on Figure 3. In order to coordinate HLA and Ptolemy time, the coupling interface in the simulator has two time lines. In this section, we consider that both time lines have the same computer representation, so both can be directly compared.
In the sequel, let us present how the timestamp can change in the Ptolemy-HLA framework when sending and receiving an event according to the time management.

Sending a Ptolemy event through the RTI:
Let be t_{1_c} the current (Ptolemy) logical time of federate `f1` and h_{1_c} be the current HLA time. The federate wants to send an event $e(t_{1_c})$ (through the HlaPublisher actor) using a HLA UAV(h_1) service; timestamp h_1 depends on the federate's time management:

$$\text{NER: } h_1 = h_{1_c} + lah, \text{with } h_{1_c} = t_{1_c} \tag{1a}$$

$$\text{TAR: } h_1 = \begin{cases} h_{1_c} + lah, & \text{if } t_{1_c} < h_{1_c} + lah \\ t_{1_c}, & \text{otherwise} \end{cases} \tag{1b}$$

Receiving a RAV from the RTI:
Let be t_{2_c} the current (Ptolemy) logical time of federate `f2` and h_{2_c} be the current HLA time. The reception of the HLA RAV(h_1) service at h_{2_c} wakes up a HlaSubscriber actor with a Ptolemy event $e(t_2)$; the timestamp t_2 depends on the federate's time management:

$$\text{NER: } t_2 = h_1 \tag{2a}$$

$$\text{TAR: } t_2 = h_{2_c} + TS \tag{2b}$$

These rules are necessary in order to produce a conservative distributed simulation that respects the rules of both HLA and Ptolemy (i.e. no TSO message sent earlier than h_c+ *lookahead*, no event insertion in Ptolemy's past).
It can be seen from equations 1a, 1b, 2a and 2b that the difference $t_2 - t_{1_c}$ between the production of an event at time t_{1_c} in `f1` and its consumption at time t_2 at `f2` depends on the time management of each federate and its parameters:

$$t_2 - t_{1_c} = f(lah, (TS), t_{1_c}, h_{1_c}, h_{2_c}).$$

Let us consider again Figure 3, where both federates `f1` and `f2` use a NER time management with h_c current HLA time for both federates and a same lookahead *lah*. Notice that they can also have different lookaheads.
Let be t_{1_c} the current (Ptolemy) logical time and $e_{\text{f1}}(t_{1_c})$ the event that wakes up the HlaPublisher actor in `f1`. By using equation 1a, the corresponding UAV is sent with timestamp $h_1 = t_{1_c} + lah$ (regardless of `f2` time management). The RAV event with timestamp h_1 received by `f2` is queued at HlaSubscriber actor as a event e_{f2} with timestamp $t_2 = h_1$ given by equation 2a. Thus, the (NER,NER) configuration introduces in the distributed simulation a delay $\delta_{NER-NER} = t_2 - t_{1_c} = lah$.

Ptolemy/HLA coupling: implementation issues

Coupling Ptolemy with HLA requires to take into account that the time representation is different: CERTI RTI uses IEEE 754 `double` (with dynamic precision) and Ptolemy uses its own `Time` representation (fixed precision). These two time representations must be well converted.
Timestamps in Ptolemy are represented under the form $t = n * r$ with n an integer (called the *time value*) and r a Java *double* (called the *time resolution*). Thus, time values in Ptolemy can only adopt values that are multiple of r, e.g. r, $2r$, $3r$, etc...
Despite the time representation difference, one must guarantee that:

- Rule 1: No event should be inserted into the simulator's past — $t_{RAV} > t_{PtII}$
- Rule 2: No UAV should be sent in the past of the HLA federation — $h_{UAV} > h_{HLA}$

Non observance of these rules violates the causality of either the federation or the simulator, outputting wrong results.
At the earliest stages of Ptolemy-HLA, the question of the conversion between different (computer) representation was eluded, even if it was well-known that a function $\hat{f}$ such as $\hat{f}(h_{HLA}) = t_{PtII}$ and $\hat{f}^{-1}(t_{PtII}) = h_{HLA}$ was unobtainable.

In the following, the difference between HLA and Ptolemy (or any other simulator that does not use *double*) time representation is taken into account. In this work, we introduce two functions f and g such as:

$$\begin{cases} f(h_{HLA}) = t_{PtII} \\ g(t_{PtII}) = h'_{HLA} \end{cases}$$

Several properties are required from f and g in order to respect temporal coherence in TAR and NER cases, such as:

- f and g are monotonous strictly increasing functions
- $\forall h_1, h_2, h_3 \quad h_3 \geq h_2 > h_1 \Rightarrow h_3 \geq (g \circ f)(h_2) > h_1$
- $\forall t_1, t_2 \quad t_2 \geq t_1 \Rightarrow (f \circ g)(t_2) \geq t_1$.

We find these properties hard to fulfill from a mathematical point of view, even probably they cannot be fulfilled if f and g are not the identity function and this is not possible with heterogeneous systems. In such a context, we can find and use algorithmic solutions in order to use existing conversion functions and to solve the list of the identified problems.
Thus, each time that t and h must be compared in Algorithm 1, we introduce f and g, two monotonous, non-strictly increasing functions, with the hypothesis that algorithmic modifications will allow the functions to work in our context. Considering the f and g functions, Algorithm 1 is modified to Algorithm 2 in the following way:

- NER is called only if $g(t_{asked}) > h_{current}$: prevents Ptolemy from requesting a time advance if the requested time is imperceptible by the HLA time.
- $t_{asked} \leftarrow t_{current} + r$, if $f(h_{received}) \leq t_{current}$, r being Ptolemy's resolution: prevents HLA from granting a time advance imperceptible by the Ptolemy time.

Equations 1a, 1b, 2a and 2b are also modified in order to take f and g into account. For instance, 1a becomes $h_1 =, h_{1_c} + lah$, with $h_{1_c} = g(t_{1_c})$.
Finally, we keep the association between the return of a function and its argument, i.e. when a value $t_1 = f(h_1)$ is received, the association between t_1 and h_1 is memorized, and we will output h_1 when evaluating $g(t_1)$.
Algorithm 2 ensures a framework where rules 1 and 2 are observed.
Because of this difference in HLA and Ptolemy time representation, the theoretical delay induced by the distribution described in the previous sub-section can be slightly altered by a value ϵ introduced by the conversion between the time representations.

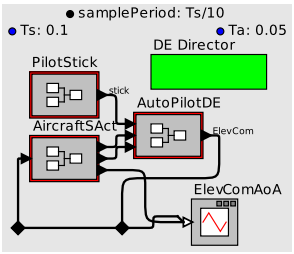
APPLICATION

A full case study of a longitudinal flight control (Lasnier et al. 2013) will be considered in this section.

Algorithm 2 Modified `ProposeTime` Algorithm (NER)

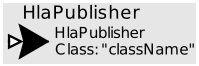
```
1:  if g(t_asked) > h_current then
2:      NER(g(t_asked))
3:      while TAG(h_received) not received do
4:          TICK()
5:      end while
6:      if RAV received then
7:          if f(h_received) > t_current then
8:              t_asked ← f(h_received)
9:          else
10:             t_asked ← t_current + r
11:         end if
12:         Schedule HlaSubscriber firing at t_asked
13:     end if
14: end if
15: return t_asked
```

Centralized simulation



(a) Centralized F-14.

(b) HlaSubscriber



(c) HlaPublisher

Figure 4: F-14

The centralized (hierarchical and heterogeneous) Ptolemy model, based on a Matlab model of an F-14 aircraft, is pictured on Figure 4a. `Aircraft` and `Stick` are composite actors modeled in the continuous MoC and the `AutoPilotDE` composite actor (controller) uses a DE MoC. As in a real aircraft, signals in the continuous domain need to be sampled in order to be used in the DE domain.
Let us highlight the model of the `AutoPilotDE` block. The block comports one output `ElevCom` that is the elevation command sent to the `Aircraft` block, and takes in three inputs:

- `stickS`: The current action on the stick.
- `alphaS`: The current vertical velocity of the F-14.
- `qS`: The current pitch rate of the F-14.

If only one of the inputs receives an event, the `AutoPilotDE` block will consider the others inputs as absent. Thus, receiving the events at three different timestamps would trigger three different rounds of computation that would output incorrect `elevCom` values. As a consequence, the events concerning variables `stickS`, `alphaS` and `qS` must be fed at the same timestamp (t, n) for the output to be correct. Thanks to the topological sort, we are guaranteed to have all these events

produced by the `PeriodicSampler` before actors within `AutoPilotDE` are fired, processing the three events at once.

Distributed simulation

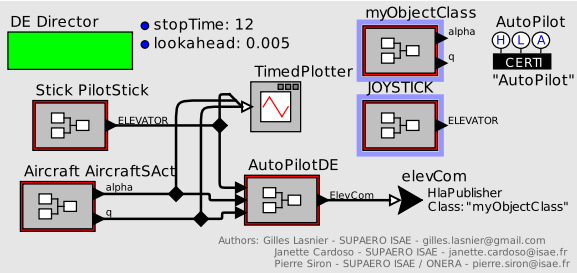


Figure 5: Federate `f14AutoPilotDE`.

A distributed version of the F-14 is obtained by creating a new federate model for each of the three actors in Figure 4a: `PilotStick`, `AutoPilotDE`, and `PilotStick`. For each actor, an input port is connected to an `HlaSubscriber` (pictured on Figure 4b) and an output port is connected to an `HlaPublisher` (pictured on Figure 4c). The `f14AutoPilotDE` federate, containing the `AutoPilotDE` block, is pictured on Figure 5. The `Stick PilotStick` and `Aircraft AircraftSAct` are composite actors containing the `HlaSubscriber` receiving events, respectively, from `f14PilotStick` and `f14Aircraft` federates, while the `elevCom` actor is an `HlaPublisher` sending events to the federation.
The three federates use the NER time advancing mechanism with a same lookahead lah.
Let us consider $t_{stick}, t_{alpha}, t_q$ the timestamps of the events arriving, respectively, at inputs `stickS`, `alphaS` (generated by the `f14Aircraft` federate) and `qS` (generated by the `f14PilotStick` federate).
According to equation 1a, the UAVs at `f14Aircraft` and `f14PilotStick` federate are sent with timestamp $h = g(t + lah)$, with h the current HLA time for the federate when the event is produced. The RAVs received at `f14AutoPilotDE` federate are then queued as events e with timestamps $t' = f(h)$ according to equation 2a.
For the `AutoPilotDE` block inside `f14AutoPilotDE` federate to work properly, events for all the inputs are required to be received at the same time. Thus, $t'_{stick} = t'_{alpha} = t'_q$. By expressing t' as a function of t, we obtain the following condition:

$$f(g(t_{stick}) + lah) = f(g(t_{alpha}) + lah) = f(g(t_q) + lah)$$

Since the three federates are Ptolemy federates, they have the same f, g functions. Moreover, as all the federates have the same lah, we can reduce the condition to

$$t_{stick} = t_{alpha} = t_q$$

This condition is satisfied as the `PeriodicSamplers` (actors sampling the output of continuous models in `Aircraft` and `Stick` blocks in Figure 4a) outputs all the events at the same time since they have the same period. Thus, the `AutoPilotDE` block receives and processes its inputs the same way than in the centralized model, albeit in a time-shifted manner. The bias lah introduced by the distribution does not, in this case, generate significant changes in the simulation behavior. So, the `AutoPilotDE` block maintains its expected behavior under some conditions on the other federates.

RELATED WORK

In (Deschamps et al. 2017), the partitioning of a simulation is discussed, addressing the simultaneity at the inputs of all components (of a distributed simulation). A formalism is proposed to prove the equality of delays by design and so the distributed simulation is valid. The case study of an f14 aircraft is presented in (Michel 2017) where, for a given partition, the *cyber* component is analyzed to guarantee that the data consumptions are simultaneous using different implementations: memory blocs, internal clock.
An aspect in distributed simulation is the multiplicity of times and the different representation of these time values. One must guarantee that the times advancing of the distributed simulation are done correctly (e.g., conservation simulation (Fujimoto 2003)). The time representation of each simulator, and the entity that allows for co-simulation – HLA or master algorithm in FMI – must be well dealt in the coupling between them. A detailed analysis of time representation in the FMI framework is done in (Cremona et al. 2017). In particular this paper discuss the choice of resolution to be used when the FMUs (components of a co-simulation) have different resolutions. The coordination of different times issue appears also in real cyber-physical systems (Shrivastava et al. 2016).
In (Nägele and Hooman 2017), a HLA/simulator coupling, called wrapper, is modeled using POOSL (Parallel Object-Oriented Specification Language). They use PoRTIco, a HLA compliant RTI and their federates are also regulating and constrained.

CONCLUSION AND PERSPECTIVES

Ptolemy-HLA framework allows to run valid distributed simulations for Event-Driven (NER) and Time-Stepped (TAR) advancing mechanisms. For a same federate model, the user can choose several parameters such as the time advancing mechanism (TAR or NER) and the lookahead. The framework implementation as well several demos are available at (The Ptolemy Project 2017). The coupling algorithm between Ptolemy and HLA in the case of NER mechanism was detailed in this paper. For highlighting a valid coupling, first we consider that

the (computer) representation of its timelines are the same, then we extended the algorithm for taking their representation difference into account. The TAR mechanism was also implemented in (The Ptolemy Project 2017). In both cases a time bias between the federates, expressed by equations 1a to 2b, is introduced but the algorithm guarantees an valid and efficient distribution. When considering different time representation, the bias can be slightly increased by a value ϵ introduced by the time conversion.

The `f14` Federation presented in this paper was obtained from the centralized model following the characteristics of a CPS: a federate for the cyber part (the controller), a federate for the plant (the aircraft), and a federate for the pilot stick that can be replaced by a real hardware in this federation. However, the distribution of a model needs some thought process about the model. In a centralized model some hypotheses are implicit, for instance the reception of all data generated by uphill actors. This constraint is ensured by the topological sort handled by Ptolemy that determines a deterministic execution order. This execution order can also be ensured in a distributed context, as well as the (logical) simultaneity of the inbound events of an actor, even considering the timestamp modification introduced by HLA. An analysis must be performed by the user to ensure that the chosen distribution is correct since a simulation can be distributed according to different mappings. For helping the user to make this analysis, the next step is the design of an extra "layer" that both helps the user to distribute a model and ensure that the centralized model is correctly distributed taking into account the bias introducing by the distribution and the coupling.

ACKNOWLEDGMENT

This research was partly supported by the French Ministry of Defense through financial support of the Direction Générale de l'Armement.

REFERENCES

Buck J.T.; Ha S.; Lee E.A.; and Messerschmitt D.G., 1994. *Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. International Journal of Computer Simulation*, 4, 155–182.

Cremona F.; Lohstroh M.; Broman D.; Tripakis S.; and Lee E.A., 2017. *Hybrid Co-Simulation: It's About Time.* Tech. Rep. UCB/EECS-2017-6, University of California at Berkeley.

Deschamps H.; Siron P.; Cardoso J.; and Cappello G., 2017. *Toward a Formalism to Study the Scheduling of Cyber-Physical Systems Simulations.* In *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT) (DS-RT'17).* Rome, Italy.

Fujimoto R.M., 2003. *Parallel Simulation: Distributed Simulation Systems.* In *35th Conference on Winter Simulation.* Winter Simulation Conference, WSC '03. ISBN 978-0-7803-8132-2, 124–134.

IEEE-SA Standards Board, 2010. *IEEE Standard for Modeling and Simulation (M & S) High Level Architecture (HLA): Federate Interface Specification.* Institute of Electrical and Electronics Engineers, New York. ISBN 978-0-7381-6247-8.

Kuhl F.; Dahmann J.; and Weatherly R., 2000. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture.* Prentice Hall PTR, Upper Saddle River, NJ. ISBN 978-0-13-022511-5.

Lasnier G.; Cardoso J.; Siron P.; Pagetti C.; and Derler P., 2013. *Distributed Simulation of Heterogeneous and Real-Time Systems.* In *IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications.* IEEE Computer Society, 55–62.

Manna Z. and Pnueli A., 1993. *Verifying Hybrid Systems.* In *Hybrid Systems*, Springer, Berlin, Heidelberg, Lecture Notes in Computer Science. ISBN 978-3-540-57318-0 978-3-540-48060-0, 4–35. doi:10.1007/3-540-57318-6_22.

Michel C., 2017. *Distributed Simulation of Cyber-Physical Systems.* Tech. rep., ESIEA, ISAE-SUPAERO.

Nägele T. and Hooman J., 2017. *Co-Simulation of Cyber-Physical Systems Using HLA.* In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC).* 1–6. doi:10.1109/CCWC.2017.7868401.

Ptolemaeus C. (Ed.), 2014. *System Design, Modeling, and Simulation Using Ptolemy II.* Ptolemy.org.

Shrivastava A.; Derler P.; Baboudr Y.S.L.; Stanton K.; Khayatian M.; Andrade H.A.; Weiss M.; Eidson J.; and Chandhoke S., 2016. *Time in Cyber-Physical Systems.* In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS).* 1–10.

The Ptolemy Project, 2017. *Ptolemy Project Home Page.* `https://ptolemy.eecs.berkeley.edu/`.