Institute for Space and Aeronautics Engineering (ISAE) - SUPAERO
RTRA STAE Foundation

POSTDOCTORAL RESEARCH REPORT

Advisor : Janette Cardoso (ISAE/DMIA)
Co-Advisor : Claire Pagetti (ONERA/DTIM)
Scientific referent : Pierre Siron (ONERA/DTIM & ISAE/DMIA)

Toward a Distributed and Deterministic Framework to Design Cyber-PhysicalSystems

Gilles LASNIER

Institute for Space and Aeronautics Engineering (ISAE - DMIA
10 avenue Édouard-Belin BP 54032
31055 Toulouse Cedex 4

Toulouse, France, October, 2013

Contents

Acknowledgement

I would like to thank my advisors Janette Cardoso, Claire Pagetti and Pierre Siron for their valuable collaboration during this one-year (Oct 2013 -> Sept 2014) post-doctoral research in Toulouse. I would like to thank ISAE Foundation that provided the funding to stay 6 weeks at the CHESS team at Berkeley Univ.

I would like also to thank Edward Lee from Berkeley Univ for receiving me so nicely in his team. I truly enjoyed the fruitful discussions and the collaboration to create a new interface to perform this co-simulation framework. During this stay, I had the support from Christopher Brooks, Patricia Derler and Marten Loshstroh: many thanks to them. Thanks also to Stavros Tripakis for useful discussions.

Chapter 1

Introduction

1.1 Context

A Cyber-Physical System (CPS) is a system composed of computational units, networks, and physical processes. Embedded controllers and networks monitor and control the physical processes. Feedback loops allow the physical processes affect computations and vice versa. Today, CPS can be found in various domains such as spatial, avionics, automotive, transportation, healthcare, etc. Systems involved in these domains are mainly referred to the distributed realtime and embedded system familly.

Designing real-time and distributed cyber physical systems (CPS) is a complex task which typically entails the use of many different methodologies and tools in different areas and at different stages of development process. The figure 1.1, extracts from the http://cyberphysicalsystems.org/ website, summarizes in an elegant way the CPS design complexities through a concept map.

Typically, different parts of the CPS are developed by different engineering teams or even external contractors. Integrating these heterogeneous parts and evaluating the composite behavior is challenging and, with today's tools, a systematic composition is nearly impossible. Over the last years, model-based tools have gained momentum in the design of CPS as they allow for modeling at higher levels of abstraction and they facilitate the communication between different teams of engineers. Additionally, many tools come with add-ons such as formal verification and code-generation. However, different parts of a CPS require different abstractions and tool support. The lack of interoperability between the tools poses a major challenge.

For control-command systems, Matlab/Simulink [16] is often used to specify the plant dynamics and the control. Thus from the specification it is possible to simulate the behaviour and verify the robustness and stability of the functions by observing the functional traces. This step occurs early in the development process. The control functions are then translated into low level code (automatically or manually) that is then executed on the target architecture (e.g. a distributed platform composed of several computing nodes connected with a real-time operating system). Analysis of the execution on the target occurs very late in the development process. It is then of great importance to provide methods and tools to give ways to analyze, at least partially and as early as possible, the behaviors of the future implementation taking into account, for instance, the mapping on the target or the interaction with some physical devices (e.g. sensors and actuators).

The process of integrating components and evaluating their interaction with the target architecture requires knowledge of several domains (e.g. hardware, software) that are typically described in heterogeneous models (e.g. continuous models, discrete models, timed properties). Analyzing worst case execution times is insufficient in order to validate the system.

Formal analysis cannot usually handle systems of that complexity. As a result, simulations are performed to analyze the functional behaviour of high level specifications mixed with more low level elements. Imagine, for in-

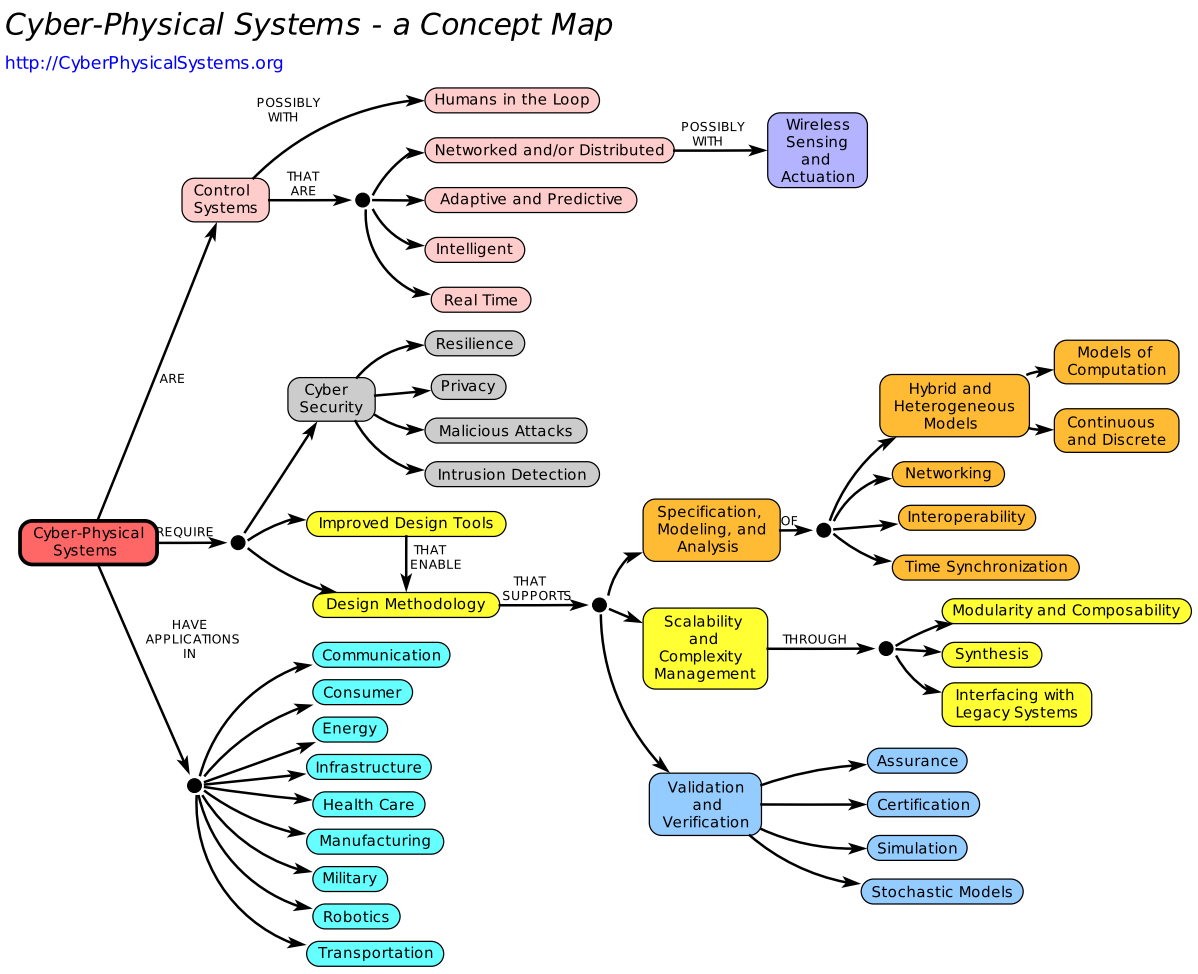


Figure 1.1: Concept map of cyber-physical systems - http://cyberphysicalsystems.org/

stance, a flight control composed of two sub-functions distributed on two calculators communicating via an Ethernet network. To observe the real behavior on the platform, it is necessary to describe the functional behavior of each component as well as the timing requirements of the platform and the network that communicates the data between the two sub-functions.

1.2 Contribution

The purpose of this research work is to provide an integrated methodology for distributed and heterogeneous simulation of realtime CPS where simulation components may be designed at different levels in the developments process. Components may be models (with different abstraction levels), software functionnalities (e.g. controllers) or hardware materials (e.g. embedded computers, sensors or actuators...).

Gather multiple simulation components evolving at different abstraction level to build a complex simulation tackles the complexity of the heterogenous aspect and the interoperability aspect of CPS design describes in the previous subsection. Distributed simulation is required here to enhance performance as weel as to design distributed models that we claim more close to the distributed architecture of the CPS.

Our framework is qualified as “co-simulation” framework according to the following definition :

Definition 1 (Co-simulation)**.** *Co-simulation (or co-operation) means a simulation methodology that allows individual components to be simulated by different simulators running simultaneously and exchanging data in a coordinate*

manner.

In this report, all references to "co-simulation" or "co-operation" will refer to the definition 1.

Our co-simulation framework has been design around the two main aspects: distributed and heterogenous simulation. To do so, the simulation tool leverages two open source tools: Ptolemy II and HLA/CERTI.

Ptolemy II [4] 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 [3] by providing different models of computation (MoC) such as continuous time for describing physical properties or discrete events for describing software and control.

The IEEE High-Level Architecture (HLA) standard [10, 9] targets distributed simulation. A CPS is 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, concrete functional codes (in C++, Java, etc.) and hardware equipments. The key benefits of HLA are interoperability and reuse. Our experiments and framework have been developed upon the HLA-compliant Open-Source RTI named CERTI [19, 20].

The co-simulation environment described here enable a Ptolemy simulation model to be executed as federate in a HLA federation. A Ptolemy federate exchange data with other federates that can be other Ptolemy simulations (i.e. only for a distributed purpose) or even C++ or Java federates (i.e. for heterogenous aspects).

The choosen strategy to enable the co-simulation is to provide a methodology to ease the development of ptolemy federate model, by the user, that could be directly integrated in a HLA federation. Main complexities such as time management, data communication and simulation execution (all discussed in this report) are handled at the framework-level. Therefore, a strong HLA-expertise is not required for the user.

According to this strategy and in order to facilitate the interaction between HLA and Ptolemy, we extend Ptolemy with dedicated components that enable the connection to HLA, the data exchange and which automatize the execution of a HLA Federation. The *HlaManager* centrally manages the advancement of time in HLA and Ptolemy. Two Ptolemy actors, an *HlaPublisher* and an *HlaSubscriber* are in charge of the data communication. Combining these two frameworks allows experimenting with: 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 equipments).

1.3 Related work

Interoperability is a very important issue in distributed discrete-event simulation as can be observed by several works targeting co-simulation. Closely related to our work are two particular cooperation tools that provide similar capabilities by providing HLA plugins.

In [8], the authors encode a connection between HLA and Modelica [17] for the virtual prototyping of mechatronic systems. In [5], the authors develop an HLA Blockset and an HLA Toolbox which provides a connection between HLA and MATLAB/Simulink. However, these solutions are not open source and the synchronization time between the federates is not described in literature. Our framework is open source, using existing open source frameworks and is intended for research, teaching, and industrial usage and several solutions for mixing the timing are explained in the current paper.

Related to this work in a broader context ins the work on Functional Mock-up Interfaces (FMI) [18], which is lead by a consortium of industrials and academics. The standard defines an interface standard for coupling different simulation tools in a co-simulation environment (e.g. by giving standardized access to simulation model equations). In this case, co-simulation is a simulation technique for coupled time-continuous and time-discrete systems. This work is rather recent and still undergoing various changes: a new version is currently being standardized [18]. Even if

part of our research is similar, FMI does not answer all our objectives presented above. The current standard presents limitations to manage discrete-event simulation properly [1] and no information is provided to address distributed simulation challenges.

This report is organized as follows. An overview of HLA and Ptolemy II are presented in Chapter 2 and chapter 3 (see also [12, 13]). Chapter 4 describes the co-simulation framework and shows how interoperability is obtained, chapter 5 presents the architecture design and implementation. A user manual is presented in chapter A. The installation guide and user guide are presented in chapter B. Finally, chapter 6 presents concluding remarks and our future work.

Chapter 2

The High Level Architecture Standard and the CERTI framework

This chapter presents the IEEE High Level Architecture standard (HLA) adressing distributed simulation complexities. We first give here a brief overview of the HLA standard and how an HLA simulation is implemented. We describe then the subset of HLA services necessary to interface a Ptolemy model with an HLA federation. This chapter focused on object management (shared data and data representation) and time management (time advancement semantics) in an HLA simulation.

2.1 Overview of HLA

The High-Level Architecture (HLA) [9, 10] is a standard for distributed discrete-event simulations, generally used to support analysis, engineering and training. The approach promotes reusability and interoperability. In HLA terminology, the entire system to be simulated is represented by a *federation* which is a collection of *federates*, i.e. simulation entities performing a sequence of computations. Federates are connected via the *Run-Time Infrastructure* (RTI), the underlying middleware functioning as the simulation kernel. Figure 2.1 describes the global architecture of a HLA simulation.

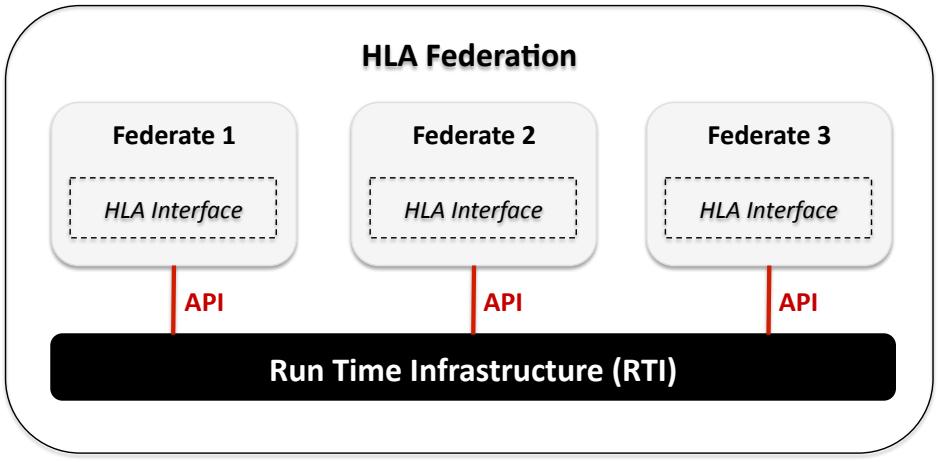


Figure 2.1: HLA Federation

The HLA specification defines:

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 (based on the OMT standard [11]) which provides a common framework for the communication between HLA simulations. For each federation, a Federation Object Model (FOM) describes the shared objects, interaction classes and their attribute.

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*.

Areas	Services	Description (non-formal)
Federation	createFederationExecution()	create a federation
	joinFederationExecution()	join a federation
	resignFederationExecution()	quit a federation
	destroyFederationExecution()	destroy a federation
	registerFederationSynchronizationPoint()	register a synchronization point
	synchronizationPointRegisterSucceeded()⋆	register synchro point succeeded
	announceSynchronizationPoint() ⋆	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
	subscribeObjectClassAttributes()	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 2.2: HLA services with a ⋆ are sent from RTI to Federates (callbacks); all other services are from Federates to RTI.

HLA services are grouped into six management areas which could be related to the federate life cycle. Figure 2.2 describes the subset of HLA services used for our approach. An informal description of each service role and a user documentation is available. The reader is referred to [10, 9] for a complete description of all HLA services.

The services in Figure 2.2 deal with the following areas: (1) Federation management: also includes a Federation Execution Data (FED) file used by the RTI to manage the whole federation; (2) Declaration management, i.e. which objects or *object attributes* each federate will publish or subscribe to; (3) Object management, i.e. the way federates produce attribute updates or receive updated attributes from the federation; (4) Time management, i.e. the mechanisms required to implement time management policies and to negotiate time advances.

2.2 FOM and data management

The federation object model is usually specified in a Federeration Execution Data (FED) file.

2.3 Time Management

HLA time management services enable deterministic and reproducible distributed simulations. Each federate manages its own logical time and communicates this time to the RTI. The RTI ensures correct coordination of federates by advancing time coherently. Logical time is roughly equivalent to *"simulation time"* in the classical discrete event simulation literature, and is used to ensure that federates observe events in the same order [6]. Logical time is not necessarily mapped to real time.

2.3.1 Federate's time policies

HLA time policies describe the involvement of each federate in the progress of time. It may be necessary to map the progress of one federate to the progress of another. A *regulating* federate participates actively in the decisions for the progress of time. A *constrained* federate follows the time progress imposed by other federates. A combination of both policies is possible. As our approach deals with the synchronization of logical time from different simulation tools, only *regulating and constrained* federates are allowed.

To properly handle time progress and ensure causality, HLA defines Time-Stamp Ordered (TSO) events which are supposed to occur at specific points in time. Regulating federates generate TSO events (possibly out of time-stamp order) that must occur no earlier than the current local time plus the *lookahead*. The *lookahead* acts as a contract value which guarantees that the federate will not produce a TSO event earlier than its current local time plus *lookahead*.

The RTI handles the ordering of the TSO events generated by the federation. Each federate implements a priority time-stamp queue for TSO events. Events stored in the ordered queue are only delivered to the federate in a time advancement phase if their timestamps are between the current and the next federate local time (the granted time, see later).

2.3.2 Time progress

Time advancement requests by federates are made through two particular services (see Figure 2.2): the *timeAdvanceRequest* service (*TAR*) is used to implement time-stepped federates; the *nextEventRequest* service (*NER*), is used to implement event-based federates. The granted time is provided by *timeAdvanceGrant* (TAG).

The time advancement phase of a Federate F in HLA is a three-step process: 1) F sends a request using NER or TAR services; 2) F can receive *reflectAttributeValue* (RAV) callbacks (i.e. updated HLA attributes); 3) F waits for the granted time t_G (TAG). At the TAG(t_G) reception, the federate's local time will be advanced to t_G according to the made request (NER or TAR).

A part of the time advancement phase semantics is shown in Figure 2.3. FederateTAR (see figure 2.3.a) produces an event at time t_1, with *updateAttributeValue* (*UAV*). The current (logical) time is t_1; let us consider that the next event is e_2 with timestamp t_2, $t_2 = t_1 + \Delta$, $\Delta > 0$. The federate asks the RTI for a time progress with the invocation of *TAR(t_2)*. Until the reception of the *TAG(t_2)* callback, the logical time of the federate is stalled at t_1, and it can receive a *RAV(v,t_1')* callback. Timestamp t_1' is in the interval $]t_1, t_2]$. FederateTAR can realise a computation with the values received with *RAV*. The federate then receives *TAG(t_2)* and can increase its local time to t_2.

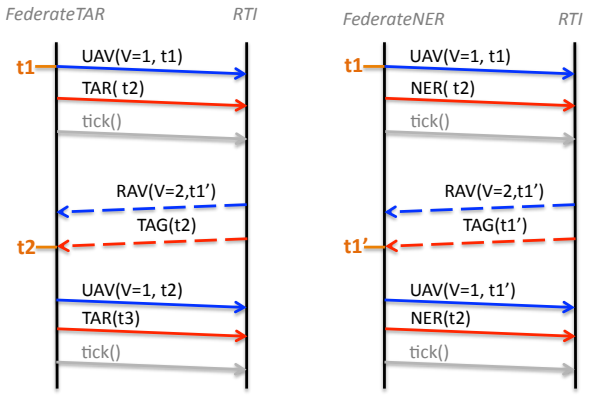


Figure 2.3: Time advancement services a. TAR b. NER

Let us now consider Figure 2.3.b. FederateNER also produces the event *UAV* at time t_1 and asks for a time advance to t_2, in that case with a *NER*(t_2). The reception of *RAV(v,t_1')* is followed by a *TAG*(t_1'). FederateNER moves its logical time to t_1' and then can realize a computation with the received value. Let us point out that a federate using the NER service is a more reactive federate since it can produce new events from time t_1'.

In a nutshell, if a TAR(t_2) has been sent, the time granted by the TAG service is $t_G = t_2$. If a NER(t_2) has been sent, the granted time is $t_G = t_1'$, with $t_1 < t_1' \leq t_2$. In the case of NER, if $t_1' < t_2$, the current federate have received one or more events from the federation with timestamp t_1'.

2.4 CERTI: an RTI implementation HLA-compliant

CERTI is a HLA-compliant open source RTI [19, 20]. More information about CERTI can be found here: https://savannah.nongnu.org/projects/certi

Chapter 3

The PtolemyII project

3.1 Overview of Ptolemy II

Ptolemy II [21, 4] is an open source modeling and simulation framework for *heterogeneous* systems. Ptolemy models are actor-oriented. *Actors* are executable and concurrent components that communicate via *ports*. Actors can be atomic or composite, where a composite actor contains an entire model inside but behaves like an atomic actor to the outside. A special model component, the director, describes the semantics of a model. Ptolemy supports a wide variety of models of computations, including discrete event (DE), continuous time (CT), synchronous reactive (SR) or synchronous data flow (SDF). The operational rules for executing a model are given by the MoC, defining the meaning of execution, concurrency and communication. These rules determine when actors perform internal computation, update their internal state, and perform external communication. Models in different MoCs can be composed hierarchically with a clearly defined semantics.

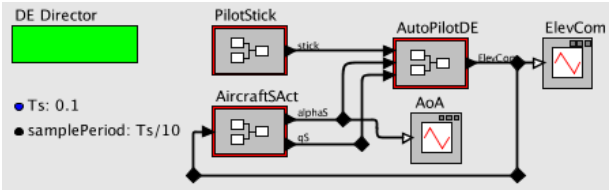


Figure 3.1: A hierarchical Ptolemy model of a CPS.

The ability to model heterogeneous systems in Ptolemy is interesting for CPS design as those typically comprise heterogeneous components expressed naturally in different MoCs. An example of a hierarchical Ptolemy model of a CPS – based on the F-14 Longitudinal Flight Control demo model from Matlab – is shown in Figure 3.1. `Aircraft` and `Stick` are composite actors modeled in the continuous MoC and the `AutoPilot` composite actor (controller) contains is a DE system. The top-level director is also a DE director. Heterogeneous composition of different MoCs sometimes requires special actors. For instance, signals in the continuous domain need to be sampled in order to be used in the DE domain. In Ptolemy, a Sampler actor needs to be placed. In order for continuous actors to read signals generated in the DE MoC, a ZeroOrderHold actor is used [15].

The next sub-section presents the semantics of the Ptolemy entities involved in our approach.

3.2 Ptolemy's entities semantics

3.2.1 Actor

An actor executes in three phases: a *setup* phase, a sequence of *iterations*, and a *wrapup* phase, as represented in Figure 3.2.

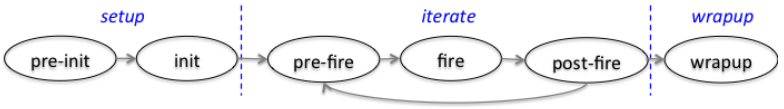


Figure 3.2: Actor's lifecycle in Ptolemy II

The *setup* phase of an actor is divided in two sub-phases: *preinitialize* and *initialize*. The preinitialize sub-phase is performed once, at the beginning of a simulation. It specifies actions that may modify or influence the actor architecture and consequently impact static analysis as well as resource deployment (composite actors may require the instantiation of internal actors), type resolution, schedulability, etc. The initialization sub-phase initializes parameters, resets local state, and sends initial messages on output ports.

An *iteration* is a sequence of operations that read input data, produce output data, and update the state. It is divided in three sub-phases (prefire, fire and postfire). The *prefire* (optionally) allows the verification of preconditions to ensure that the actor will complete the iteration. The *fire* sub-phase typically performs the computation of the actor. In this sub-phase, input data from ports is read, data is processed and output data is produced on output ports. An actor may have persistent state that evolves during execution. Finally, the *wrapup* phase performs the correct termination of the actor.

An actor comes with a set of primitive communication operations which allow to retrieve information from the communication channels (*get* method) or send information to the channels (*put* method). As the meaning of the communication is determined by the director, the connection between the external port of a composite actor and some other port on the outside will obey the semantics of the director on the outside. For example, in figure 3.1, the three composite actors must send/receive data obeying DE semantics imposed by the DE top level director.

3.2.2 Attribute

Attributes are used to store parameters and access them anywhere in the model.

The attribute's lifecycle is similar to the actor's one but contains only the setup and wrapup phases. Thus, an attribute can be used to statically or dynamically parameterize a model.

3.2.3 Decorator

Some actors and some attributes are implemented as so-called decorators. These entities can decorate other Ptolemy entities (e.g. model, actors, etc) with specific information in an 'aspect-oriented' way. Any number of attributes can be attached to a Ptolemy entity.

The remainder of this paper explains how a specific attribute may be designed to enable the time synchronization between both Ptolemy and HLA/CERTI at the director level and by preserving the MoC semantics.

3.2.4 Model and director

The director is responsible for the progress of the simulation by firing the actors in a specific order. In timed actors the model time (i.e. logical time) is advanced as part of this process. In a hierarchical model, only the top-level director advances time. The rules of time advancement depend of the (timed) MoC implemented by the director. Ptolemy II uses the same model of time for all MoC, known as *superdense time*. Superdense time is represented by a pair (t,n), called a timestamp, where t is a model time and n is a micro-step. The model time represents the time at which some event occurs, and the microstep represents the sequence of events that occur at the same model time. The superdense time semantics is defined in [14] and only an overview is given here.

In the case of a CT director, at time t_k, all actors are fired following a precise order and the next timestamp $t_k + 1$ is computed by the solver.

In the case of a DE director, an actor is only fired at the timestamp t if either it has an event in one of its input ports or it has explicitly asked to be fired at t (using the `fireAt` method). To ensure determinism, the order in which actors are fired is important. The DE director maintains a calendar queue of DE events and a global ordering between these events. A DE event is a tuple (*value*, *timestamp*), where the timestamp is a superdense time (t,n). Let us consider $e1 = (t_1,n_1)$ and $e2 = (t_2,n_2)$. The calendar queue is sorted:

1. by timestamp, if $t_1 < t_2$ or if $t_1 = t_2$ and $n_1 < n_2$ then $e1$ is executed before $e2$;
2. by actor's ranking, if $t_1 = t_2$ and $n_1 = n_2$, the director selects the actor that has the lowest rank.

The actor ranking is defined by a topological sort of the actors in the model in data-precedence order [14].

The time advancement phase in a DE model is observed when the DE director selects the earliest event in the calendar queue, making its timestamp the current model time (and firing the destination actor of the event). This algorithm is performed at the director level and during this phase no actor is fired.

Chapter 4

Co-Simulation for Heterogeneous and Distributed Simulation

The design of cyber-physical system model for simulation is a rigorous task where various models of the system, at different abstraction level, cooperate. These models can also be executed on different embedded platforms to hence the simulation performances or to produce relevant results with a deployment close to the real distributed architecture of the application.

Distributed simulation is useful for network modeling and simulation (exhibit non-determinism and complex behavior) and large-scale simulation. It enables the abstraction by separation of simulation tasks into different simulations (using standard interfaces), and increases the processing power of a simulation by scheduling tasks. But, distributed simulation also comes with complexities. Developing a distributed simulation architecture is more complex than a serial one and two majors issues are timing (synchronization of nodes) and control (how to maintain the control of simulation).

The co-simulation framework for heterogeneous and distributed simulation leverages two open source simulators: Ptolemy and HLA/CERTI. In chapter 2 and chapter 3 we have presented the time management semantics and the data communication operated by each simuation tools. In particular, both present the same characteristic to separate the time advancement phase and the data communication. In our case data communication means the events (timestamp + value) exchanged between nodes.

Considering the DE model in Ptolemy, event interaction between I/O ports are based on a token + receivers mecanism and actor are fired at specific point in time to process the event (i.e. the token). HLA is based on the publish/subscribe paradigm and two services, UAV and RAV, are provided for emission and reception of updated HLA attribute values.

DE event semantics is based on the implementation of a calendar queue where events are stored and sorted following a specific order (see chapter 3). The *time advancement phase* in DE correspond to take the earliest event from the queue and process it. The DE Director (behavior) is responsible of the queue management.

The next sections describe the strategy chosen for the co-simulation Ptolemy-HLA/CERTI and discuss about how the time management and the data communication between both frameworks are handled in our approach.

4.1 Co-simulation strategy

In HLA, the RTI implements the simulation control. An *HLA Federate* uses one of the 4 time advancement algorithms defined (TAR, NER, TARA or NERA) to declare a time advance request that is granted or not by the RTI

according to the semantics given to the request as presented in section 2.

Enabling distributed simulation using Ptolemy model, requires the definition and the implemention of an *entity* (distributed or not) to maintain the control of the simulation. As the RTI of HLA has been precisely implemented to achieve this task, we choose to use the RTI as the simulation control authority for the co-simulation. This choice brings us to consider only one strategy to enable the co-simulation between Ptolemy and HLA/CERTI. A Ptolemy model is slightly modified to become a Ptolemy federate participating in a HLA federation.

To adapt a Ptolemy model as a federate two approches have been considered. The first one is to realize a binding of HLA services for Ptolemy (as implemented in Forwardsim [5]). Thus, each HLA service is implemented as one or several actors or attributes. The main advantage of this approach is to give a maximum of flexibility to the the user to model a Ptolemy federate. The main drawbacks are that a user needs to have a strong expertise of HLA in addition to Ptolemy. On one hand, the Ptolemy initial model will be deeply modified and complexified and in another hand, the implementation work to enable HLA services as actors in Ptolemy is very important (required ressources and time).

The other approach is to develop a limited number of actors or attributes and restrict the adaptation of a Ptolemy model as the simple interactions which allow: the execution of time advance request, the execution of UAV service (to send event to the federation) and a mecanism to get callbacks from the RTI (to receive TAG and RAV services). This requires to fix the behavior of a Ptolemy federate, and to encapsulate the logic to execute HLA services in a high-level Ptolemy entity.

The main advantage of this approach is to simplify the adaptation of a Ptolemy model to be integrate as a federate in a HLA feeration. Ptolemy users with few knowledges of HLA may easily use the co-simulation to make distributed simulation between Ptolemy models or another HLA-compliant simulator. A configuration interface is provided to the user that allows to tune some HLA services as the ones from the time management.

The initial Ptolemy model chosen to co-operate with a HLA federation is a DE model. We have seen that the DE semantics is based on the one from discrete-event theory and shares similarities with the event-based federates implemented in HLA/CERTI. The current version of framework considers only DE model as the top-level model in Ptolemy. A continuous model can be inserted as a composite actor in a DE (top-)model using predefined actors that allow the interface between CT-DE and vice-versa. (see the demo `f14` in the Ptolemy projec). This is allowed by the hierarchical composition of Ptolemy models and specially between DE and CT domains presented in chapter 2.

4.2 Time management

As we pointed out in our motivation, dealing with co-simulation for distributed real-time and embedded systems requires to make consistent the different notion of times (logical time, platform time and realtime) involved in a simulation model, a simulation implementation or the physical world. In our case an event of the co-simuation, from Ptolemy as well as from HLA/CERTI, is a couple (timestamp, value) where the timestamp represents a specific point in time of the logical time of the simulators.

The time management is a fundamental issue in our co-simulation approach. To make event's timestamp consistent in both simulators - i.e. in Ptolemy and HLA/CERTI - it is necessary to understand precisely when occurs a time advancement phase in both approaches. These elements has already been presented in section 3 and section 2 and summarized in the introduction of this chapter.

4.2.1 Synchronization models

Figures 4.1 and 4.3 describe the synchronization models designed for the co-simulation framework. These synchronization models respects stricly the semantics of time management proposed in Ptolemy and HLA/CERTI. The

co-simulation supports the 4 semantics of time management services (TAR, NER, TARA and NERA) proposed by HLA.

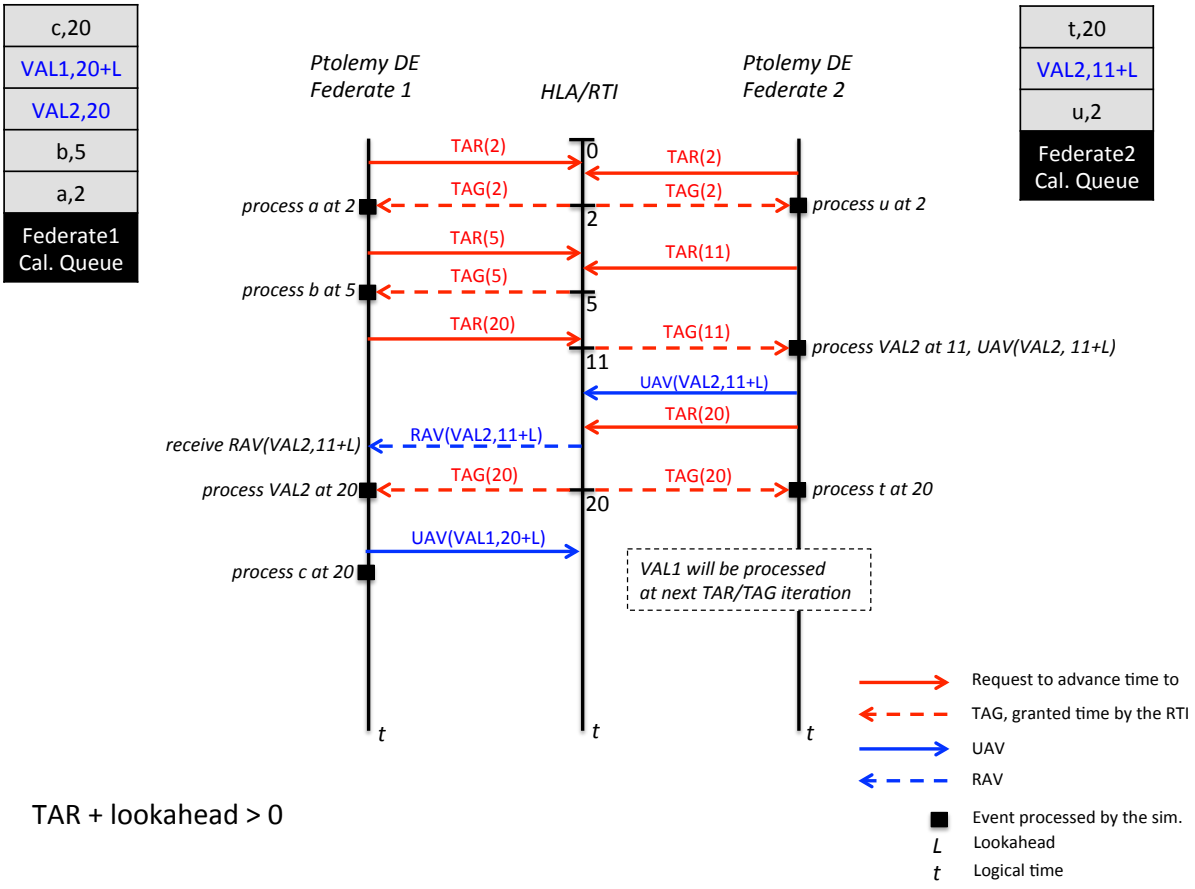


Figure 4.1: Synchronization model using TAR and lookahead > 0

Scenario For each synchronization model we propose the following scenario. Two Ptolemy DE federates, `Federate1` and `Federate2`, have to process "local" events[1]: (a,2), (b,5) and (c,20) for `Federate1` and (u,2), (t,20) for `Federate2`. `Federate2` publishes an updated value of event `VAL2` (at timestamp `11`) and susbcribes to `VAL1`. `Federate1` susbcribes to event `VAL2` and at the reception of the event `VAL2` publishes an updated value of event `VAL1`. Based on this simple scenario, the behavior of the co-simumation is different according to the time management services (TAR, NER, TARA or NERA) used by a federate.

4.2.2 Main behavior

The entrypoint is a request of time advancement made by the DE director of a Ptolemy federate. When the director consults its calendar queue and takes a new event to process - i.e. select the next actor to be fired - we perform a time advance request to the federation using one of the semantics (NER, TAR, NERA or TARA) provided by HLA. The timestamp of the next DE event to process is used as the time to advance to - i.e. the parameter - of the HLA request. During this phase, the DE director execution is blocked because of the active waiting loop (as described in section 2) implemented to wait the TAG callback from the RTI - i.e. the granted time to advance to authorized by the RTI. The RAV callbacks are received during the execution of the loop. At the reception of a TAG callback, the execution of the DE director is released and the timestamp of the TAG is returned to the director as the time to advance to.

[1] A local event is an event that is neither published to the federation nor received from the federation.

The table in fig. 4.2 summarizes the results of the execution of the co-simulation for the events `VAL1` and `VAL2` (exchanged between both federates via the federation) when using the different time advancement algorithms (TAR, NER, TARA, NERA). The use of the NERA or TARA services allow to send updated values of `VAL1` and `VAL2` without advancing the time of the simulation. This behavior is the same as the one specified by HLA.

4.3 Co-simulation requirements

In the previous sections of this chapter, we have defined the synchronization models and proposed a time advancement algorithm for our co-cosimulation framework. This section summarizes the different requirements needed to make effective the interface between a DE Ptolemy federate and an HLA/CERTI. These requirements are defined in three categories: interoperability, repeatability and determinism.

HLA-TM services	Events	`Federate1`	`Federate2`
TAR (L>0)	`VAL1`	UAV at 20 + L	process at next iteration TAR/TAG with t>20
	`VAL2`	process at 20	UAV at 11 + L
NER (L>0)	`VAL1`	UAV at 11 + L + L	process at 11 + L + L
	`VAL2`	process at 11 + L	UAV at 11 + L
TARA (L=0)	`VAL1`	UAV at 20	process at next iteration TAR/TAG with t>=20
	`VAL2`	process at 20	UAV at 11
NERA (L=0)	`VAL1`	UAV at 11	process at 11
	`VAL2`	process at 11	UAV at 11

Figure 4.2: Co-simulation behavior: `VAL1` and `VAL2` events processing, L = lookahead

In the next-subsection we give the algorithm to adapt the time advancement mecanism of a DE director to handle these 4 time advancement algorithms.

4.3.1 Time management algorithm for the co-simulation

In the previous subsection we have discussed the four time advancement algorithms which represent the different behaviors expected by the co-simulation Ptolemy - HLA/CERTI. To allow these behaviors it is necessary to adapt the DE director time management algorithm and to interface the Ptolemy model with an HLA/CERTI federation. To do so, we propose the algorithm 4. Algorithms 1, 2 and 3 are reminders of the initial time advancement algorithms for a Ptolemy DE model and an HLA federation (using TAR, NER, NERA or TARA services).

4.3.2 Interoperability

Requirements that belong to the interoperability's category are relatives to the simulation interface between a Ptolemy DE federate and a HLA simulation. The requirements, presented below, are a refinement of those presented in paper [HLArequirements] adapted for the needs of our co-simulation framework. Thus, a Ptolemy (federate) DE model must have or provide:

- R1. Ability to initialize the distributed simulation prior to simulation execution
- R2. Ability to block the simulation execution
- R3. Access to the time of the next event to be simulated
- R4. Ability to introduce new events from an external source into the event list

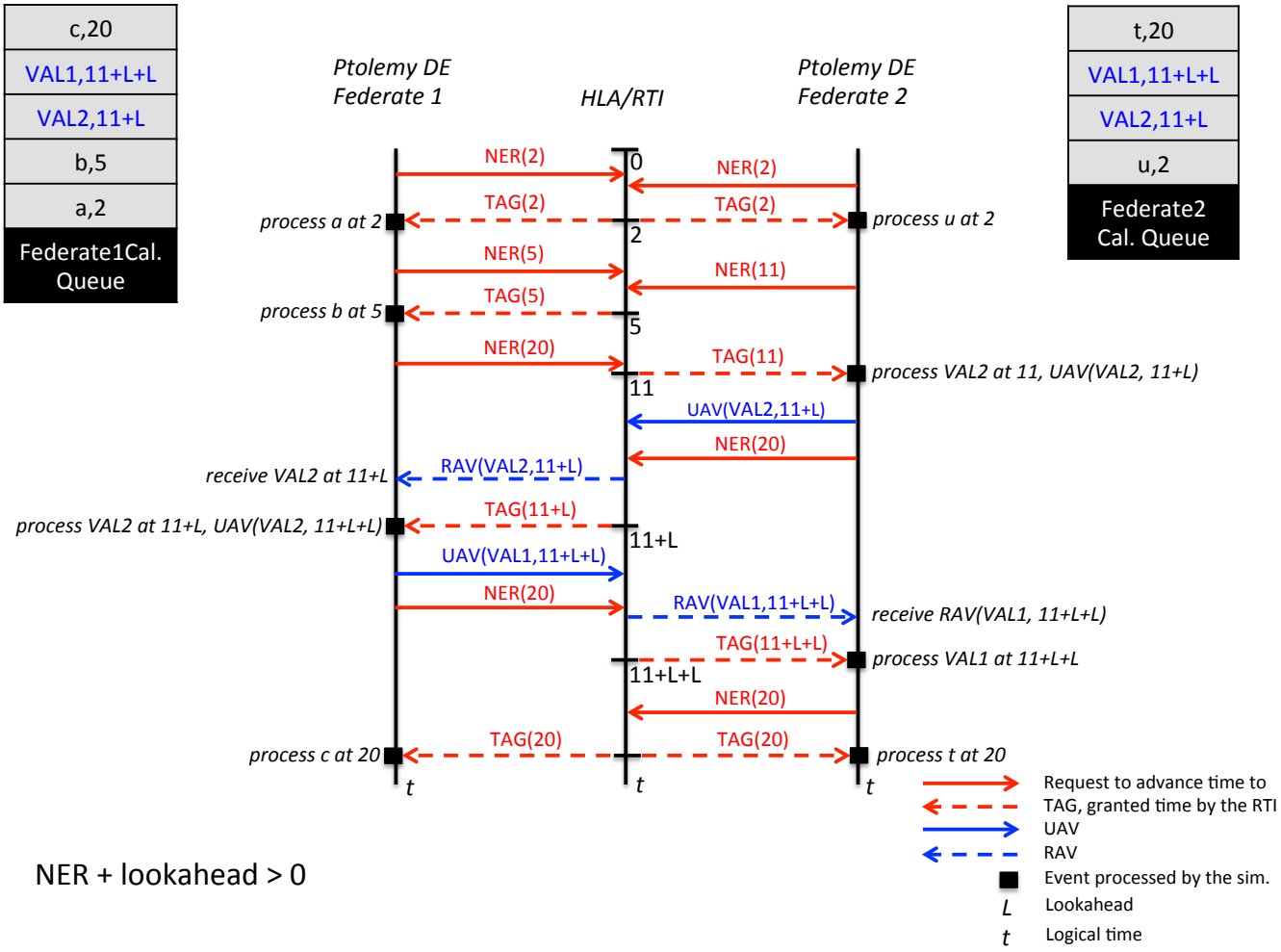


Figure 4.3: Synchronization model using NER and lookahead > 0

- R5. Access to information of simulation object/entities that are shared among federates

Requirements `R1` and `R2` are defined because a federate needs to first join the simulation and then blocks its execution until all the other federates join, before the simulation begins. The requirement `R3` is necessary to allow the utilisation of the NER service where the timestamp of the next event needs to be known. Requirement `R4` allows to handle new events received from other federates during the time advancement phase, while the federate is blocked and waits the time grant. These new events are merged into the DE calendar queue and handled as classical Ptolemy events. When the control is returned to the federate - i.e. when the DE director execution is released -, the events are simulated at their timestamp. Finally, the requirement `R5` is about the access of the data structure of events exchanged between federates. These information are required for building the correct representation of event's value in both simulators.

Data: *queue*, the calendar queue
Result: *currentTime*, time to advance to
select next event $e(t, dest)$ in *queue*;
set $currentTime \leftarrow t$;
– process all events at t for actor *dest*;

Algorithm 1: Ptolemy DE time advancement

Data: t, next time to advance to
Result: $currentTime$, granted time to advance to given by the RTI
call $NER(t)$ or $TAR(t)$;
while *not* $TAG(tg)$ **do**
  if $RAV(val,t')$ **then**
    store (val,t') – (t' =< tg =< t) ;
  end
end
set $currentTime \leftarrow tg$;

Algorithm 2: HLA time advancement: TAR or NER services

Data: t, next time to advance to
Result: $currentTime$, granted time to advance to given by the RTI
call $NERA(t)$ or $TARA(t)$;
while *not* $TAG(tg)$ **do**
  if $RAV(val,t')$ **then**
    store (val,t') – (t' =< tg =< t) ;
  end
end
for *number of NERA or TARA needed* **do**
  call $NERA(tg)$ or $TARA(tg)$;
  while *not* $TAG(tg)$ **do**
    if $RAV(val,tg)$ **then**
      store (val,tg) ;
    end
  end
end
call $NER(tg)$ or $TAR(tg)$;
while *not* $TAG(tg)$ **do**
  if $RAV(val,tg)$ **then**
    store (val,tg) ;
  end
end
set $currentTime \leftarrow tg$;

Algorithm 3: HLA time advancement: TARA and NERA services

4.3.3 Repeatability

The second category of requirements is about the repeatability in distributed simulation presented in section 2. We remind below the definition of repeatability in HLA.

Definition 2. *A distributed simulation is said to be repeatable if successive executions utilizing the same inputs produce exactly the same outputs [6].*

According to HLA, the RTI helps to support repeatability to the extent that it enables repeatable ordering of simultaneous events, but random number, floating point, etc, need to be handle by the user - i.e. remains to the federate implementation. A part of the repeatability requirement is satisfied in our co-simulation framework by the strict respect of the contracts defined in section 2 and section3. A proof to ensure repeatibility is also given there.

Based on the study of the HLA standard, the requirements for a Ptolemy federate to ensure repeatability are:

- R6. Ptolemy federate must process event in timestamp ordering
- R7. Ptolemy federate must handle simulateneous event ordering

[debut de preuve] Assuming that a federate handles simultaneous events, the repeatability is sastified in HLA for the following cases:

- Lookahead > 0: C1 + C3 or C2 + C3
- Lookahead = 0: C4.1 + C4.2

4.3.4 Determinism

The last requirement is about deterministic simulation. Determinism in the simulation of CPS is an important property. For our co-simulation framework we give the following definition:

Definition 3. *A distributed simulation is said to be deterministic if successive executions process the same inputs in the same order and produce exactly the same outputs.*

The main requirement for a Ptolemy federate to handle "determinism" is:

- R8. Ptolemy federate must process input events in order

4.4 Proposition of a co-simulation architecture

In the previous section we have studied the different behaviors and requirements needed for the co-simulation framework Ptolemy-HLA/CERTI. We have also proposed a time advancement algorithm that allows the use of the time management services provided by HLA. Thus, the user has the possibility to configure the framework according to the Ptolemy federate behavior that he wants to design.

From the analysis of the requirements and the strategy chosen to implement the co-simulation framework we can describe the following new elements for having a Ptolemy federate model:

- a simulation interface in Ptolemy to adapt the time advancement algorithm of the DE director;
- an entity (actor or attribute) to encapsulate the high-level logic of HLA (initilialization, creation, join, declaration, time management, etc.);
- a configuration interface to tune the HLA time management services;
- a ptolemy entity to send Ptolemy events to the HLA federation;
- a ptolemy entity to introduce events from the HLA federation in the Ptolemy model.

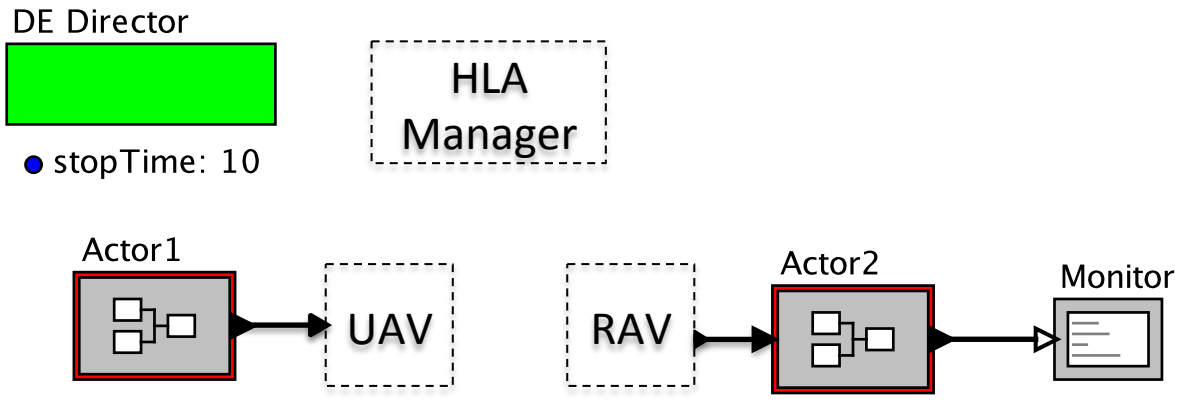


Figure 4.4: Proposition of the Co-Simulation architecture

These elements are defined and implemented as actors or attributes in ptolemy. Based on these needs we proposed the architecture presented in figure 4.4, for the co-simulation framework. A general simulation interface was designed that allows a director to modify its time advancement algorithm. This interface may be enabled by the deployment of a specific ptolemy entities in the model. The new framework has the following components:

HLA Manager This manager handles all the high-level logic of HLA and enables the interopoerability between a Ptolemy federate model and the HLA federation. This HLA Manager will also provide a configuration interface to tune the time management services required by the user. This interface will also allow the specification of others HLA information such as the name of the federate, the name of the federation, etc. This manager will also be responsible of the management of RAV callbacks received from the federation and leading to store a new event in the Ptolemy model and to inject this event in the DE queue.

Two specific actors: UAV and RAV As in HLA UAV and RAV are distinct services not executed at the same level (UAV during the *computation* phase of the federate, and RAV during the *time advancement* phase), we propose to implement two actors that allow or trigger these operations. The configuration of these actors will also specify the information relative to identify the HLA attribute to publish or to subscribe to (name and type). For example, the event send to the input port of the UAV may trigger the UAV service implemented in the HLA manager. A more complex mecanism is required for the RAV actors. RAV needs to be handled by the HLA manager. Ptolemy events produced from a RAV may be stored in its corresponding RAV actor and send through the output port of this actor at its timestamp. This mecanism may be implemented using the fireAt() method of an actor to program a re-firing time of the actor.

Data: *queue*, the DE calendar queue
Result: *currentTime*, granted time to advance to given by the RTI to the DE director
select next event $e(t, dest)$ in *queue*;
if *eventBased and* $L > 0$ **then** call NER(t);
;
else if *eventBased and* $L == 0$ **then** call NERA(t);
;
;
else if $timeStepped$ *and* $L > 0$ **then** call TAR(t);
;
;
else if $timeStepped$ *and* $L == 0$ **then** call TARA(t);
;
;
else raise HLA service configuration error;
while *not* $TAG(tg)$ **do**
    if $RAV(val, t')$ **then** store (val, t') and merge in queue;
    ;
end
if *eventBased and* $L == 0$ **then**
    for *number of NERA needed* **do**
        call $NERA(tg)$;
        while *not* $TAG(tg)$ **do**
            if $RAV(val, tg)$ **then** store (val, tg) and merge in queue;
            ;
        end
    end
    call $NER(tg)$;
    while *not* $TAG(tg)$ **do**
        if $RAV(val, tg)$ **then** store (val, tg) and merge in queue;
        ;
    end
    set $currentTime \leftarrow tg$;
end
if $timeStepped$ *and* $L == 0$ **then**
    for *number of TARA needed* **do**
        call $TARA(tg)$;
        while *not* $TAG(tg)$ **do**
            if $RAV(val, tg)$ **then** store (val, tg) and merge in queue;
            ;
        end
    end
    call $NER(tg)$;
    while *not* $TAG(tg)$ **do**
        if $RAV(val, tg)$ **then** store (val, tg) and merge in queue;
        ;
    end
    set $currentTime \leftarrow tg$;
end
if $L > 0$ **then** set $currentTime \leftarrow tg$;
;
return *currentTime* ;

Algorithm 4: DE time management algorithm for co-simulation

Chapter 5

Architecture design and implementation

5.1 Framework architecture

This section gives technical and implementation details about the architecture and the choices made to design and to implement our co-simulation framework. The framework makes interoperable two open-source simulation tools: PtolemyII and HLA/CERTI.

The strategy is focused on interfacing a Ptolemy simulation model with a HLA/CERTI Federation - e.g. integrate a Ptolemy simulation as a federate. in a federation. Our framework has been developped, integrated and released directly with the Ptolemy core project. More information on Ptolemy is provided at `http://chess.eecs.berkeley.edu/ptexternal/`.

The co-simulation framework is not a *simple* binding of the HLA/CERTI services for Ptolemy. As detailed in section 4, it was designed for handling properly time management, data communication and simulation execution between both simulators. The JCERTI java bindings for CERTI is used to interface Ptolemy federates implemented in Java with HLA/CERTI written in C++.

The framework architecture is built in two parts. The first part is the design and the implementation of a specific interface so called `TimeRegulator`. This interface allows for a Ptolemy director, in an elegant way, to consult a specific time advancement algorithm when it advances its time. The second part is the `ptolemy.apps.hlacerti` library that provides a set of classes wich realize the data exchanges, the time management between both the Ptolemy simulation model and the HLA/CERTI Federation. Additionnal features have been implemented to ease the design of Ptolemy federates in Vergil and the to automatize the simulation execution of a HLA Federation from Ptolemy.

Next sections describe this interface and the main entities implemented to realize the cooperation.

5.2 `TimeRegulator` interface

The `TimeRegulator` is a joint effort made during the visit of Dr G. Lasnier (ISAE) to the Ptolemy team at the University of California, Berkeley. The Java source code of this interface is given in Annex A-C.1 and is located in the `ptolemy.actor` java package in the Ptolemy project.

Ptolemy attributes belong to the global notion of decorators provided by Ptolemy and have already been presented in chapter 4. The attribute's lifecycle is similar to the actor's one but contains only the setup (`pre-initialize()` and `initialize()` methods) and wrapup (`wrapup()` method) phases. No iterate phase (`prefire()`, `fire()` or `postfire()` methods) is performed.

5.2.1 Ptolemy core modification

The `TimeRegulator` interface requires to be implemented by a Ptolemy attribute and allows to parameterize the time advancement algorithm of a director. Due to the number of directors implemented in Ptolemy, this interface has been designed to be introduced in a general (and friendly) way to each director (behavior) that may requires the use of those regulators.

Listing 5.1: Code snippets of ptolemy.actor.Director.java

```
/** Consult all attributes contained by the container of this director
 *  that implement the {@link TimeRegulator} interface, if any, and return the
 *  smallest time returned by those regulators. If there are no such
 *  attributes, return the proposedTime argument.
 *  @param proposedTime The time proposed.
 *  @return The smallest time returned by a TimeRegulator, or the
 *   proposedTime if none.
 *  @exception IllegalActionException If a time regulator throws it.
 */
protected Time _consultTimeRegulators(Time proposedTime) throws IllegalActionException
     {
    Time returnValue = proposedTime;
    List<TimeRegulator> regulators = getContainer().attributeList(TimeRegulator.class)
        ;
    for (TimeRegulator regulator : regulators) {
        Time modifiedTime = regulator.proposeTime(returnValue);
        if (modifiedTime.compareTo(returnValue) < 0) {
            returnValue = modifiedTime;
        }
    }
    return returnValue;
}
```

The listing 5.1 shows the modification of the high-level class `ptolemy.actor.Director` which implements a Director in Ptolemy. The private method `_consultTimeRegulators()` has been added and enables for a director to consult the *time regulator* Ptolemy attributes before they advance time. The call of this private method must be explictly made by the domain-specific director such as DE director, CT director, etc. The listing 5.2 shows this mechanism added to the method `_getNextActorToFire()` of the DE director.

Listing 5.2: Code snippets of ptolemy.domains.de.kernel.DEDirector.java

```
protected Actor _getNextActorToFire() throws IllegalActionException {

 // . . .

        if (actorToFire == null) {
            // If the actorToFire is not set yet,
            // find the actor associated with the event just found,
            // and update the current tag with the event tag.
            Time currentTime;
            int depth = 0;
            try {
                synchronized (_eventQueue) {
                    lastFoundEvent = _eventQueue.get();
                    currentTime = _consultTimeRegulators(lastFoundEvent
                            .timeStamp());
 // . . .
```

5.2.2 `proposeTime()` method

Each attribute that implements a `TimeRegulator` interface needs to provide an unique public method with the signature `proposeTime(Time proposedTime)`. This method implements the user time advancement algorithm. The only user-restriction to take care is:

Restriction 1. *The time value returned by a call to proposeTime() must be greater or equal to the director's currentTime.*

This constraint allows in most of the cases to preserve the semantics of the director's MoC.

The `proposeTime()` method has a parameter `proposedTime` which indicates the time the director wants to advance to. The semantic of this parameter is defined by the MoC of the director. For example, in the case of the DE director, the `proposedTime` is the timestamp of the next DE event to be consumed from its calendar queue.

As this method is always executed at the director level; *during its execution no actor involved in the simulation model is fired*. This important property prevents that any event may be produced during a time advancement phase. Assuming the correct implementation of the `proposeTime()` method by the user, this guarantees the preservation of the time advancement semantics of a particular MoC in Ptolemy.

An example of time advancement algorithm implemented in a `proposeTime()` method is given in the next section 5.3.

5.2.3 Usage of multiple time regulator attributes

Whenever a director has to advance its current time, if one or more *time regulators* attributes are attached to the top-level model, then the `proposeTime()` method of thoses attributes are executed before the time is advanced by the director. Only the smallest time returned by those attributes is returned to the director. If there is no such attribute, the initial time advancement algorithm (in Director.java) is performed.

An example of the use of different attributes attached to the same Ptolemy model is given in the `ptolemy.apps.hlacerti.demo` demo. This demo shows the behavior of the `SynchronizeToRealTime` attribute, which implements real-time synchronization with the execution platform, and the `HlaManager` attribute which implements the time management according to the HLA semantics. Both *time regulator* attributes are executed during the simulation.

5.3 HlaManager attribute

The HlaManager is the main entity of our framework which handles the time management and data communication mecanisms required between a Ptolemy federate and a HLA federation. This entity is implemented as a Time Regulator Ptolemy attribute (describes above) and provides a set of mecanisms and methods to realize the followng operations:

1. initialize a federation;
2. initialize a Ptolemy federate model as a regulator and/or contrained federate;
3. connect a Ptolemy federate model to a federation;
4. publish or subscribe to a HLA attribute values specified by the HLA actors deployed in the Ptolemy model;
5. synchonize the start of the Ptolemy federate execution according to other federates;

6. launch a federation (rtig and rtia processes);
7. quit and destroy properly a federation;
8. convert a Ptolemy token as HLA event (data+timestamp)
9. convert a HLA event (data+timestamp) as Ptolemy token (token + DE Event)
10. send an UAV to the federation
11. get a RAV from the federation
12. manage and normalize event timestamp from both simulators, according to the time management algorithm proposed in 4.

5.3.1 Architecture

The architecture of the HlaManager attribute is described in figure 5.1. 5 services are defined. All services implement one or several methods to realize one or several operations. The `Initialization` service refers to operations 1 to 6. The `Wrapup` service refers to operation 7. The `Data wrapper` service refers to operations 8 and 9. The `UAV` service refers to operation 10. The `Time management` service refers to operations 11 and 12. Finally, the `FederateAmbassador` service is a particular service that refers to all operations. According to the CERTI architecture, all information from the RTI are delivered to the federate with the use of a standardized HLA services implemented as callbacks. The `FederateAmbassador` service implements all the callbacks required for the different operations supported.

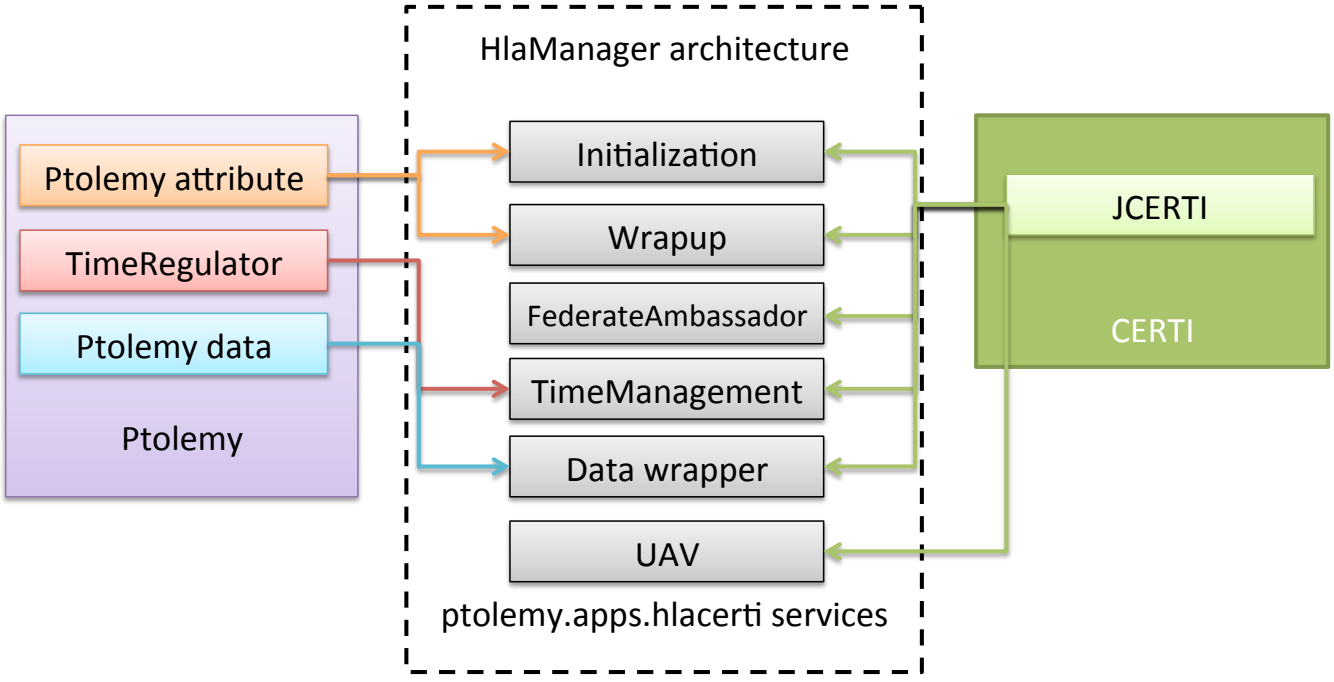


Figure 5.1: Architecture of the *HlaManager* attribute

The figure 5.2 describes a subset of those methods based on a class diagram. For more information the reader may refer to the the ptolemy.apps.hlacerti javadoc [1] or to the HlaManager.java class given in Annex C.3. These services are detailed in the following.

[1] http://sisyphus.eecs.berkeley.edu:8079/hudson/job/ptII/javadoc/ptolemy/apps/hlacerti/lib/package-frame.html

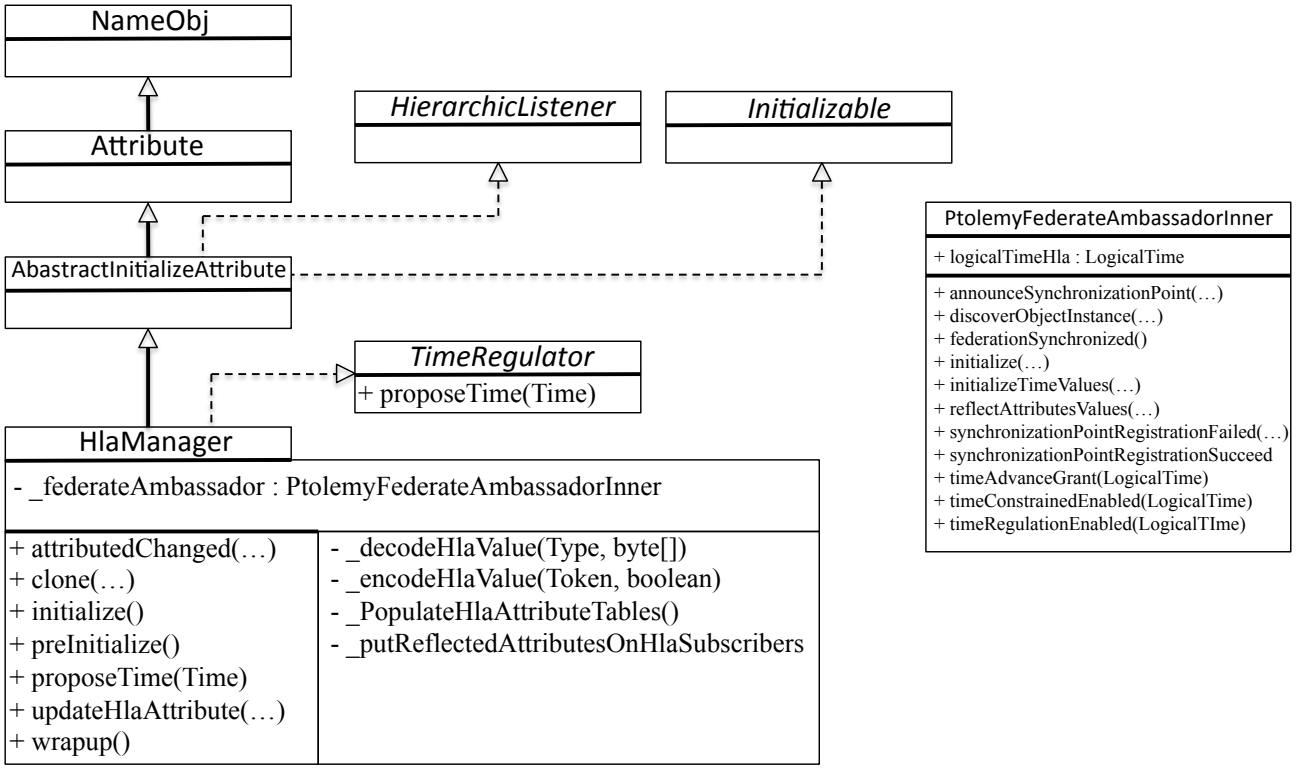


Figure 5.2: Simplified class diagram of the *HlaManager* attribute

5.3.2 Initialization

The Initialization service is composed of two methods `preinitialize()` and `initialize()` that belongs to the Ptolemy attribute core specification (see class diagram figure 5.2). The preinitialize() method implements the calls to the `ptolemy.apps.hlacerti.lib.CertiRtig` API to initiate the launch of the rtig process (related to operation 6). The initialize() method is dedicated to the initialization of the Ptolemy federate and performes different tasks.

First, it calls the _populateHlaAttributeTables() method which configures and populates the routing tables for the HlaPublisher and HlaSubscriber actors. These tables are used to allow a HlaManager attribute to interact with the different actors deployed in the Ptolemy model.

The second task is a set of calls to the JCERTI bindings - i.e. HLA services - for the creating and the initialization of the rtia. This instantiated object provided by the FederateAmbassador service acts as proxy for the communication between the federate and the RTI and implements the different callbacks required by a Ptolemy federate.

The third task is the initialization of the different timing values as start time, time step and lookahead for the Ptolemy federate, specified by the user.

The fourth task is the configuration of the HLA time advancement mecanism used by the federate (e.g. constrained and regulator) and specified by the user during the design of the Ptolemy federate model.

Finally, the last task is the synchronization of the start of the execution of the Ptolemy federate if required by the user. The HLA synchronization point services are used here to suspend the execution of the federate until the end of the initalization phase of the other federates. Without this mecanism, it is possible to have some federates that finish their execution before a federate joins the federation.

5.3.3 Wrapup

The wrapup service implements only the wrapup() method of the Ptolemy attribute core specification (see class diagram figure 5.2). This method is responsible to the unpublish and unsubscribe tasks of HLA attribute values, to disconnect the federate to the federation and to destroy the federation if needed.

5.3.4 Data wrapper

A complex issue of co-simulation involving simulation frameworks implemented in distinct execution platforms, is data types representation. Ptolemy handles a set of basic types (e.g byte, double, float, etc) as well as more complex types (record, etc). Most of them are based on Java primitive data types. Fortunately, JCERTI provides the "encodingHelpers" API which interfaces Java primitive data types with several data types in C++ handled by CERTI. In addition, our approach targets distributed realtime and embedded systems simulation where memory are limited resources and data needs to be as simple as possible. The provided API is enough to match data types between Ptolemy and HLA/CERTI.

The data wrapper service implements two methods _encodeHlaValue() to encode a data to send to the federation, and _decodeHlaValue() to decode a data received from the federation to introduce in Ptolemy. These methods are detailed below.

Encode data to HLA/CERTI The _encodeHlaValue() converts a Ptolemy token as a HLA attribute value according to its type specification. `encodingHelpers` are used to ensure the correct wrapping of a token in a data representation recognizable and handled by HLA/CERTI federates. The Ptolemy type specification mecanism (using the Ptolemy inference type mecanism) is used here with a HlaPublisher actor (that is mapped to a HLA attribute value). Most of the time the user does not to set the type of the input port of a HlaPublisher thanks to the type inference mecanism provided by Ptolemy.

Decode data from HLA/CERTI The _decodeHlaValue() converts a HLA attribute value in a Ptolemy token. To be able to detect the correct data representation of a HLA attribute value in a Ptolemy, a user-specification is required here. The user needs to set the type of the output port of the HlaSubscriber actor mapped to a HLA attribute value. This information allows to to retrieve the correct type of the token to produce.

These methods are declared 'private' as this mecanism to wrap the data to exhange between Ptolemy and HLA/CERTI is opaque for the user.

5.3.5 Time management

Let us consider an event $e = (v, t_s)$ with value v and timestamp t_s. The HlaManager time management service should ensure that events exchanged between federates are delivered at timestamp t_s in a Ptolemy model as well as in a HLA Federation.

The complexity introduced by the co-simulation comes from the different semantics attached to the sending operation of an event. For Ptolemy, an event is processed and sent at logical time $t = t_s$. But the specification of HLA [2] says that the federate shall guarantee that it will not generate an event at any time in the future with a time stamp less than or equal to the local time (last granted time) (or less than the local time plus the federate's lookahead if its lookahead is not zero).

Therefore, the HLA standard provides two additionnal services TARA and NERA to deal with a null lookahead ($l = 0$).

Let us recall that the HlaManager is a time regulator attribute. It implements the proposeTime() method (section 5.2.2) that realizes the time management service between both simulators. The algorithm proposed in section 4 has been implemented in this method.

Under some restrictions, we provide the use of the 4 HLA time advancement mecanisms (TAR, NER, TARA and NERA) to the user. The next sub-sections will describe how they are integrated in the HlaManager.

5.3.5.1 proposeTime()

The proposeTime() method of the HlaManager time regulator attribute is given in listing 5.3. The time management is only effective if the Ptolemy federate is declared as constrained and regulator (see our restrictions in chapter 4).

The different steps described in the time advancement algorithm proposed in chapter 4 are easily distinguishable. According to the configuration of the HlaManager, set by the user, the Ptolemy federate will execute one and only one of the four time advancement mecanisms during the simulation.

Listing 5.3: ptolemy.apps.hlacerti.lib.HlaManager.proposeTime()

```
/** Propose a time to advance to. This method is the one implementing the
 *  TimeRegulator interface and using the HLA/CERTI Time Management services
 *  (if required). Following HLA and CERTI recommendations, if the Time
 *  Management is required then we have the following behavior:
 *  Case 1: If lookahead = 0
 *   -a) if time-stepped Federate, then the timeAdvanceRequestAvailable()
 *       (TARA) service is used;
 *   -b) if event-based Federate, then the nextEventRequestAvailable()
 *       (NERA) service is used
 *  Case 2: If lookahead > 0
 *   -c) if time-stepped Federate, then timeAdvanceRequest() (TAR) is used;
 *   -d) if event-based Federate, then the nextEventRequest() (NER) is used;
 *  Otherwise the proposedTime is returned.
 *  NOTE: For the Ptolemy II - HLA/CERTI cooperation the default (and correct)
 *  behavior is the case 1 and CERTI has to be compiled with the option
 *  "CERTI_USE_NULL_PRIME_MESSAGE_PROTOCOL".
 *  @param proposedTime The proposed time.
 *  @return The proposed time or a smaller time.
 *  @exception IllegalActionException If this attribute is not
 *  contained by an Actor.
 */
@Override
public Time proposeTime(Time proposedTime) throws IllegalActionException {
    Time breakpoint = null;

    // This test is used to avoid exception when the RTIG subprocess is
    // shutdown before the last call of this method.
    // GL: FIXME: see Ptolemy team why this is called again after STOPTIME ?
    if (_rtia == null) {
        if (_debugging) {
            _debug(this.getDisplayName() + " proposeTime() -"
             + " called but _rtia is null");
        }
        return proposedTime;
    }

    // If the proposedTime has already been asked to the HLA/CERTI Federation
    // then return it.

    // GL: FIXME: Comment this until the clarification with NERA and TARA
    // and the use of TICK is made.
    /*
     if (_lastProposedTime != null) {
```

```
 if (_lastProposedTime.compareTo(proposedTime) == 0) {

 // Even if we avoid the multiple calls of the HLA Time management
 // service for optimization, it could be possible to have events
 // from the Federation in the Federate's priority timestamp queue,
 // so we tick() to get these events (if they exist).
 try {
 _rtia.tick();
 } catch (ConcurrentAccessAttempted e) {
 throw new IllegalActionException(this, e, "ConcurrentAccessAttempted ");
 } catch (RTIinternalError e) {
 throw new IllegalActionException(this, e, "RTIinternalError ");
 }

 return _lastProposedTime;
 }
 }
 */

// If the HLA Time Management is required, ask to the HLA/CERTI
// Federation (the RTI) the authorization to advance its time.
if (_isTimeRegulator && _isTimeConstrained) {
    synchronized (this) {
        // Build a representation of the proposedTime in HLA/CERTI.
        CertiLogicalTime certiProposedTime = new CertiLogicalTime(
                              proposedTime.getDoubleValue());

        // Call the corresponding HLA Time Management service.
        try {
            if (_eventBased) {
                if (_hlaLookAHead > 0) {
                    // Event-based + lookahead > 0 => NER.
                    if (_debugging) {...}
                    _rtia.nextEventRequest(certiProposedTime);
                } else {
                    // Event-based + lookahead = 0 => NERA + NER.
                    // Start the time advancement loop with one NERA call.
                    if (_debugging) {...}

                    _rtia.nextEventRequestAvailable(certiProposedTime);

                    // Wait the grant from the HLA/CERTI Federation (from the RTI)
                        .
                    _federateAmbassador.timeAdvanceGrant = false;
                    while (!(_federateAmbassador.timeAdvanceGrant)) {
                        if (_debugging) {...}

                        try {
                            _rtia.tick2();
                        } catch (Exception e) {
                            throw new ... Exception + reason ...
                        }
                    }

                    // End the loop with one NER call.
                    if (_debugging) {...}

                    _rtia.nextEventRequest(certiProposedTime);

                }
            } else {
                if (_hlaLookAHead > 0) {
                    // Time-stepped + lookahead > 0 => TAR.
                    if (_debugging) {...}
                    _rtia.timeAdvanceRequest(certiProposedTime);
```

```
                } else {
                    // Time-stepped + lookahead = 0 => TARA + TAR.
                    // Start the loop with one TARA call.
                    if (_debugging) {...}

                    _rtia.timeAdvanceRequestAvailable(certiProposedTime);

                    // Wait the grant from the HLA/CERTI Federation (from the RTI)
                        .
                    _federateAmbassador.timeAdvanceGrant = false;
                    while (!(_federateAmbassador.timeAdvanceGrant)) {
                        if (_debugging) {...}

                        try {
                            _rtia.tick2();
                        } catch (Exception e) {
                            throw new  ... Exception + reason ...
                        }
                    }

                    // End the loop with one TAR call.
                    if (_debugging) {...}

                    _rtia.timeAdvanceRequest(certiProposedTime);

                }
            }
  } catch (Exception e) {
    throw new  ... Exception + reason ...
            if (_debugging) {...}
            return proposedTime;
        }

        // Wait the grant from the HLA/CERTI Federation (from the RTI).
        _federateAmbassador.timeAdvanceGrant = false;
        while (!(_federateAmbassador.timeAdvanceGrant)) {
            if (_debugging) {...}

            try {
                _rtia.tick2();
            } catch (Exception e) {
                throw new ... Exception + reason ...
            }
        }

        // At this step we are sure that the HLA logical time of the
        // Federate has been updated (by the reception of the TAG callback
        // (timeAdvanceGrant()) and its value is the proposedTime or
        // less, so we have a breakpoint time.
        try {
            breakpoint = new Time(
              _director,
              ((CertiLogicalTime) _federateAmbassador.logicalTimeHLA)
              .getTime());
  } catch (Exception e) {
    throw new ... Exception + reason ...
        }

        // Stored reflected attributes as events on HLASubscriber actors.
        _putReflectedAttributesOnHlaSubscribers();
    }
}

//_lastProposedTime = breakpoint;
return breakpoint;
```

```
        }
```

After the execution of the time advancement request to the proposeTime value, the HlaManager waits the response (TAG condition) of the federation by "ticking" the rtig. The "ticking" operation is an active loop which is broken at the the TAG reception. During this phase, each event sent from another federate to the Ptolemy federate, is delivered through the execution of a reflectedAttributeValues() (RAV) callback implemented by the FederateAmbassador service.

At the reception of a RAV callback, each HLA event (e.g. an updated attribute value) is converted to a Ptolemy event (DE Event or token). The DE Event is automatically merged to the DE Queue (opaque operation from the Ptolemy framework) and the token associated to this event needs to be stored in its HlaSubscriber destinatory.

When the TAG condition is validated, the ticking loop is broken, each token produced from RAV is stored in its corresponding HLASubscriber actor. This mecanism is implemented in the _putReflectedAttributesOnHlaSUbscribers() method. Finally the timestamp associated to the TAG is returned to the director as the time to advance to.

5.3.5.2 Event-based federates

Ptolemy DE model owns a logic of event-based simulation execution. Timestamped events are produced and delivered at a specific point in time, modeling the progression of the simulation. Most of the time, it is quite natural for the user to choose the use of HLA event-based time advancement mecanisms (NER and NERA) to design a Ptolemy federate model that will exchanges data with a federation.

The semantics attached to those services (and given in chapter 2) is well suited for an integration in the Ptolemy framework. Thanks to the time management service described above, events received from the other federates of the federation are injected at timestamp in the Ptolemy federate model (see NER and NERA semantics in chapter 4).

5.3.5.3 Time-stepped federates

The user may select time-stepped time advancement mecanisms (TAR and TARA) for a specific purpose. One reason could be to force a Ptolemy federate model to produce outputs (resp. to receive inputs) to the federation only at periodic time. This could be usefull for predictable computations as required for schedulability analysis.

As we deal with the strong semantics of TAR and TARA defined by the HLA standard, the user have to assume the following restriction:

Restriction 2. *The use of Ptolemy time-stepped federates implies that each RAV received during the time advancement phase will have its timestamp set to the proposeTime value used for the time advance request.*

By this restriction, the user assumes that a Ptolemy federate may receive HLA events with timestamp less than the proposeTime value that will be injected in the Ptolemy simulation with a different timestamp (= proposeTime value). This restriction avoids to introduce HLA events in a Ptolemy simulation with a timestamp less than the currentTime of the Director.

Therefore, modeling Ptolemy time-stepped federate behavior is less flexible than event-based federates and required specific knowledges of the HLA standard.

5.3.6 FederateAmbassador

The FederateAmbassador service contains the implementation of the callbacks required by a Ptolemy federate. These callbacks defines a subset of HLA callback services proposed by the HLA standard. The table 5.3 resumes the callbacks supported by a Ptolemy federate.

HLA Areas	HLA callback services	Description (non-formal)
Federation	synchronizationPointRegisterSucceeded() announceSynchronizationPoint() federationSynchronized()	register synchro point succeeded wait a synchronization point announce synchronization
Object	discoverObjectInstance() reflectAttributeValues() RAV	for object instances discovering receive updated value
Time	timeRegulationEnabled() timeConstrainedEnabled() timeAdvanceGrant() TAG	federate as regulator succeeded federate as contrained succeeded notify time advancement granted

Figure 5.3: HLA/CERTI callbacks implemented in the FederateAmbassador serivce

Particularly, the reflectAttributeValues() (RAV) callback is the one responsible to the conversion of a HLA event to Ptolemy event (DE Event + token). It uses the data wrapper service and the Ptolemy framework. The code of this method is given in the listing C.3, Appendix C.

5.3.7 UAV and updateHlaAttribute()

The UAV service is responsible for sending a Ptolemy event to the HLA federation. This service is not directly executed by the HlaManager, but it is provided to the HlaPublisher actor connected to the output port of the actor in the Ptolemy model that want to send an updated (attribute) value to the federation.

The listing 5.4 describes the updateHlaAttribute() method that implements the UAV service. We use the _hlaAttributesToPublish routing table of HlaPublisher actors to make the relation between the updated attribute value to send and its corresponding HLA attribute (FOM specification). According to the HLA standard these information are required to build the different parameters used by the updateAttributeValues() (UAV) HLA service provided by JCERTI.

According to the HLA standard, an updated attribute value must be sent with a timestamp greater than the current HLA logical time (i.e. to the last TAG timestamp received by the federate) if the TAR or NER time advancement mecanisms is used with a lookahead value > 0; or greater or equal to the current HLA logical time in the case of TARA and NERA mecanisms with a lookahead = 0.

During the implementation of the co-simulation framework we have not restricted this possibility to offer more flexibility during the development of a Ptolemy federate. Unfortunately, the case of TAR and NER with lookahead > 0 requires an adaptation due to the Ptolemy DE semantics where an event $e = (v, t_s)$ is processed and sent at timestamp t_s. Let us recall that HLA standard states that if the logical time of a federate is t then it only can send an UAV (update attribute value) with time $t+$ *lookahead*. To avoid the federate (at logical time t) to send an event to the federation with timestamp t and to get an exception from CERTI we choose to add an arbitrary ε value to the timestamp t, sending the value v at $t+\varepsilon$, with ε=*lookahead*. The problem with this choice is that the event is sent with a delay: instead of sending an event $e = (v, t_s)$, the federate will send $e' = (v, t_s+lookahead)$ (see listing 5.4).

Time representation consistency In the framework, the time representation consistency is simple. JCERTI provides the CertiLogicalTime class to build a time representation value for HLA/CERTI. The CertiLogicalTime class is a rich encapsulation of a Java double. The correct conversion of CertiLogicalTime java object to a CertiLogicalTime c++ object is ensured by JCERTI.

In Ptolemy, model time representation belongs to the ptolemy.actor.util.Time object class. This class is a rich encapsulation of a java.math.BigInteger which is very accurate to handle big time representation values. The Time class provides converters such as the ptolemy.actor.util.Time.getDoubleValue() with the limitation that the convertion may not preserve the time resolution because of the limited digits for double representation.

As the CertiLogicalTime supports only double we use this converter to convert a timestamp of a Ptolemy event to a timestamp of HLA/CERTI. The Time class also provides constructor based on a double value. We use this constructor to build a Time object in a Ptolemy simulation model from the timestamp of a HLA event.

Listing 5.4: ptolemy.apps.hlacerti.lib.HlaManager.updateHlaAttribute()

```
/** Update the HLA attribute <i>attributeName</i> with the containment of
 *  the token <i>in</i>. The updated attribute is sent to the HLA/CERTI
 *  Federation.
 *  @param hp The HLA publisher actor (HLA attribute) to update.
 *  @param in The updated value of the HLA attribute to update.
 *  @throws IllegalActionException If a CERTI exception is raised then
 *  displayed it to the user.
 */
void updateHlaAttribute(HlaPublisher hp, Token in)
        throws IllegalActionException {
    Time currentTime = _director.getModelTime();

    // The following operations build the different arguments required
    // to use the updateAttributeValues() (UAV) service provided by HLA/CERTI.

    // Retrieve information of the HLA attribute to publish.
    Object[] tObj = _hlaAttributesToPublish.get(hp.getName());

    // Encode the value to be sent to the CERTI.
    byte[] valAttribute = _encodeHlaValue(hp, in);

    SuppliedAttributes suppAttributes = null;
    try {
        suppAttributes = RtiFactoryFactory.getRtiFactory()
                .createSuppliedAttributes();
    } catch (RTIinternalError e) {
        throw new IllegalActionException(this, e, "RTIinternalError ");
    }
    suppAttributes.add((Integer) tObj[4], valAttribute);

    byte[] tag = EncodingHelpers.encodeString(((TypedIOPort) tObj[0])
            .getContainer().getName());

    // Create a representation of the current director time for CERTI.
    // HLA implies to send event in the future when using NER or TAR services with
        lookahead > 0.
    // To avoid CERTI exception when calling UAV service, in the case of NER or TAR
    // with lookahead > 0, we add the lookahead value to the event's timestamp.
    CertiLogicalTime ct = new CertiLogicalTime(currentTime.getDoubleValue()
            + _hlaLookAHead);
    try {
        _rtia.updateAttributeValues((Integer) tObj[5], suppAttributes, tag,
                ct);
    } catch (Exception e) {
        throw new IllegalActionException(this, e, e.getMessage());
    }
    if (_debugging) {...}
}
```

5.4 `HlaPublisher` and `HlaSubscriber` actors

Mecanisms and operations required to exchange data between a Ptolemy simulation models and a HLA/CERTI Federation are realized across the development of two Ptolemy actors so called `HlaPublisher` and `HlaSubscriber`. Both actors belongs to the `ptolemy.apps.hlacerti` library and are combined with the `HlaManager` to enable the co-simulation PtolemyII - HLA/CERTI.

According to their semantics, actors express all the behavior characteristics that are required to implement the followings operations:

- send!: send a Ptolemy event to a HLA/CERTI Federation according to its timestamp and with respect to the Ptolemy event processing semantics.
- receive?: receive HLA/CERTI events and process them at timestamp in a ptolemy simulation model.

The next sub-sections shows the implementation of the actor `HlaPublisher` (resp. `HlaSubscriber`) which realizes the *send!* (resp. *receive?*) operation.

5.4.1 `HlaPublisher`: send Ptolemy events to the HLA/CERTI Federation

According to the DE event semantics, a DE event is consumed when the token attached to this event is taken from the receiver queue of its destination actor's input port. Token are then processed at timestamp.

Our approach defines a specific actor so called `HlaPublisher` that implements a publisher in a HLA/CERTI Federation. Its algorithm is quite simple: At each reception of a token on the actor's input port, the actor is fired and the operation *send!* has to be performed. (see listing C.2 in Appendix C).

The listing 5.5 shows only the `fire()` method of the `HlaPublisher` triggered after the reception of an event on its input port.

Listing 5.5: ptolemy.apps.hlacerti.lib.HlaPublisher.java

```
/** Each tokens, received in the input port, are transmitted to the
 *  {@link HlaManager} for a publication to the HLA/CERTI Federation.
 */
public void fire() throws IllegalActionException {
    if (inputPortList().get(0).hasToken(0)) {
        Token in = inputPortList().get(0).get(0);
        _hlaManager.updateHlaAttribute(this.getName(), in,
                _asHLAPtidesEvent);

        if (_debugging) {
            _debug(this.getDisplayName()
                    + " Called fire() - the update value \""
                    + in.toString() + "\" of the HLA Attribute \""
                    + this.getName() + "\" has been sent to \""
                    + _hlaManager.getDisplayName() + "\"");
        }
    }
}
```

To simplify the design of these new actors, we volontary choose to implement the data communication logic in the `HlaManager` attribute. This attribute provides the `updateHlaAttribute()` method to a `HlaPublisher` actor to realize the *sent!* operation.

Mapping `HlaPublisher` to HLA attribute A `HlaPublisher` actor is mapped to one HLA attribute. A Ptolemy token, received in the input port of this actor, is interpreted as an updated value of the HLA attribute. The updated value is published to the whole HLA Federation using a mecanism provided by the `HlaManager` attribute deployed in a Ptolemy model.

The name of this `HlaPublisher` actor is mapped to the name of the HLA attribute in the federation. This name needs to match the Federate Object Model (FOM) specified for the Federation (see Table 5.1). The data type specified for the actor's input port actor has to be the same type of the one specified for the HLA attribute. Finally, the parameter `classObjectHandle` links the publisher to the attribute object class describes in the FOM.

FOM	HlaPublisher actor
object class	parameter "Name of object class in FOM"
attribute	name of the actor

Table 5.1: Mapping `HlaPublisher` to the FOM.

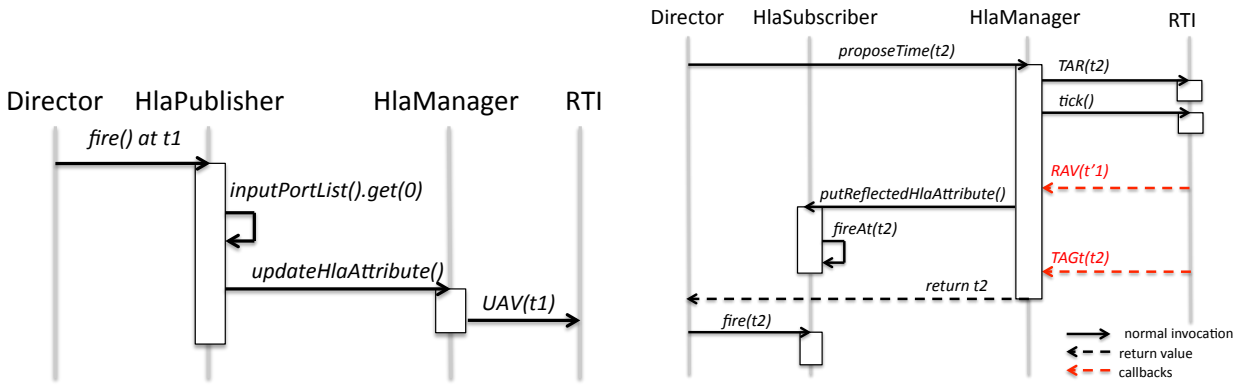
5.4.2 HlaSubscriber: to introduce HLA events in a Ptolemy model

The HlaSubscriber actor implements the mecanisms required to *inject* a HLA event coming from a federate converting it in a Ptolemy event in the simulation model. The actor's architecture is composed by:

- a set of parameters that allow to specify all the information required by HLA to subscribe to a HLA attribute value. The mapping HlaSubscriber actor to HLA attribute is the same as the one describes for the HlaPublisher actor (see section 5.4.1).
- a set of internal variables. The HlaSuscriber actor specifies a _reflectedAttributeValues data structure that implements an event queue. This queue is used to store HLA events converted as Ptolemy events.
- a set of methods. The public fire() method is triggered by the director at a specific model time declared by the actor - i.e. programmed through the call of the fireAt() method executed by the actor. The firing of a HlaSubscriber actor means that an event needs to be processed in its internal queue. The processing of the event leads to the production of a token which is send through the output port of the actor.
  - The private _buildToken() method is called by the fire() method and is responsible of the construction of a typed token which matches the type of the corresponding HLA attribute specified by the user.
  - The public _putReflectedHlaAttribute() method, described in listing 5.6, is provided by the actor to the HlaManager. This method is invoked during the time management phase to store all HLA events received from the federation, converted as a Ptolemy event and which has the HlaSubscriber actor as destinatory. This method stores the Ptolemy event in the internal event queue and program the re-firing of the actor (using fireAt()) at the event timestamp.

Listing 5.6: Code snippets of ptolemy.apps.hlacerti.HlaSubscriber.java

```
/** Store each updated value of the HLA attribute (mapped to this actor) in
 *  the tokens queue. Then, program the next firing time of this actor to
 *  send the token at its expected time. This method is called by the
 *  {@link HlaManager} attribute.
 *  @param event The event containing the updated value of the HLA attribute
 *  and its time-stamp.
 *  @exception IllegalActionException Not thrown here.
 */
public void putReflectedHlaAttribute(TimedEvent event)
        throws IllegalActionException {
    // Add the update value to the queue.
    _reflectedAttributeValues.add(event);
```



a) Updating of a HLA Attribute b) Handling reflected HLA Attribute

Figure 5.4: Execution flow

```
        // Program the next firing time for the update value received.
        _fireAt(event.timeStamp);
    }
```

5.5 Interaction between TimeRegulator, HlaManager and HLA actors

This section presents the interaction between the TimeRegulator interface (section 5.2), HlaManager attribute (section 5.3) and the HlaSubscriber and HlaPublisher actors (section 5.4).

5.5.1 Interaction with `HlaPublisher`

Let us consider now the methods `updateHlaAttribute`() and `proposeTime`() of the `HlaManager` attribute presented in Figure 5.1 and 5.2. Figure 5.4.a describes the interaction (through its execution flow), to publish an update of a HLA attribute, between the `director`, the `HlaPublisher` actor, the `HlaManager` attribute and the RTI. The methods proposed by the RTI are provided by the JCERTI API.

The `updateHlaAttribute`() method is provided by the `HlaManager` to `HlaPublisher` actors. This method encapsulates the operations to build an updated value of a HLA attribute from a Ptolemy event, and calls the Object management services relative to the publication of the updated value (only the call to the `updateAttributeValues`() service is showed in the Figure 5.4.a.

The initialize() method (not showed in Figure 5.4.a of the `HlaPublisher` allows to retrieve the reference to the `HlaManager` deployed in the model. Thus the actor can invoke the `updateHlaAttributeValue`() method provided by the manager during the execution of its `fire`() method (triggered by the director). A publication action is enabled by the connection of an output port of a Ptolemy source actor of the core library to the input port of the `HlaPublisher`. To summarize, each reception of an event $e = (v,t)$ on this input port (at time t) leads to the call of the updateAttributeValues() (UAV) service, by the `HlaManager`, with value v and time t. The JCERTI API is used to build a correct representation of the timestamp t.

5.5.2 Interaction with `HlaSubscriber`

Figure 5.4.b describes the execution flow to introduce a new update of a HLA attribute, received from the HLA/CERTI Federation. This interaction involves the `director`, the `HlaSubscriber` actor, the `HlaManager` attribute and the RTI.

The `HlaSubscriber` contains a queue and provides the putReflectedAttribute() method to the `HlaManager` to store every updated value in it. When a Federate uses the HLA Time management, the reception of these values (e.g. reception of the reflectedAttributeValues() (RAV) callbacks from the RTI) is only possible during a time advancement phase, through the call to the HLA tick() (see Figure 5.4.b). At the end of this phase each RAV received by the `HlaManager` is stored as a (Ptolemy) event in a `HlaSubscriber`. The manager is able to retrieve the corresponding `HlaSubscriber` and to invoke its putReflectedHlaAttribute(). The value of the event is the updated value and its timestamp is the HLA logical time specified by the RAV. The treatment of the RAV is handled in the `proposeTime`() method.

During the execution of the putReflectedHlaAttribute(), a call to the fireAt() method is performed to program the next firing time of the `HlaSubscriber` (i.e. the time when the director have to call its fire() method). This ensures the delivery of the timed-event at the correct time in the Ptolemy simulation. A subscription operation is enabled by connecting the `HlaSubscriber`'s output port to an input port of a Ptolemy sink actor of the core library.

Chapter 6

Conclusion and perspectives

In this work we propose a co-simulation framework for complex, heterogeneous systems such as encountered in Cyber Physical systems. Topics addressed in this work are different notions of time across distributed components, time coordination, communication latencies, co-simulation of heterogeneous components, and interoperability of different simulation environments. This approach discusses the distributed simulation capabilities of HLA/CERTI, and describes how the simulation tool Ptolemy was extended in order to interact with HLA. The extensions in Ptolemy, although specific to the tool, can be generalized for arbitrary simulation tools.

The framework presented here is a joint work between CHESS-UCB and DISC-ISAE and allows the design of complex simulations containing several different simulation models as well as simulation tools, functional source code executed on specific embedded platforms and hardware.

Up to now, the caractheristics of a Ptolemy-HLA federate are the following:

- the top model of a Ptolemy-HLA federate must have a DE director. Models with a Continuous director can also be simulated provided they are encapsulated in a composite actor.
- a Ptolemy-HLA federate is time-constrained and regulated
- for the time managment only NER service was fully tested
- only one synchronization point is allowed.
- only one instance of a same class can be used in a Ptolemy-HLA federate.

Several demos are available at the Ptolemy tree. Among then, there is the case study of a distributed simulation of an hierarchical Ptolemy model based on the F-14 Longitudinal Flight Control demo model from Matlab. This federation has 3 federates: the physical model (Continuous director), the AutoPilot (DE director) and the Pilot Stick. The Pilot Stick federate was successfully replaced by a real joystick controlling the elevator.

This framework is an on-going work and some points will be improved, mainly:

- have a fully tested environnment for TAR, TARA and NERA services,
- allow for multiple instances of a class,
- formalize the algorithms for the time management in Ptolemy-HLA,
- improve the test suite allowing the verification of good behavior of this co-simulation framework.

Bibliography

[1] M.U. Awais, P. Palensky, A. Elsheikh, E. Widl, and S. Matthias. The HLA RTI as a master to the functional mock-up interface components. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*, pages 315–320, 2013.

[2] IEEE-SA Standards Board. Ieee standard for modeling and simulation high level architecture (hla) – federate interface specification. Technical report, IEEE, Sept 2000.

[3] P. Derler, E. A. Lee, and A. Sangiovanni-Vincentelli. Modeling cyber-physical systems. *Proceedings of the IEEE (special issue on CPS)*, 100:13 – 28, Jan 2012.

[4] J. Eker, J. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Sachs, Y. Xiong, and S. Neuendorffer. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91 - 1:127–144, 2003.

[5] ForwardSim Inc. Simulation and Technologies. HLA Toolbox for MATLAB - HLA Blockset for Simulink, 2013.

[6] R. M. Fujimoto. HLA time management: Design document. Technical report, Georgia Tech College of Computing, Aug 1996.

[7] Lasnier Gilles. Toward a Distributed and Deterministic Framework to Design Cyber-PhysicalSystems. Technical report, Institut Superieur de l'Aeronautique et de l'Espace - ISAE, Oct 2013.

[8] Hassen Hadj-Amor and Thierry Soriano. A contribution for virtual prototyping of mechatronic systems based on real-time distributed high level architecture. *Journal of computing and information science in engineering*, 12(1), 2012.

[9] IEEE. IEEE Standard for Modeling and Simulation High Level Architecture (HLA) – Federate Interface Specification. *IEEE Std 1516.1TM-2010*, pages 1–378, 2010.

[10] IEEE. IEEE Standard for Modeling and Simulation High Level Architecture (HLA) – Framework and Rules. *IEEE Std 1516TM-2010*, pages 1–38, 2010.

[11] IEEE. IEEE Standard for Modeling and Simulation High Level Architecture (HLA) – Object Model Template (omt) Specification. *IEEE Std 1516.2TM-2010*, pages 1–110, 2010.

[12] G. Lasnier, J. Cardoso, P. Siron, and Pagetti. Environnement de cooperation de simulation pour la conception de systemes cyber-physiques. *Journal europeen des systemes automatises*, 47, 2013.

[13] G. Lasnier, J. Cardoso, P. Siron, C. Pagetti, and P. Derler. Distributed simulation of heterogeneous and real-time systems. In *Distributed Simulation and Real Time Applications (DS-RT), 2013 IEEE/ACM 17th International Symposium on*, page 20, 2013.

[14] Edward A. Lee. Modeling concurrent real-time processes using discrete events. *Ann. Software Eng.*, 7:25–45, 1999.

[15] Edward A. Lee. Heterogeneous actor modeling. In *Proceedings of the 11th International Conference on Embedded Software (EMSOFT 2011)*, pages 3–12, 2011.

[16] MathWorks. MATLAB/Simulink, 2013.

[17] Modelica Association. Modelica: A unified object-oriented language for physical systems modeling, language specification version 3.3, 2012.

[18] Modelisar. Functional mock-up interface for model exchange and co-simulation, Version 2.0 beta 4. Technical report, Information Tech for European Advancement, Aug 2012.

[19] Eric Noulard, Jean-Yves Rousselot, and Pierre Siron. CERTI, an open source RTI, why and how ? *Spring Simulation Interoperability Workshop*, 2009.

[20] ONERA. The Open Source middleware CERTI, 2013.

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

Appendix A

User Manual for HLA-PTII federates

For more information about the HLA-PTII co-simulation framework read [13, 12, 7].

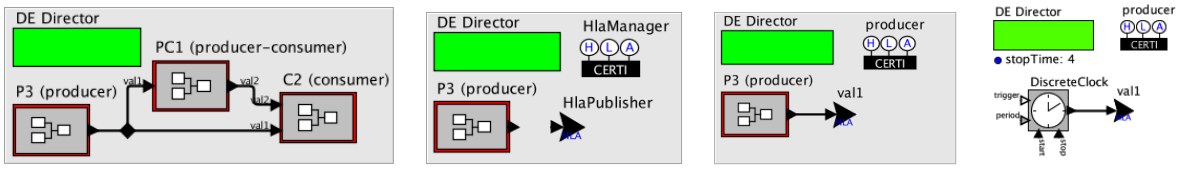
Demos can be found in `$PTII/ptolemy/apps/hlacerti/demo`. The actors can be found in the library under `MoreLibrairies->Co-Simulation->HLA`: HlaManager, HlaPublisher and HlaSubscriber.
The information here are up to revision r71806 (it comes with jcerti.jar and the rtig is launched automatically when running a co-simulation). You need also the HLA compliant RTI called CERTI (tested with versions 3.4.2, 3.4.3 and 3.5) http://www.nongnu.org/certi/certi_doc/Install/html/index.html.

Put in your .bash_profile un export for $PTII and $CERTI_HOME. Or at least put aliases:
`alias myPtII=export PTII=your-path-to-ptII"`
`alias cfgCerti="source your-CERTI_HOME/share/scripts/myCERTI_env.sh."`
Do not forget to execute these aliases in each terminal you open.

1 Building a model with HLA-PTII co-simulation framework

Let us suppose you have already a (centralized) Ptolemy model as the one of figure A.1.a. You want to simulate this model in a distributed way using 3 simulators, one for each composite actor.

Important remark: The top-level model must use a DE director, but you can have a Continuous director in a composite actor inside.



a) Centralized model b) Hla actors c) Naming the actors d) flat federate

Figure A.1: Building Ptolemy federates

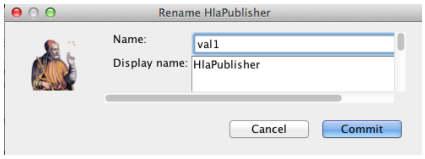
Consider that you want to create the `producer` federate. The steps are the following:

1. Create a FOM file with the classes and attributes used in the simulation as in figure A.2.c
2. Duplicate the model of figure A.1.a, removing all the blocs except the composite actor "P3 (producer)"

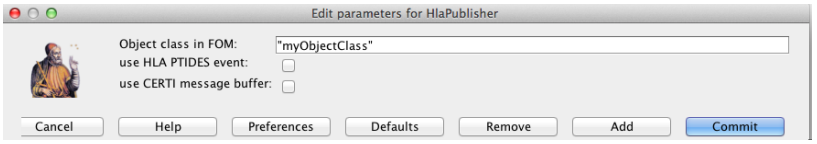
3. Drag the actors HlaManager and HlaPublisher from `MoreLibrairies->Co-Simulation->HLA` (see figure A.1.b)

4. Configuring HlaPublisher. Connect the output of the composite actor to the HlaPublisher actor, then: a) right-click on HlaPublisher, Configure/Rename and put `val1` (the name of the class attribute in the fed file as in figure A.2.a) in the field `name` ; the name displayed will be the same (figure A.1.c); b) double-click HlaPublisher (the window of figure A.2.b pops out) and put `MyObjectClass` (the name of the class according to your fed file in figure A.2.c) in the parameter `Object class in FOM`

5. Configuring HlaManager. Double-click this actor; the window in figure A.2.d appears. Put the name of the federation and the federate as in the fed file (figure A.2.c) and browse the fed file. You can change the lookahead value. If you need a synchronization point, tick the field "Require synchronizationPoint?" and choose a same name for all federates of your distributed simulation. Remark: only the last federate to be launched must have the field "Is synchronization point creator?" ticked.



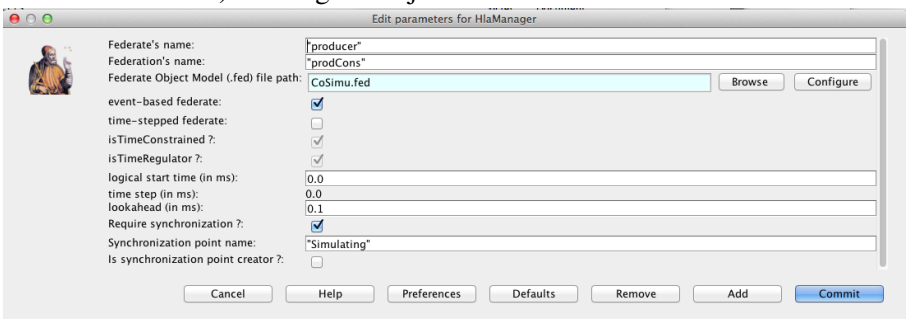
a) Renaming with attribute



b) Naming the object class in HlaPublisher

```
;; CoSimulation
(Fed
  (Federation prodCons)
  (Fedversion v1.3)
  (Federate "prodcons" "Public")
  (Federate "consumer" "Public")
  (Federate "producer" "Public")
  (Spaces)
  (Objects
    (Class ObjectRoot
      (Attribute privilegeToDelete reliable timestamp)
      (Class RTIprivate)
       (Class myObjectClass
              (Attribute val RELIABLE TIMESTAMP)
              (Attribute val1 RELIABLE TIMESTAMP)
              (Attribute val2 RELIABLE TIMESTAMP))))
  (Interactions
    (Class InteractionRoot BEST_EFFORT RECEIVE
      (Class RTIprivate BEST_EFFORT RECEIVE))))
```

c) CoSimu.fed file (FOM)



d) Configuring HlaManager

Figure A.2: FOM

Now your Producer federate is ready and you can create the federates `consumer` and `prodcons`. The steps are similar.

- Do step 2 (duplicate the centralized model removing all composite actors excete the one you want to simulate)

- For `prodcons` federate, add a HlaPublisher for `val2` and a HlaSubscriber for `val1`. Follow setp 4 for HlaPublisher actor. For the HlaSubscriber the procedure is the same. Add also a HlaManager (step 5). Pay attention to name the federate as `prodcons` in the configuration window of HlaManager (figure A.2.d)

- For `consumer` federate, add two HlaSubscriber and a HlaManager. Proceed as in steps 4 and 5 respectively

- Check if all your federates (but one) have the field "Is synchronization point creator?" unticked. Only one of them must have this field ticked and it must be the last one to be launched.

Remark: if your composite actor has a DE director, you can use directly the composite actor as in figure A.1.c or you can have a *flat* model as in figure A.1.d. If the composite actor has a Continuous model then you need to use the composite actor.

2 Running the federation

1. Open a terminal and run `cfgCerti` as said in the beginning of this manual or execute the `$CERTI_HOME/share/scripts/myCERTI_env.sh` shell
2. Go to the folder where the 3 federate models are (or give the absolute address) and open the models: `$PTII/bin/vergil consumer.xml producer.xml prod-cons.xml &`
3. Check there is no `rtig` process running (the first model to be run will automatically launch this process). If there is a `rtig` running, kill the process
4. Check there is only one model that has the field "Is synchronization point creator?" ticked. Run the other models in any order but the last to be run is the one that has the cited field ticked.

Appendix B

Installation guide, user guide and known issues

1 Installation Guide introduction

The current co-simulation framework [1] has been deployed and tested under the following architectures: MacOSX 32 bits, MacOSX 64 bits, Linux Ubuntu (13.04 - *The Raring Ringtail*) 32 and 64 bits. This guide could also be used at entrypoint to install the framework on different Linux based distributions or even under Windows platforms according to the different versions supported by PtolemyII and CERTI.

As our co-simulation is fully integrated in the Ptolemy framework, the execution of a simulation using the PtolemyII - HLA/CERTI co-simulation framework is driven by Ptolemy.

This guide does not intent to duplicate the original documentation provided by Ptolemy and CERTI. Thus, it is highly recommended to consult the Ptolemy[2] and the CERTI [3] websites for their installation process. However, as we use the CERTI environment in a specific configuration you must be sure to set the correct options to the CMAKE build system for CERTI, see subsection 3 for instruction.

In this guide we assume that all sources, binaries, scripts, etc... will be located in the user folder '**$HOME/pthla**'. Make sure to create this folder before running the installation process.

2 Installing Ptolemy and `vergil`

If you want only to use the co-simulation framework for running the demo and create new distributed models, you need just to install the graphical `vergil` editor of Ptolemy and CERTI (section 3).

We recommend to create the '**$HOME/pthla/workspace**' folder and to download (using svn) the `ptII` project in this folder.

You need:

- `SVN` (1.6.17 or later), an open-source version control, used by Ptolemy.
- A Java compiler (Java JDK 6.0 or later), used to compile Ptolemy.

[1] This chapter was partially update on March, 2015. This framework was tested for Ptolemy up to revision r71806 and CERTI 3.5. After revision r71518, Ptolemy comes with `jcerti.jar` so section 4 would be no longer needed.

[2] http://ptolemy.eecs.berkeley.edu/

[3] http://savannah.nongnu.org/projects/certi

```
0. go to $HOME/pthla/workspace
1. svn co https://repo.eecs.berkeley.edu/svn-anon/projects/eal/ptII/trunk ptII
2. export PTII=$HOME/pthla/workspace/ptII
3. cd $PTII
4- ./configure (there is a build.xml in $PTII)
5. ant
6. cd $PTII/bin
7. make
```

You can run now: `$PTII/bin/vergil &`
For running Ptolemy-HLA federates, you need just to install CERTI (see sections 3.1 to 3.4).

3 Installation of the CERTI environment

This section describes the different steps to configure, install and deploy the CERTI environment for the co-simulation PtolemyII - HLA/CERTI.

3.1 Requirements

We assume that you are already familliar with the use of basic `UNIX` commands such as `cp`, `mv`, `pwd` and the edition of path and global variables. The following tools are required to set the environment for the co-simulation framework:

- `CMAKE` (2.8.10 or later), an cross-platform, open-source build system, used by CERTI.
- `CVS` (1.12.13 or later) or `git`, open-source version control, used by CERTI.
- `Lex and Yacc` (GNU Bison 2.3 or later), syntactic analyzer generator, used by CERTI.
- `ANT` (1.8.3 or later), a pure java build tool, used to compile the JCERTI jar.
- `Eclipse`, an open source IDE mostly provided in Java which aims to provide an universal toolset for development. The framework has been tested using `Juno` or `Kepler` version of this tool. Ptolemy is built on the top of Eclipse.
- A C++ compiler (GCC 4.3.2 or later), used to compile CERTI. JCERTI.

3.2 Installation process:

```
0. Open a terminal and create a folder named pthla at your home directory
1.  Go to $HOME/pthla
2. Get the last CERTI cvs source:

    cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/certi co certi

3. Create the folder $HOME/pthla/certi-tools
4. Go to the $HOME/pthla/certi folder (downloaded at step 2.)
5. Create the folder $HOME/pthla/certi/build-certi
6. Go to folder $HOME/pthla/certi/build-certi
7. Run the command (remark:  put the whole command in a unique line)
```

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

8. Run 'make'
9. Run 'make install'

The command executes in step 7. enables the use of the NULL PRIME MESSAGE PROTOCOL algorithm in CERTI for the time management services. After the step 9. CERTI has been compiled and installed in the '**$HOME/pthla/certi-tools**' folder. CERTI provides a script to set the environment (global variables, binaries, etc) to allow the correct launch of RTIG process, RTIA process and federates. To set the CERTI environment properly in a terminal run the command:

```
source $HOME/pthla/certi-tools/share/scripts/myCERTI_env.sh
```

3.3 Deployment and execution

For the deployment and the exeuction of an HLA simulation (in C++), you need one terminal to launch the 'rtig' process, and one terminal per federate if you need to display outputs. To execute the 'rtig' do this procedure:

1. Open a new terminal
2. Source the CERTI environment script:

   ```
   source $HOME/pthla/certi-tools/share/scripts/myCERTI_env.sh
   ```

3. Run 'rtig' for the RTI.

For testing the execution of a federate, you can run the demo `Billard` from CERTI. Make sure you executed `myCERTI_env.sh`. Check if your `$CERTI_HOME` is `$HOME/pthla/certi-tools`.

1. Open a new terminal, source the CERTI environment script myCERTI_env.sh
2. Run 'rtig'
3. In a new terminal, source myCERTI_env.sh
4. Run the billard program:
   billard -n 1 fTest -Ftest.fed
5. You can run other instances of this federate in other windows, just change the name from 1 to another number. After run the last instance, go back to the windows where you run billard 1 for starting the simulation.

You can set the debug mode for the billard demo doing:

```
export BILLARD=DI
```

3.4 Enable the Debug mode

CERTI allows to display traces, i.e. messages exhanged between RTIG, RTIA and federate. These traces are enabled by setting environment variables in the the terminal where the RTIG or the federate is launched:

To enable messages exchanged between RTIA <–> RTIG, run in the RTIG terminal:

```
export RTIG_MSG=D
```

To enable messages exchanged between Federate <–> RTIA, run in a Federate terminal:

```
export RTIA_MSG=I
```

4 Installation of the JCERTI interface

This section describes the procedure to install and deploy the JCERTI interface in Eclipse. We remind that JCERTI is a Java bindings for CERTI.

Remark: After revision r71518, Ptolemy comes with `jcerti.jar`.

4.1 Installation process:

1. Open a terminal and go to '$HOME/pthla'
2. Get the last JCERTI cvs source

```
cvs -z3 -d:pserver:anonymous@cvs.savannah.nongnu.org:/sources/certi co jcerti
```

3. Go to the '$HOME/pthla/jcerti' folder (downloaded at step 2.)
4. Run 'ant'

After the step `4.` the sources of the JCERTI interface have been compiled and packaged in the java jar '**$HOME/pthla/jcerti/build/jar/jcerti.jar**'.

4.2 Deployment in Eclipse:

To import the **jcerti.jar** in a Java project, do:

1. Open Eclipse
2. Right click on your java project
3. Click on 'Build Path' > 'Configure Build Path'
4. In the new frame open, select 'Libraries'
5. Click on 'Add External Archives'
6. Go to the '$HOME/pthla/jcerti/build/jar/' directory
7. Select 'jcerti.jar', then 'Open'
8. Click on 'OK'

After the step `8.`, if you have selected in the Eclipe's tab 'Project' > 'Build Automatically', Eclipse will re-compiled the source of your java project and resolved the dependencies with the JCERTI interface. Otherwise, you have to re-compile your java project.

5 Installation of the PtolemyII project

More information for installing Ptolemy can be found at http://chess.eecs.berkeley.edu/ptexternal/

The PtolemyII svn repository is now located at: https://repo.eecs.berkeley.edu/svn-anon/projects/eal/ptII/trunkptII.

We recommend to create the '**$HOME/pthla/workspace**' folder and to download (using svn) the ptII project in this folder.

5.1 Shortcut to launch a Ptolemy model from Eclipse

The Eclipse tool configured for the Ptolemy project and to run Vergil (ptolemy's graphical editor) needs to have access to the CERTI environment (global variables, binaries, scripts, etc).

At this step, we assume that you have already installed PtolemyII and configure the Vergil application in your Eclipse. The following instructions maybe used to set a shortcut to launch a Ptolemy model (a .xml file) directly from Eclipse.

```
1. Open Eclipse
2. Right click on the 'ptII' java project
3. Select 'Run As' > 'Run Configurations...'
4. In the new frame open, in the right part select 'Java Application'
5. Right click on 'vergil' then select 'Duplicate'
6. Now, click on 'vergil (1)'
7. In the left part, rename 'vergil (1)' as 'vergil with file'
8. Cick on the 'Arguments' tab
9. Under the 'Program arguments' block, click on 'Variables..."
10. Select 'selected_resource_loc' and click on 'Ok'
11. Click on 'Close' and save the new configuration
```

After the step 11. you can now directly launch a Ptolemy model from the Eclipse explorer. To do so, click on the name of your model (a .xml file) then select the configuration **vergil with file**. A Vergil editor will be launched and the selected model will be opened directly. This could be very helpful to save time in development or test phases.

6 Compile the PtolemyII - HLA/CERTI co-simulation framework with Ptolemy

This section presents how to enable the PtolemyII - HLA/CERTI co-simulation framework in the Ptolemy java project (in Eclipse). We assume that you have already succeeded the instructions given in sections 3, 4 and 5 relative to the installation of the CERTI environment, the JCERTI interface and the Ptolemy tool.

Even if our co-simulation framework is integrated in Ptolemy, the framework is not enabled by default because of the strong dependencies to the CERTI environment built on the user machine. This means that the implementation files of the co-simulation framework are not compiled with the Ptolemy project. The implementation files are located in the '**ptII/ptolemy/apps/hlacerti**' folder. By default all files stored in this folder are excluded to the build of Ptolemy.

Figures B.1 to B.10 shows the steps to enable the compilation '**ptII/ptolemy/apps/hlacerti**' sources in Ptolemy using Eclipse.

The first step is to remove the '**ptII/ptolemy/apps/hlacerti**' folder in the exclusion rules of the build path of ptolemy. Go to the 'Build Path' interface of Ptolemy, 'Configure Build Path", in the 'Sources' tab, see figures B.1 and B.2, then select 'Excluded...' and click on 'Edit...". In the 'Exclusion patterns' block, select 'Add Multiple...' (see figure B.3). In the 'Exclusion Pattern Selection' frame select all folders in the '**ptII/ptolemy/apps/** folder except

the '**.svn**' and the '**hlacerti**' folders, click on 'Ok' (see figure B.4). In the same window, in the 'Exclude' frame (see figure B.5), select '**ptII/ptolemy/apps/**' and click on 'Remove'. Then click on 'Finish'. After this step, Eclipse will try to re-compile the 'ptII' project.

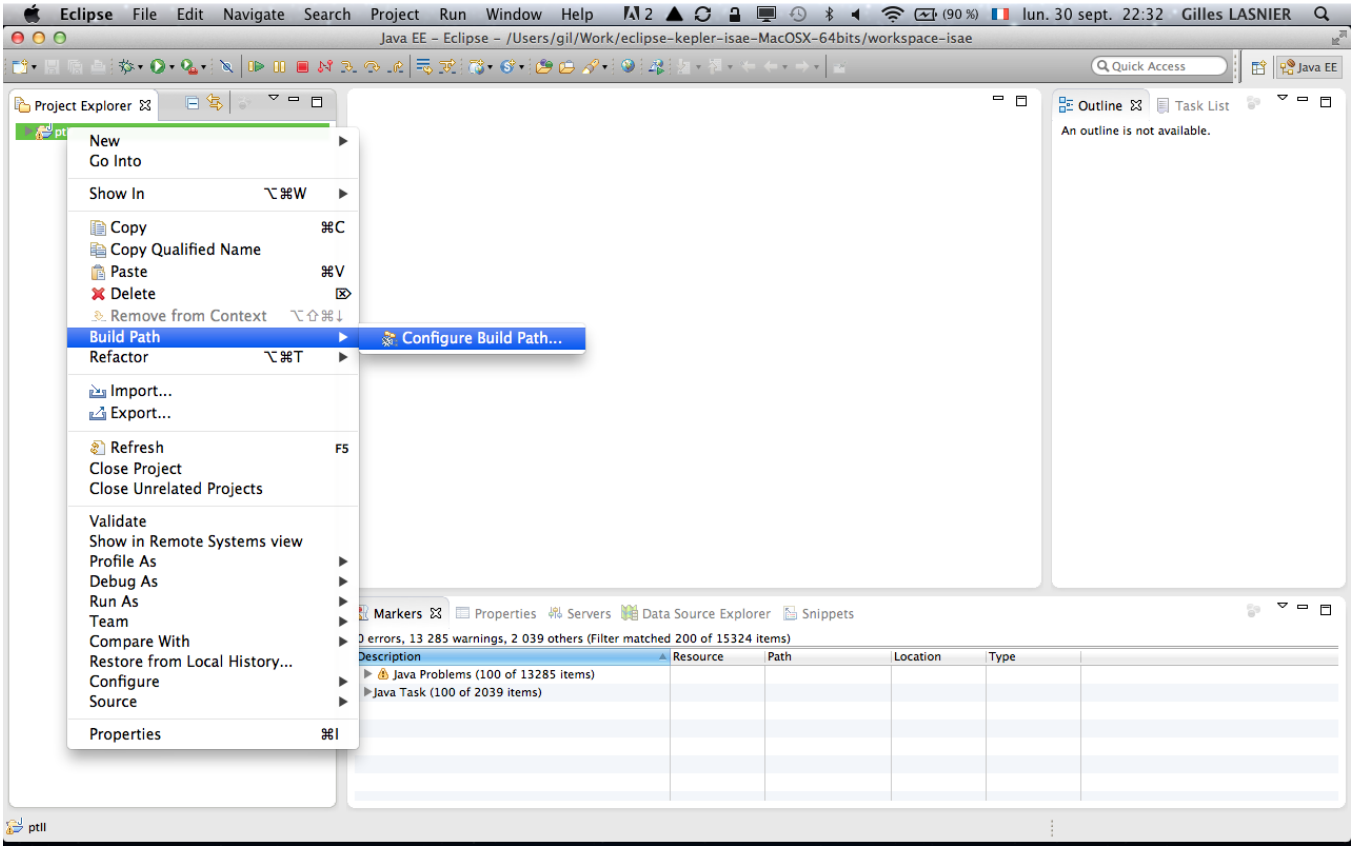


Figure B.1: Build path interface of 'ptII'

At this step, if you have not configure the JCERTI interface for the 'ptII' java project, following the instructions given in section 4, you'll see that the class **HlaManager.java** located in the '**ptII/ptolemy/apps/hlacerti**' java package doest no compile (see figure B.6). These errors are caused by the dependencies to the JCERTI interface which are not resolved yet. Figures B.7 to B.9 shows the steps to do to resolve this problem (the process is described in section 4). The figure B.10 shows the the correct compilation of the **HlaManager.java** class.

Finally, after all those steps you must have no errors in your 'ptII' java project.

7 Running a quick example

This section describes a simple example of PRODUCER - CONSUMER to test your environment. This example is composed on two Ptolemy federates that exchange updated value of an HLA attribue.

8 Known issues

The current implementation of the co-simumalion framework is an on-going work provided as prototype. Even if we have made several deep tests on our framework it may be possible to observe some strange behaviors or bugs, running under different architectures. Also, as we make interoperable different open-source tools it is sometimes

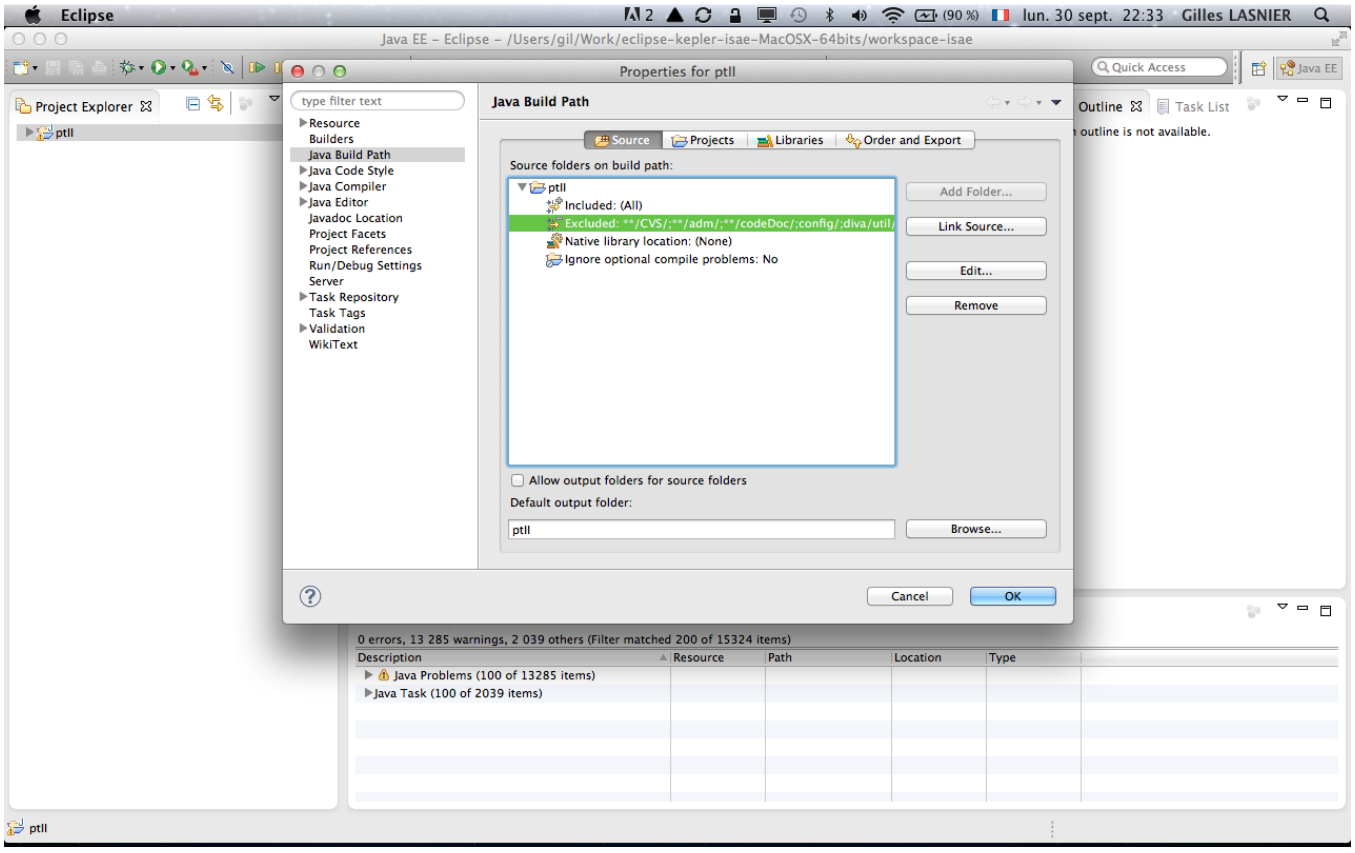


Figure B.2: Build path > Source tab of 'ptII' >

difficult to track all the bugs. Nevertheless, if you find some bugs or bad behaviors, you're welcome to contact our team.

In this section we present a list of bugs or bad utilization of our framework that may cause some problems to run a Ptolemy - HLA/CERTI distributed simulation.

8.1 Specification of Ptolemy federate

This subsection concerns bad behavior that may happen if something is wrong with the specification of a Ptolemy federate.

Typed port of HLA actors We have seen in chapter **??** that an HLA actor is mapped to an HLA attribute described in the FOM. The type of the input or output port of the HLA actor must match the type of its corresponding HLA attribute. Ptolemy provides a feature to the modeler called 'Backward type inference' that simplifies the type specification of ports involved in a Ptolemy model. Such mechanism is based on the analysis of the actors and their ports involved in the model.

This mechanism may cause some troubles in the specification of HLA actors particularly with the `HlaSubscriber` actor. As this actor allows to introduce updated value of an HLA attribute received from the federation, sometimes the 'Backward type inference' mechanisms can not resolved the type of its output port due to a lack of information from the model.

It is recommended to the user to specify manually the type of the output port of the `HlaSubscriber` to avoid such troubles.

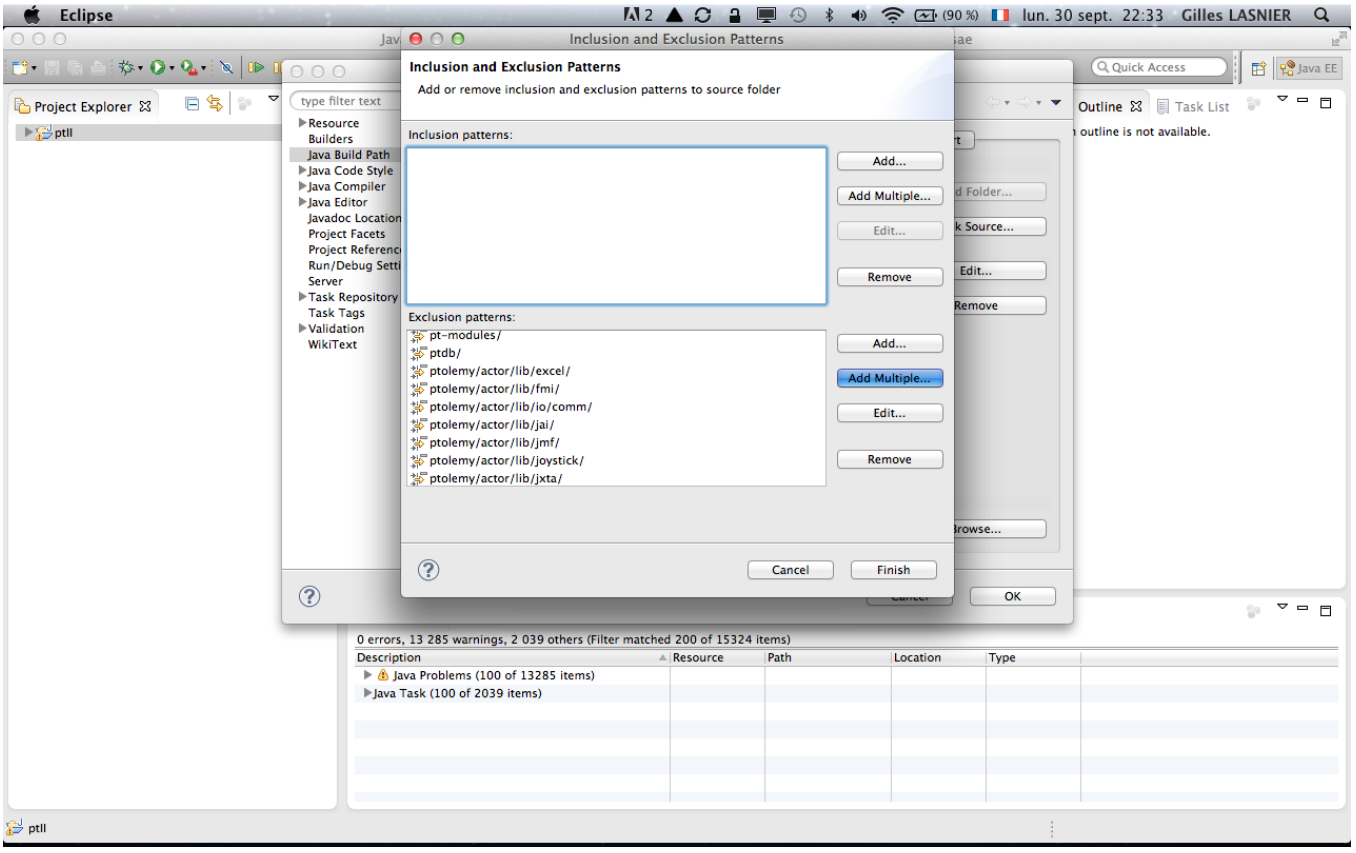


Figure B.3: Inclusion and Exclusions Patterns of 'ptII'

8.2 Running PtolemyII - HLA/CERTI simulation

RTIA socket connection failed We have observed under some recent and powerful computers an issue involving Ptolemy federate and the RTIG process when launching a simulation the first time. It appears that during the first launch, the Ptolemy federate is not able to connect to the RTIG.

Our first investigation supposed that the Ptolemy federate running under Eclipse and using the JCERTI interface are not able to retrieve the RTIA socket. It seems that the initialization and the creation of the RTIA process triggered by JCERTI interface used by the Ptolemy federate is slower than the execution of the federate.

The result of this issue is the '`RTIinternalError: Connection to RTIA failed. libRTI: Network Read Error`' exception caught by Vergil (see figure B.11). Due to the lack of time and resources we have not yet resolved this problem. But a simple solution is to re-launch the simulation.

Reflected HLA attribute value using CERTI Message Buffer When using the 'CERTI Message Buffer' option with HLA actors, if an `HlaSubscriber` actor tries to un-wrapp an reflected value of HLA attribute, received from the federation, that is not a 'CERTI Message Buffer' then the exception '`RTIinternalError: hla.rti.RTIinternalError serial:0`' is raised and caught by Vergil (see figure B.12).

To resolve this problem, the modeler needs to check the option selected using the HLA actor interface.

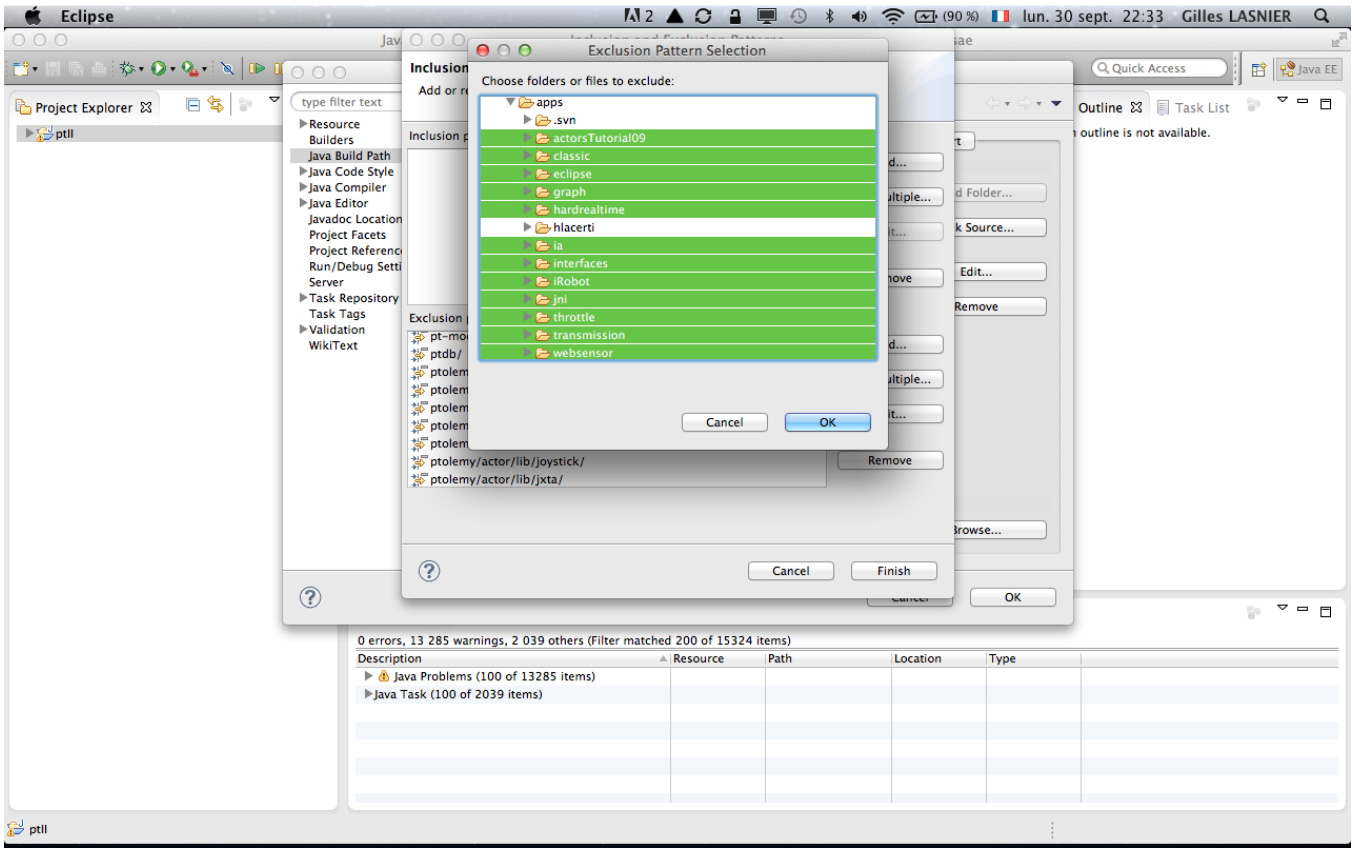


Figure B.4: Exclusion Pattern Sellection frame of 'ptII'

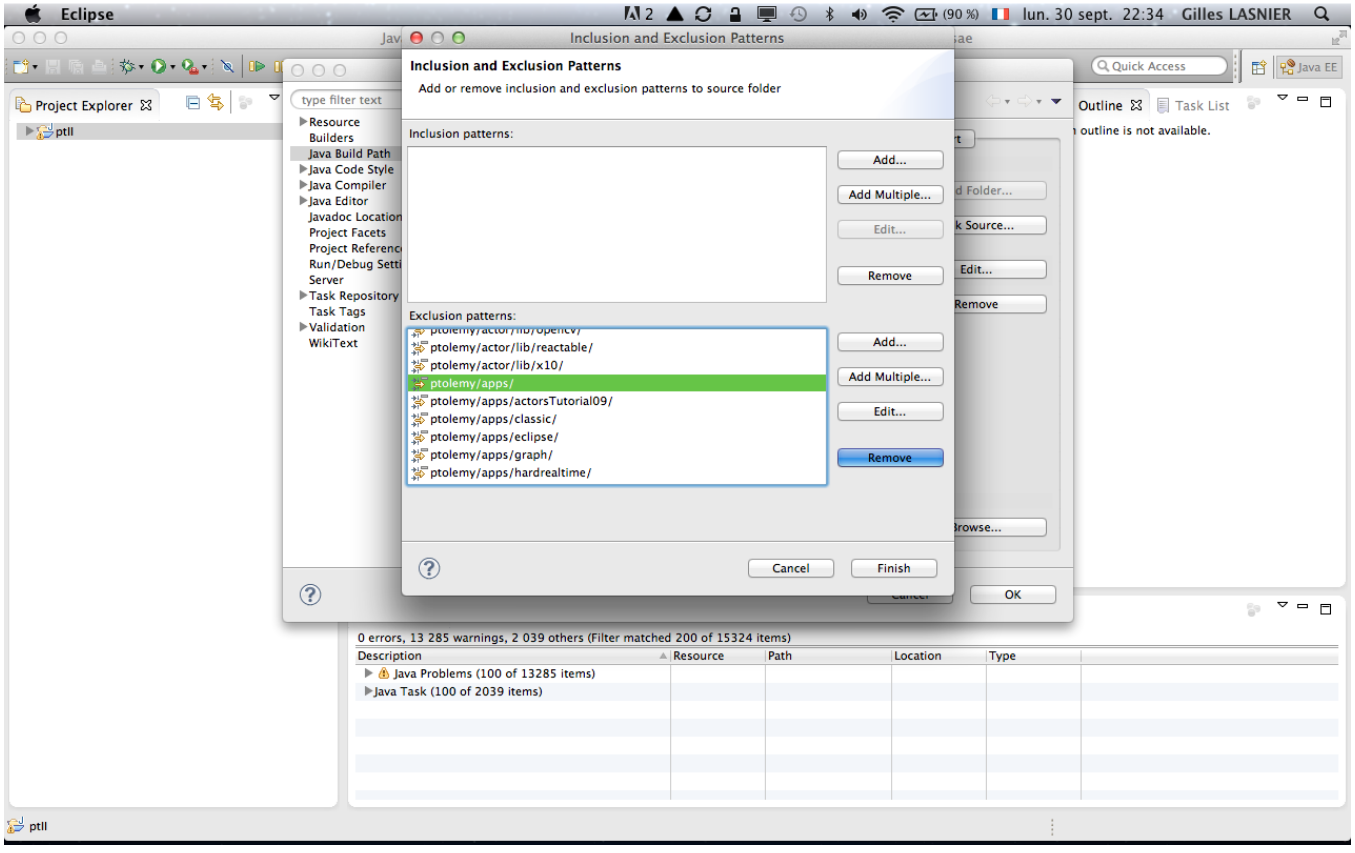


Figure B.5: Inclusion and Exclusion Patterns frame of 'ptII'

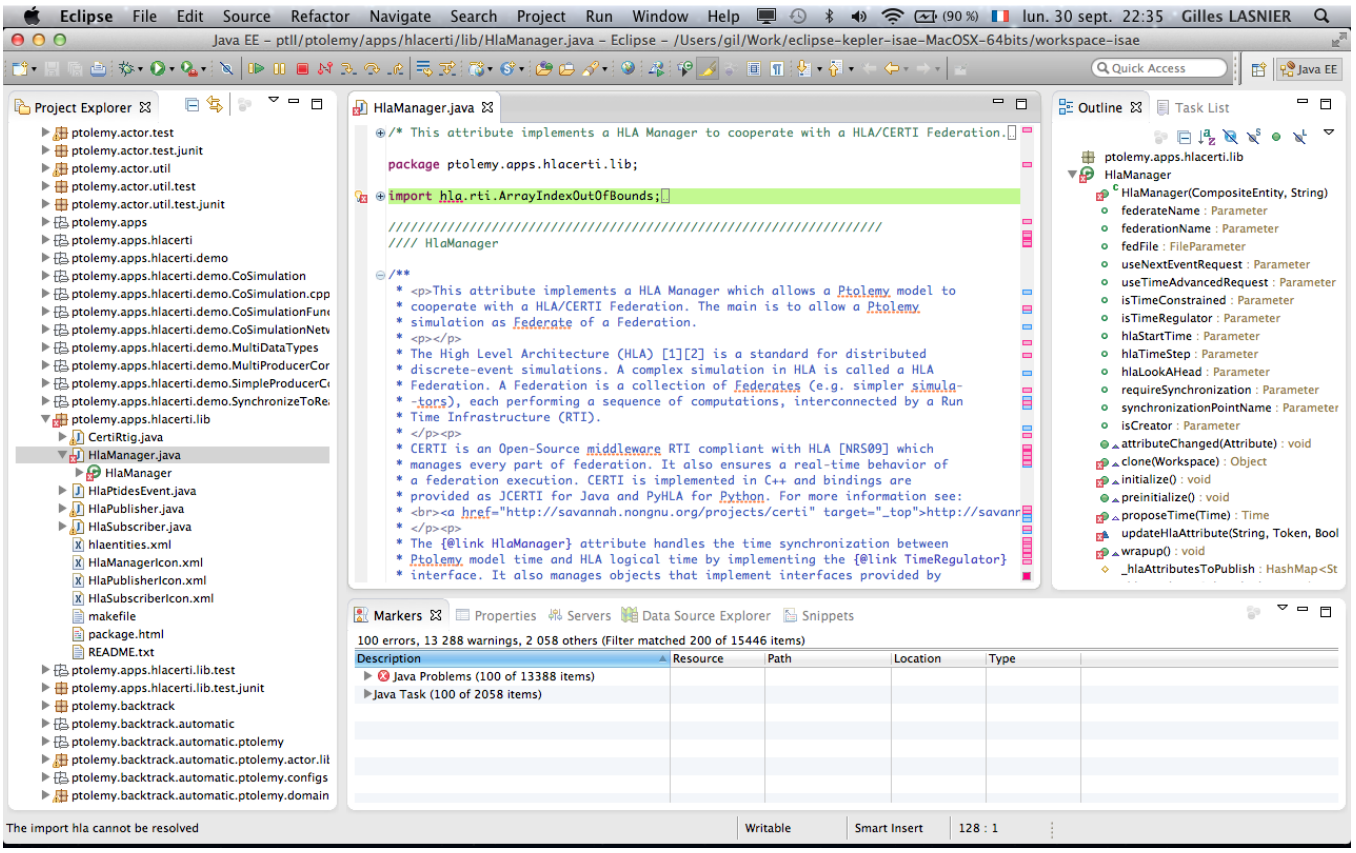


Figure B.6: Compilation errors for the HlaManager.java in 'ptII'

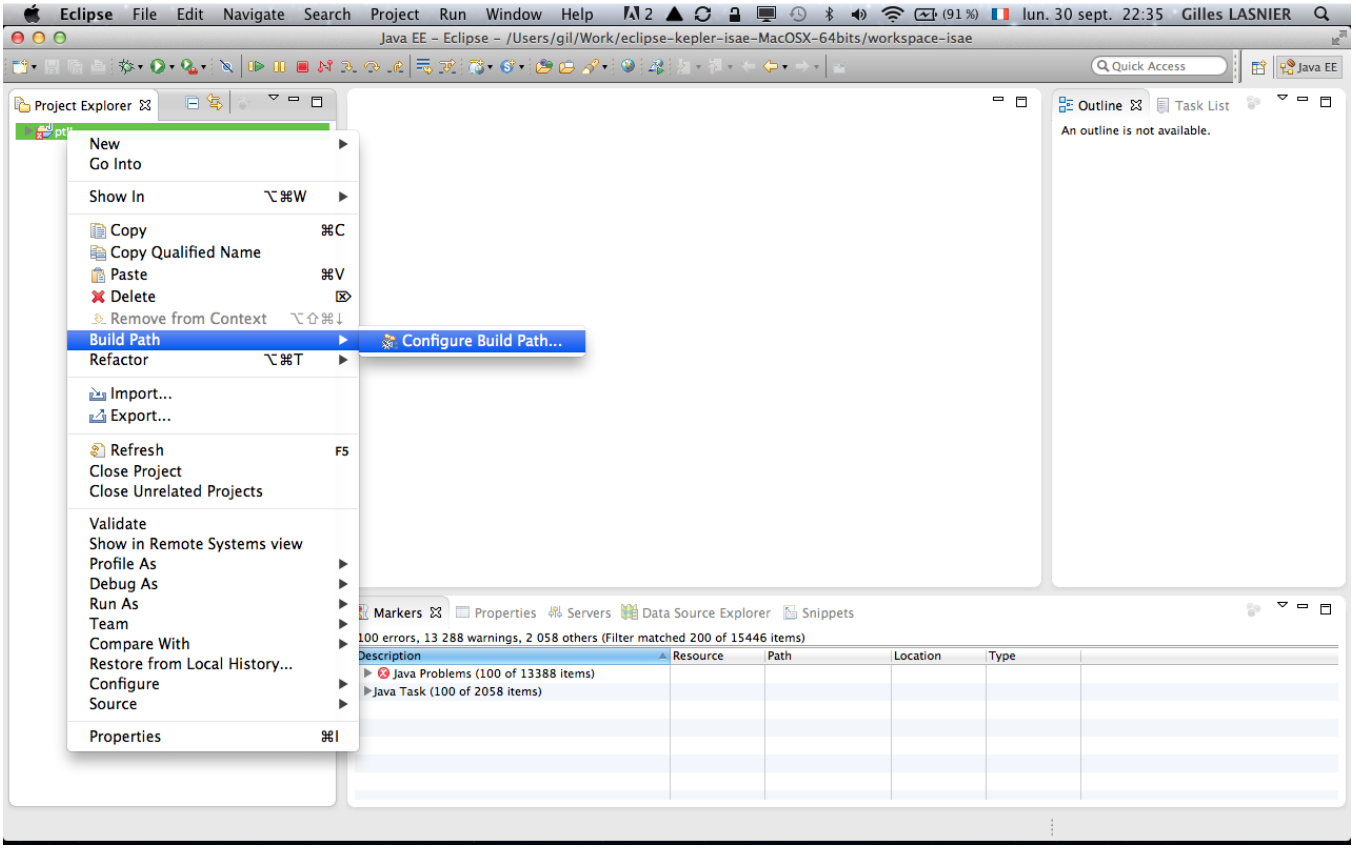


Figure B.7: Resolve JCERTI interface dependencies in 'ptII' (1)

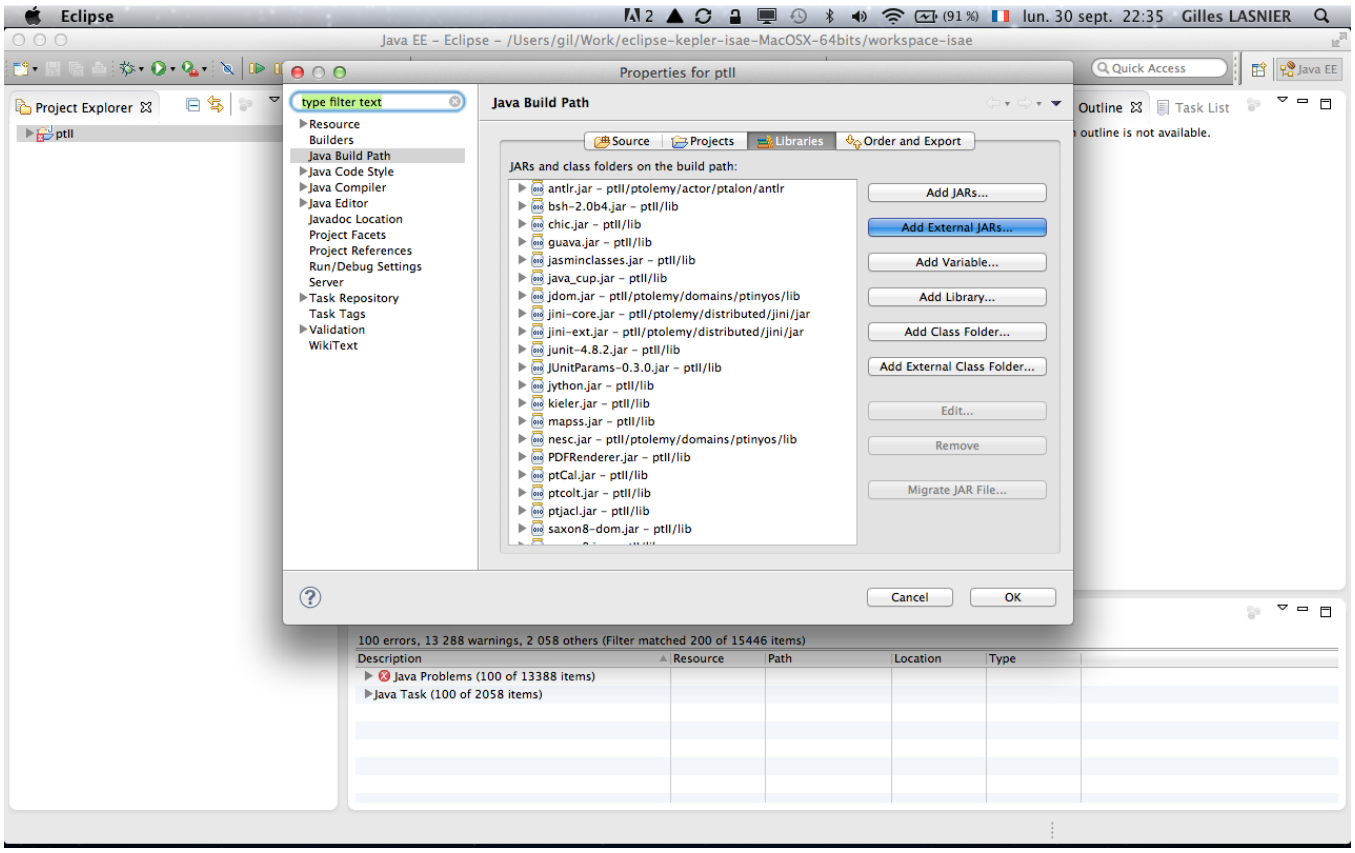


Figure B.8: Resolve JCERTI interface dependencies in ’ptII’ (2)

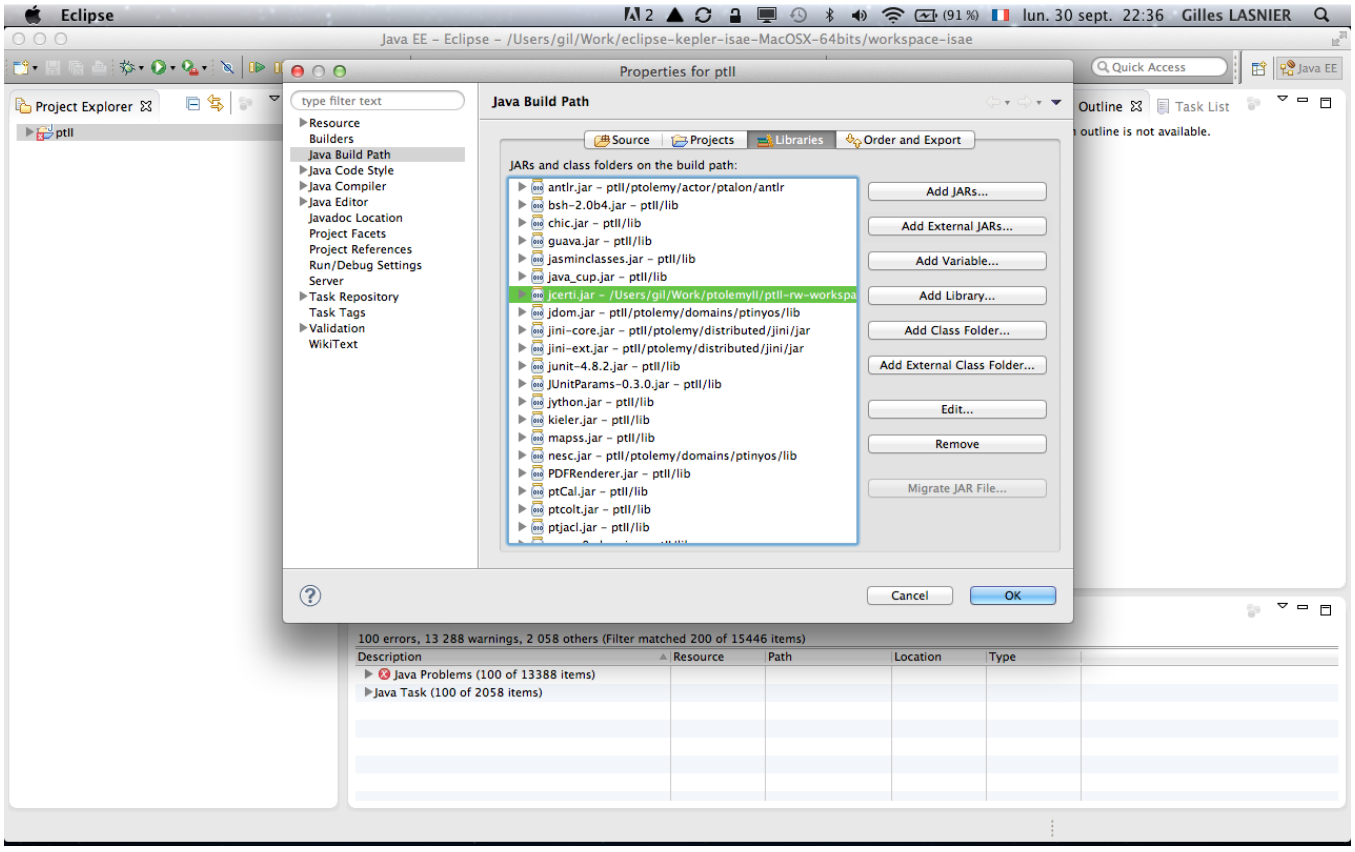


Figure B.9: Resolve JCERTI interface dependencies in ’ptII’ (3)

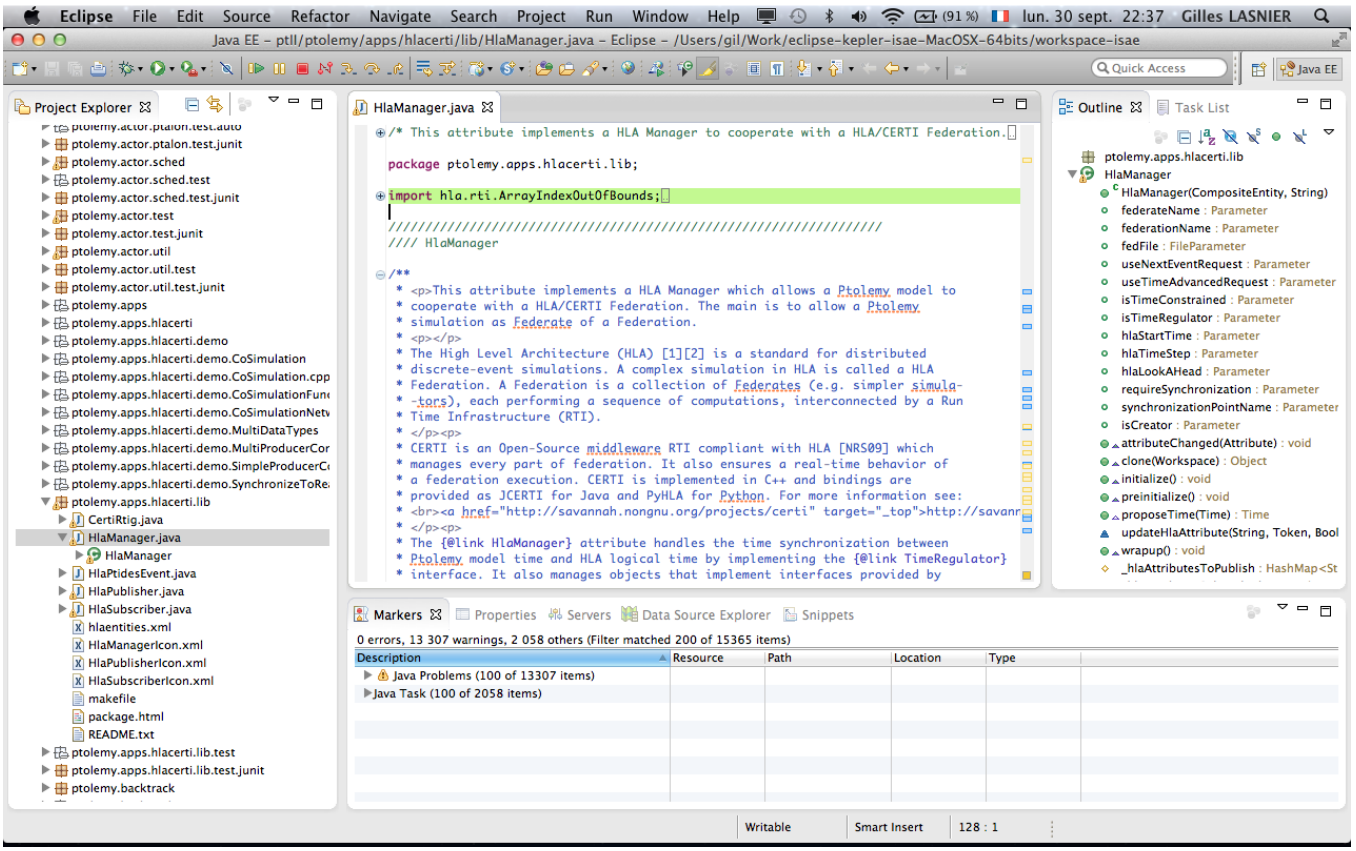


Figure B.10: Correct compilation of the HlaManager.java class in 'ptII'

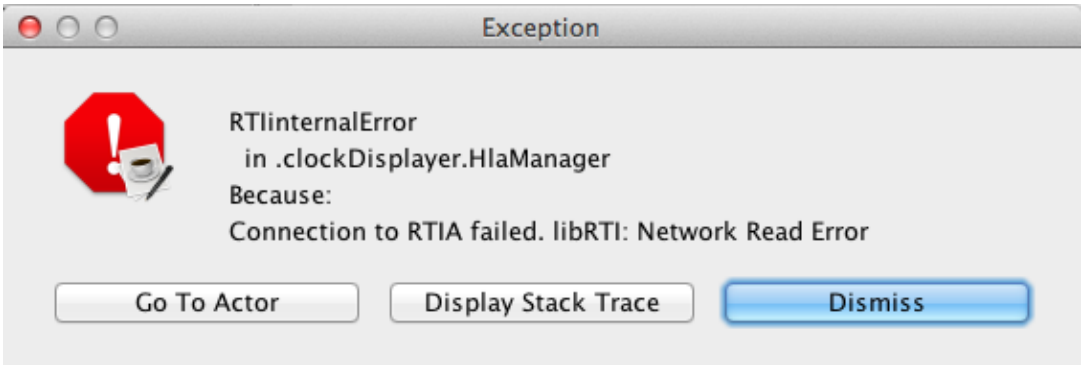


Figure B.11: RTIinternalError: Connection to RTIA failed

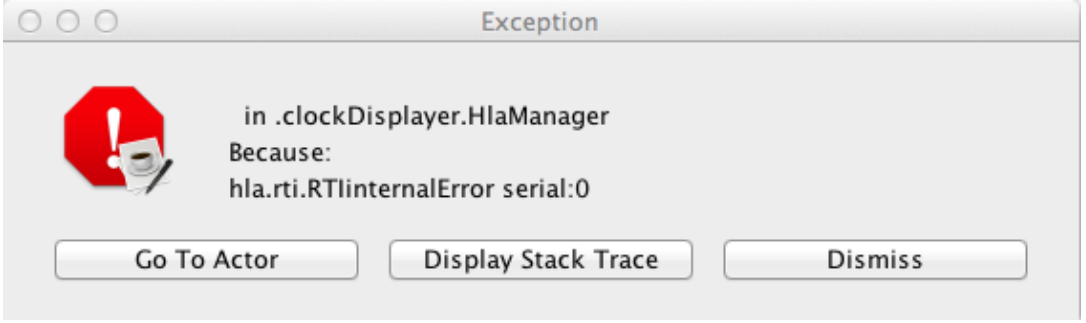


Figure B.12: RTIinternalError: hla.rti.RTIinternalError serial:0

Appendix C

Code snippets

Listing C.1: ptolemy.actor.TimeRegulator.java

```
/* Interface for attributes that regulate the passage of time.

 Copyright (c) 2007-2013 The Regents of the University of California.
 All rights reserved.
 Permission is hereby granted, without written agreement and without
 license or royalty fees, to use, copy, modify, and distribute this
 software and its documentation for any purpose, provided that the above
 copyright notice and the following two paragraphs appear in all copies
 of this software.

 IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY
 FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
 THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE
 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF
 CALIFORNIA HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
 ENHANCEMENTS, OR MODIFICATIONS.

 PT_COPYRIGHT_VERSION_2
 COPYRIGHTENDKEY
 */
package ptolemy.actor;

import ptolemy.actor.util.Time;
import ptolemy.kernel.util.IllegalActionException;

//////////////////////////////////////////////////////////////////
//// TimeRegulator

/**
 This interface is implemented by attributes that wish to be consulted
 when a director advances time. In particular, the director will call
 the one method in this interface, passing it a proposed time to advance to,
 and the method will return either the same proposed time or
 a smaller time. The method may not return immediately. For example,
 it might wait for real time to advance to the proposed time, and then
 simply return the proposed time.

 @author Edward A. Lee, Gilles Lasnier, Patricia Derler
```

```
 @version $Id: TimeRegulator.java 422 2013-09-05 13:43:09Z g.lasnier $
 @since Ptolemy II 10.0
 @Pt.ProposedRating Yellow (eal)
 @Pt.AcceptedRating Red (cxh)
 */
public interface TimeRegulator {

    ////////////////////////////////////////////////////////////////////
    ////                         public methods                     ////

    /** Propose a time to advance to.
     *  @param proposedTime The proposed time.
     *  @return The proposed time or a smaller time.
     *  @exception IllegalActionException If the time regulator is being misused.
     */
    public Time proposeTime(Time proposedTime) throws IllegalActionException;
}
```

Listing C.2: ptolemy.apps.hlacerti.lib.HlaPublisher.java

```
/* This actor implements a publisher in a HLA/CERTI federation.

package ptolemy.apps.hlacerti.lib;

import java.util.List;

////////////////////////////////////////////////////////////////////
//// HlaPublisher

/**
 * <p>This actor implements a publisher in a HLA/CERTI federation. This
 * publisher is associated to one HLA attribute. Ptolemy's tokens, received in
 * the input port of this actor, are interpreted as an updated value of the
 * HLA attribute. The updated value is published to the whole HLA Federation
 * by the {@link HlaManager} attribute, deployed in a Ptolemy model.
 * </p><p>
 * The name of this actor is mapped to the name of the HLA attribute in the
 * federation and need to match the Federate Object Model (FOM) specified for
 * the Federation. The data type of the input port has to be the same type of
 * the HLA attribute. The parameter <i>classObjectHandle</i> needs to match the
 * attribute object class describes in the FOM. The parameter
 * <i>asHlaPtidesEvent</i> indicates if we need to handle PTIDES events as
 * RecordToken in HLA events.
 * </p>
 *
 *  @author Gilles Lasnier, Contributors: Patricia Derler
 *  @version $Id: HlaPublisher.java 423 2013-09-05 16:38:46Z g.lasnier $
 *  @since Ptolemy II 10.0
 *
 *  @Pt.ProposedRating Yellow (glasnier)
 *  @Pt.AcceptedRating Red (glasnier)
 */
public class HlaPublisher extends TypedAtomicActor {

    /** Construct the HlaPublisher actor.
     *  @param container The container.
     *  @param name The name of this actor.
     *  @exception IllegalActionException If the entity cannot be contained
     *  by the proposed container.
     *  @exception NameDuplicationException If the container already has an
     *  actor with this name.
     */
    public HlaPublisher(CompositeEntity container, String name)
            throws NameDuplicationException, IllegalActionException {
```

```
    super(container, name);

    // The single output port of the actor.
    input = new TypedIOPort(this, "input", true, false);

    classObjectHandle = new Parameter(this, "classObjectHandle");
    classObjectHandle.setDisplayName("Object class in FOM");
    classObjectHandle.setTypeEquals(BaseType.STRING);
    classObjectHandle.setExpression("\"myObjectClass\"");
    attributeChanged(classObjectHandle);

    asHLAPtidesEvent = new Parameter(this, "asHLAPtidesEvent");
    asHLAPtidesEvent.setTypeEquals(BaseType.BOOLEAN);
    asHLAPtidesEvent.setExpression("false");
    asHLAPtidesEvent.setDisplayName("asHLAPtidesEvent ?");
    attributeChanged(asHLAPtidesEvent);

    _hlaManager = null;
    _asHLAPtidesEvent = false;
}

///////////////////////////////////////////////////////////////////
////                         public variables                  ////

/** The object class of the HLA attribute to publish. */
public Parameter classObjectHandle;

/** Indicate if the event is for a Ptides platform. */
public Parameter asHLAPtidesEvent;

/** The input port. */
public TypedIOPort input = null;

///////////////////////////////////////////////////////////////////
////                         public methods                    ////

/** Call the attributeChanged method of the parent. Check if the
 *  user as set the object class of the HLA attribute to subscribe to.
 *  @param attribute The attribute that changed.
 *  @exception IllegalActionException If the object class parameter is
 *  empty.
 */
public void attributeChanged(Attribute attribute)
        throws IllegalActionException {
    if (attribute == classObjectHandle) {
        String value = ((StringToken) classObjectHandle.getToken())
                .stringValue();
        if (value.compareTo("") == 0) {
            throw new IllegalActionException(this,
                    "Cannot have empty name !");
        }
    } else if (attribute == asHLAPtidesEvent) {
        _asHLAPtidesEvent = ((BooleanToken) asHLAPtidesEvent.getToken())
                .booleanValue();
    }
    super.attributeChanged(attribute);
}

/** Clone the actor into the specified workspace.
 *  @param workspace The workspace for the new object.
 *  @return A new actor.
 *  @exception CloneNotSupportedException If a derived class contains
 *  an attribute that cannot be cloned.
 */
public Object clone(Workspace workspace) throws CloneNotSupportedException {
```

```
        HlaPublisher newObject = (HlaPublisher) super.clone(workspace);
        newObject._hlaManager = _hlaManager;

        return newObject;
    }

    /** Retrieve and check if there is one and only one {@link HlaManager}
     *  deployed in the Ptolemy model. The {@link HlaManager} provides the
     *  method to publish an updated value of a HLA attribute to the HLA/CERTI
     *  Federation.
     *  @exception IllegalActionException If there is zero or more than one
     *  {@link HlaManager} per Ptolemy model.
     */
    public void initialize() throws IllegalActionException {
        super.initialize();

        CompositeActor ca = (CompositeActor) this.getContainer();

        List<HlaManager> hlaManagers = ca.attributeList(HlaManager.class);
        if (hlaManagers.size() > 1) {
            throw new IllegalActionException(this,
                    "Only one HlaManager attribute is allowed per model");
        } else if (hlaManagers.size() < 1) {
            throw new IllegalActionException(this,
                    "A HlaManager attribute is required to use this actor");
        }

        // Here, we are sure that there is one and only one instance of the
        // HlaManager in the Ptolemy model.
        _hlaManager = hlaManagers.get(0);
    }

    /** Each tokens, received in the input port, are transmitted to the
     *  {@link HlaManager} for a publication to the HLA/CERTI Federation.
     */
    public void fire() throws IllegalActionException {
        if (inputPortList().get(0).hasToken(0)) {
            Token in = inputPortList().get(0).get(0);
            _hlaManager.updateHlaAttribute(this.getName(), in,
                    _asHLAPtidesEvent);

            if (_debugging) {
                _debug(this.getDisplayName()
                        + " Called fire() - the update value \""
                        + in.toString() + "\" of the HLA Attribute \""
                        + this.getName() + "\" has been sent to \""
                        + _hlaManager.getDisplayName() + "\"");
            }
        }
    }

    ///////////////////////////////////////////////////////////////////
    ////                         private variables                 ////

    /** A reference to the associated {@link HlaManager}. */
    private HlaManager _hlaManager;

    /** Indicate if the event is for a Ptides platform. */
    private boolean _asHLAPtidesEvent;
}
```

Listing C.3: ptolemy.apps.hlacerti.lib.HlaManager.java

```
/* This attribute implements a HLA Manager to cooperate with a HLA/CERTI Federation.
```

```
 */

package ptolemy.apps.hlacerti.lib;

import hla.rti.ArrayIndexOutOfBounds;
import hla.rti.AsynchronousDeliveryAlreadyEnabled;
import hla.rti.AttributeHandleSet;
import hla.rti.AttributeNotDefined;
import hla.rti.AttributeNotKnown;
import hla.rti.AttributeNotOwned;
import hla.rti.ConcurrentAccessAttempted;
import hla.rti.CouldNotDiscover;
import hla.rti.CouldNotOpenFED;
import hla.rti.EnableTimeConstrainedPending;
import hla.rti.EnableTimeConstrainedWasNotPending;
import hla.rti.EnableTimeRegulationPending;
import hla.rti.EnableTimeRegulationWasNotPending;
import hla.rti.ErrorReadingFED;
import hla.rti.EventRetractionHandle;
import hla.rti.FederateAlreadyExecutionMember;
import hla.rti.FederateAmbassador;
import hla.rti.FederateInternalError;
import hla.rti.FederateNotExecutionMember;
import hla.rti.FederateOwnsAttributes;
import hla.rti.FederatesCurrentlyJoined;
import hla.rti.FederationExecutionAlreadyExists;
import hla.rti.FederationExecutionDoesNotExist;
import hla.rti.FederationTimeAlreadyPassed;
import hla.rti.InvalidFederationTime;
import hla.rti.InvalidLookahead;
import hla.rti.InvalidResignAction;
import hla.rti.LogicalTime;
import hla.rti.LogicalTimeInterval;
import hla.rti.NameNotFound;
import hla.rti.ObjectAlreadyRegistered;
import hla.rti.ObjectClassNotDefined;
import hla.rti.ObjectClassNotKnown;
import hla.rti.ObjectClassNotPublished;
```

```
import hla.rti.ObjectClassNotSubscribed;
import hla.rti.ObjectNotKnown;
import hla.rti.OwnershipAcquisitionPending;
import hla.rti.RTIambassador;
import hla.rti.RTIinternalError;
import hla.rti.ReflectedAttributes;
import hla.rti.ResignAction;
import hla.rti.RestoreInProgress;
import hla.rti.SaveInProgress;
import hla.rti.SpecifiedSaveLabelDoesNotExist;
import hla.rti.SuppliedAttributes;
import hla.rti.SynchronizationLabelNotAnnounced;
import hla.rti.TimeAdvanceAlreadyInProgress;
import hla.rti.TimeAdvanceWasNotInProgress;
import hla.rti.TimeConstrainedAlreadyEnabled;
import hla.rti.TimeRegulationAlreadyEnabled;
import hla.rti.jlc.EncodingHelpers;
import hla.rti.jlc.NullFederateAmbassador;
import hla.rti.jlc.RtiFactory;
import hla.rti.jlc.RtiFactoryFactory;

import java.util.logging.Logger;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.MalformedURLException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.NoSuchElementException;
import ptolemy.actor.AbstractInitializableAttribute;
import ptolemy.actor.Actor;
import ptolemy.actor.CompositeActor;
import ptolemy.actor.TimeRegulator;
import ptolemy.actor.TypedIOPort;
import ptolemy.actor.util.Time;
import ptolemy.actor.util.TimedEvent;
import ptolemy.data.BooleanToken;
import ptolemy.data.DoubleToken;
import ptolemy.data.FloatToken;
import ptolemy.data.IntToken;
import ptolemy.data.LongToken;
import ptolemy.data.RecordToken;
import ptolemy.data.ShortToken;
import ptolemy.data.StringToken;
import ptolemy.data.Token;
import ptolemy.data.UnsignedByteToken;
import ptolemy.data.expr.FileParameter;
import ptolemy.data.expr.Parameter;
import ptolemy.data.type.BaseType;
import ptolemy.data.type.RecordType;
import ptolemy.data.type.Type;
import ptolemy.domains.de.kernel.DEDirector;
import ptolemy.kernel.CompositeEntity;
import ptolemy.kernel.util.Attribute;
import ptolemy.kernel.util.IllegalActionException;
import ptolemy.kernel.util.NameDuplicationException;
import ptolemy.kernel.util.NamedObj;
import ptolemy.kernel.util.Workspace;
```

```
import certi.communication.CertiException;
import certi.communication.MessageBuffer;
import certi.rti.impl.CertiLogicalTime;
import certi.rti.impl.CertiLogicalTimeInterval;
import certi.rti.impl.CertiRtiAmbassador;

////////////////////////////////////////////////////////////////
//// HlaManager

/**
 * <p>This attribute implements a HLA Manager which allows a Ptolemy model to
 * cooperate with a HLA/CERTI Federation. The main is to allow a Ptolemy
 * simulation as Federate of a Federation.
 * <p></p>
 * The High Level Architecture (HLA) [1][2] is a standard for distributed
 * discrete-event simulations. A complex simulation in HLA is called a HLA
 * Federation. A Federation is a collection of Federates (e.g. simpler simula-
 * -tors), each performing a sequence of computations, interconnected by a Run
 * Time Infrastructure (RTI).
 * </p><p>
 * CERTI is an Open-Source middleware RTI compliant with HLA [NRS09] which
 * manages every part of federation. It also ensures a real-time behavior of
 * a federation execution. CERTI is implemented in C++ and bindings are
 * provided as JCERTI for Java and PyHLA for Python. For more information see:
 * <br><a href="http://savannah.nongnu.org/projects/certi" target="_top">http://savannah.
    nongnu.org/projects/certi</a></br>
 * </p><p>
 * The {@link HlaManager} attribute handles the time synchronization between
 * Ptolemy model time and HLA logical time by implementing the {@link TimeRegulator}
 * interface. It also manages objects that implement interfaces provided by
 * JCERTI relatives to Federation, Declaration, Object and Time management
 * areas in HLA (each management areas provides a set of services).
 * </p><p>
 * To develop a HLA Federation it is required to specify a Federate Object
 * Model (FOM) which describes the architecture of the Federation (HLA version,
 * name of Federates which belong to, shared HLA attributes) and the interaction
 * between Federates and shared attributes. Data exchanged in a HLA Federation
 * are called HLA attributes and their interaction mechanism is based on the
 * publish/subscribe paradigm. The FOM is specified in a .fed file used by
 * the RTI (e.g. by the RTIG process when using CERTI). More information in [3].
 * <br><a href="http://savannah.nongnu.org/projects/certi" target="_top">http://savannah.
    nongnu.org/projects/certi</a></br>
 * </p><p>
 * To enable a Ptolemy model as a Federate, the {@link HlaManager} has to be
 * deployed and configured (by double-clicking on the attribute).
 * Parameters <i>federateName</i>, <i>federationName</i> have to match the
 * declaration in the FOM (.fed file). <i>fedFile</i> specifies the FOM file and
 * its path.
 * </p><p>
 * Parameters <i>useNextEventRequest</i>, <i>UseTimeAdvanceRequest</i>,
 * <i>isTimeConstrained</i> and <i>isTimeRegulator</i> are
 * used to configure the HLA time management services of the Federate. A
 * Federate can only specify the use of the <i>nextEventRequest()
 * service</i> or the <i>timeAdvanceRequest()</i> service at a time.
 * <i>istimeConstrained</i> is used to specify time-constrained Federate and
 * <i>istimeRegulator</i> to specify time-regulator Federate. The combination of
 * both parameters is possible and is recommended.
 * </p><p>
 * Parameters <i>hlaStartTime</i>, <i>hlaStepTime</i> and <i>hlaLookAHead</i>
 * are used to specify Hla Timing attributes of a Federate.
 * </p><p>
 * Parameters <i>requireSynchronization</i>, <i>synchronizationPointName</i>
 * and <i>isCreatorSyncPt</i> are used to configure HLA synchronization point.
 * This mechanism is usually used to synchronize the Federates, during their
 * initialization, to avoid that Federates that only consume some HLA
```

```
 * attributes finished their simulation before the other federates have started.
 * <i>isCreatorSyncPt</i> indicates if the Federate is the creator of the
 * synchronization. Only one Federate can create the named synchronization
 * point for the whole HLA Federation.
 * </p><p>
 * {@link HlaPublisher} and {@link HlaSubscriber} actors are used to
 * respectively publish and subscribe to HLA attributes. The name of those
 * actors and their <i>classObjectHandle</i> parameter have to match the
 * identifier of the shared HLA attributes and their object class that they
 * belong to, specified in the FOM (.fed file).
 * </p><p>
 * For a correct execution, the <i>CERTI_HOME</i> environment variable has to
 * be set. It could be set in the shell (by running one of the scripts provided
 * by CERTI) where Vergil is executed, or as a parameter of the Ptolemy model
 * or as a parameter of the {@link HlaManager}:
 * </p><pre>
 * CERTI_HOME="/absolute/path/to/certi/"
 * </pre><p>
 * Otherwise, the current implementation is not able to find the CERTI
 * environment, the RTIG binary and to perform its execution. See also
 * the {@link CertiRtig} class.
 * </p><p>
 * NOTE: For a correct behavior CERTI has to be compiled with the option
 * "CERTI_USE_NULL_PRIME_MESSAGE_PROTOCOL"
 * </p>
 *
 * <b>References</b>:
 * <br>
 * [1] Dpt. of Defense (DoD) Specifications, "High Level Architecture Interface
 *     Specification, Version 1.3", DOD/DMSO HLA IF 1.3, Tech. Rep., Apr 1998.
 * [2] IEEE, "IEEE standard for modeling and simulation High Level Architecture
 *     (HLA)", IEEE Std 1516-2010, vol. 18, pp. 1-38, 2010.
 * [3] D. of Defense (DoD) Specifications, "High Level Architecture Object Model
 *     Template, Version 1.3", DOD/DMSO OMT 1.3, Tech. Rep., Feb 1998.
 * [4] E. Noulard, J.-Y. Rousselot, and P. Siron, "CERTI, an open source RTI,
 *     why and how ?", Spring Simulation Interoperability Workshop, pp. 23-27,
 *     Mar 2009.
 *
 *  @author Gilles Lasnier, Contributors: Patricia Derler, Edward A. Lee
 *  @version $Id: HlaManager.java 429 2013-09-09 23:23:25Z g.lasnier $
 *  @since Ptolemy II 10.0
 *
 *  @Pt.ProposedRating Yellow (glasnier)
 *  @Pt.AcceptedRating Red (glasnier)
 */
public class HlaManager extends AbstractInitializableAttribute implements
TimeRegulator {

    /** Construct a HlaManager with a name and a container. The container
     *  argument must not be null, or a NullPointerException will be thrown.
     *  This actor will use the workspace of the container for synchronization
     *  and version counts. If the name argument is null, then the name is set
     *  to the empty string. Increment the version of the workspace.
     *  @param container Container of this attribute.
     *  @param name Name of this attribute.
     *  @exception IllegalActionException If the container is incompatible
     *  with this actor.
     *  @exception NameDuplicationException If the name coincides with
     *  an actor already in the container.
     */
    public HlaManager(CompositeEntity container, String name)
            throws IllegalActionException, NameDuplicationException {
        super(container, name);

        _lastProposedTime = null;
```

```
_rtia = null;
_federateAmbassador = null;

_hlaAttributesToPublish = new HashMap<String, Object[]>();
_hlaAttributesSubscribedTo = new HashMap<String, Object[]>();
_fromFederationEvents = new HashMap<String, LinkedList<TimedEvent>>();
_objectIdToClassHandle = new HashMap<Integer, Integer>();

_hlaStartTime = null;
_hlaTimeStep = null;
_hlaLookAHead = null;

// HLA Federation management parameters.
federateName = new Parameter(this, "federateName");
federateName.setDisplayName("Federate's name");
federateName.setTypeEquals(BaseType.STRING);
federateName.setExpression("\"HlaManager\"");
attributeChanged(federateName);

federationName = new Parameter(this, "federationName");
federationName.setDisplayName("Federation's name");
federationName.setTypeEquals(BaseType.STRING);
federationName.setExpression("\"HLAFederation\"");
attributeChanged(federationName);

fedFile = new FileParameter(this, "fedFile");
fedFile.setDisplayName("Federate Object Model (.fed) file path");
new Parameter(fedFile, "allowFiles", BooleanToken.TRUE);
new Parameter(fedFile, "allowDirectories", BooleanToken.FALSE);
fedFile.setExpression("$CWD/HLAFederation.fed");

// HLA Time management parameters.
useNextEventRequest = new Parameter(this, "useNextEventRequest");
useNextEventRequest.setTypeEquals(BaseType.BOOLEAN);
useNextEventRequest.setExpression("true");
useNextEventRequest.setDisplayName("useNextEventRequest (NER) ?");
attributeChanged(useNextEventRequest);

useTimeAdvancedRequest = new Parameter(this, "useTimeAdvancedRequest");
useTimeAdvancedRequest.setTypeEquals(BaseType.BOOLEAN);
useTimeAdvancedRequest.setExpression("false");
useTimeAdvancedRequest.setDisplayName("useTimeAdvancedRequest (TAR) ?");
attributeChanged(useTimeAdvancedRequest);

isTimeConstrained = new Parameter(this, "isTimeConstrained");
isTimeConstrained.setTypeEquals(BaseType.BOOLEAN);
isTimeConstrained.setExpression("true");
isTimeConstrained.setDisplayName("isTimeConstrained ?");
attributeChanged(isTimeConstrained);

isTimeRegulator = new Parameter(this, "isTimeRegulator");
isTimeRegulator.setTypeEquals(BaseType.BOOLEAN);
isTimeRegulator.setExpression("true");
isTimeRegulator.setDisplayName("isTimeRegulator ?");
attributeChanged(isTimeRegulator);

hlaStartTime = new Parameter(this, "hlaStartTime");
hlaStartTime.setDisplayName("logical start time (in ms)");
hlaStartTime.setExpression("0.0");
hlaStartTime.setTypeEquals(BaseType.DOUBLE);
attributeChanged(hlaStartTime);

hlaTimeStep = new Parameter(this, "hlaTimeStep");
hlaTimeStep.setDisplayName("time step (in ms)");
hlaTimeStep.setExpression("0.0");
```

```
        hlaTimeStep.setTypeEquals(BaseType.DOUBLE);
        attributeChanged(hlaTimeStep);

        hlaLookAHead = new Parameter(this, "hlaLookAHead");
        hlaLookAHead.setDisplayName("lookahead (in ms)");
        hlaLookAHead.setExpression("0.0");
        hlaLookAHead.setTypeEquals(BaseType.DOUBLE);
        attributeChanged(hlaLookAHead);

        // HLA Synchronization parameters.
        requireSynchronization = new Parameter(this, "requireSynchronization");
        requireSynchronization.setTypeEquals(BaseType.BOOLEAN);
        requireSynchronization.setExpression("true");
        requireSynchronization.setDisplayName("Require synchronization ?");
        attributeChanged(requireSynchronization);

        synchronizationPointName = new Parameter(this,
                "synchronizationPointName");
        synchronizationPointName.setDisplayName("Synchronization point name");
        synchronizationPointName.setTypeEquals(BaseType.STRING);
        synchronizationPointName.setExpression("\"Simulating\"");
        attributeChanged(synchronizationPointName);

        isCreator = new Parameter(this, "isCreator");
        isCreator.setTypeEquals(BaseType.BOOLEAN);
        isCreator.setExpression("false");
        isCreator.setDisplayName("Is synchronization point creator ?");
        attributeChanged(isCreator);
    }

    ///////////////////////////////////////////////////////////////////
    ////                     public variables                      ////

    /** Name of the Ptolemy Federate. This parameter must contain an
     *  StringToken. */
    public Parameter federateName;

    /** Name of the federation. This parameter must contain an StringToken. */
    public Parameter federationName;

    /** Path and name of the Federate Object Model (FOM) file. This parameter
     *  must contain an StringToken. */
    public FileParameter fedFile;

    /** Boolean value, 'true' if the Federate requires the use of the
     *  nextEventRequest() HLA service. This parameter must contain an
     *  BooleanToken. */
    public Parameter useNextEventRequest;

    /** Boolean value, 'true' if the Federate requires the use of the
     *  timeAdvanceRequest() HLA service. This parameter must contain an
     *  BooleanToken. */
    public Parameter useTimeAdvancedRequest;

    /** Boolean value, 'true' if the Federate is declared time constrained
     *  'false' if not. This parameter must contain an BooleanToken. */
    public Parameter isTimeConstrained;

    /** Boolean value, 'true' if the Federate is declared time regulator
     *  'false' if not. This parameter must contain an BooleanToken. */
    public Parameter isTimeRegulator;

    /** Value of the start time of the Federate. This parameter must contain
     *  an DoubleToken. */
    public Parameter hlaStartTime;
```

```
/** Value of the time step of the Federate. This parameter must contain
 *  an DoubleToken. */
public Parameter hlaTimeStep;

/** Value of the lookahead of the HLA ptII federate. This parameter
 *  must contain an DoubleToken. */
public Parameter hlaLookAHead;

/** Boolean value, 'true' if the Federate is synchronised with other
 *  Federates using a HLA synchronization point, 'false' if not. This
 *  parameter must contain an BooleanToken. */
public Parameter requireSynchronization;

/** Name of the synchronization point (if required). This parameter must
 *  contain an StringToken. */
public Parameter synchronizationPointName;

/** Boolean value, 'true' if the Federate is the creator of the
 *  synchronization point 'false' if not. This parameter must contain
 *  an BooleanToken. */
public Parameter isCreator;

////////////////////////////////////////////////////////////////
////                         public methods                 ////

/** Checks constraints on the changed attribute (when it is required) and
 *  associates his value to its corresponding local variables.
 *  @param attribute The attribute that changed.
 *  @exception IllegalActionException If the attribute is empty or negative.
 */
public void attributeChanged(Attribute attribute)
        throws IllegalActionException {
    super.attributeChanged(attribute);

    if (attribute == federateName) {
        String value = ((StringToken) federateName.getToken())
                .stringValue();
        if (value.compareTo("") == 0) {
            throw new IllegalActionException(this,
                    "Cannot have empty name !");
        }
        _federateName = value;
        setDisplayName(value);
    } else if (attribute == federationName) {
        String value = ((StringToken) federationName.getToken())
                .stringValue();
        if (value.compareTo("") == 0) {
            throw new IllegalActionException(this,
                    "Cannot have empty name !");
        }
        _federationName = value;
    } else if (attribute == useNextEventRequest) {
        _useNextEventRequest = ((BooleanToken) useNextEventRequest
                .getToken()).booleanValue();

    } else if (attribute == useTimeAdvancedRequest) {
        _useTimeAdvancedRequest = ((BooleanToken) useTimeAdvancedRequest
                .getToken()).booleanValue();

    } else if (attribute == isTimeConstrained) {
        _isTimeConstrained = ((BooleanToken) isTimeConstrained.getToken())
                .booleanValue();

    } else if (attribute == isTimeRegulator) {
```

```
            _isTimeRegulator = ((BooleanToken) isTimeRegulator.getToken())
                    .booleanValue();

        } else if (attribute == hlaStartTime) {
            Double value = ((DoubleToken) hlaStartTime.getToken())
                    .doubleValue();
            if (value < 0) {
                throw new IllegalActionException(this,
                        "Cannot have negative value !");
            }
            _hlaStartTime = value;
        } else if (attribute == hlaTimeStep) {
            Double value = ((DoubleToken) hlaTimeStep.getToken()).doubleValue();
            if (value < 0) {
                throw new IllegalActionException(this,
                        "Cannot have negative value !");
            }
            _hlaTimeStep = value;
        } else if (attribute == hlaLookAHead) {
            Double value = ((DoubleToken) hlaLookAHead.getToken())
                    .doubleValue();
            if (value < 0) {
                throw new IllegalActionException(this,
                        "Cannot have negative value !");
            }
            _hlaLookAHead = value;
        } else if (attribute == requireSynchronization) {
            _requireSynchronization = ((BooleanToken) requireSynchronization
                    .getToken()).booleanValue();

        } else if (attribute == synchronizationPointName) {
            String value = ((StringToken) synchronizationPointName.getToken())
                    .stringValue();
            if (value.compareTo("") == 0) {
                throw new IllegalActionException(this,
                        "Cannot have empty name !");
            }
            _synchronizationPointName = value;
        } else if (attribute == isCreator) {
            _isCreator = ((BooleanToken) isCreator.getToken()).booleanValue();

        } else {
            super.attributeChanged(attribute);
        }
    }

    /** Clone the actor into the specified workspace.
     *  @param workspace The workspace for the new object.
     *  @return A new actor.
     *  @exception CloneNotSupportedException If a derived class contains
     *  an attribute that cannot be cloned.
     */
    public Object clone(Workspace workspace) throws CloneNotSupportedException {
        HlaManager newObject = (HlaManager) super.clone(workspace);

        newObject._hlaAttributesToPublish = new HashMap<String, Object[]>();
        newObject._hlaAttributesSubscribedTo = new HashMap<String, Object[]>();
        newObject._fromFederationEvents = new HashMap<String, LinkedList<TimedEvent>>();
        newObject._objectIdToClassHandle = new HashMap<Integer, Integer>();

        newObject._rtia = null;
        newObject._federateAmbassador = null;
        newObject._federateName = _federateName;
        newObject._federationName = _federationName;
        newObject._isTimeConstrained = _isTimeConstrained;
```

```
    newObject._isTimeRegulator = _isTimeRegulator;
    try {
        newObject._hlaStartTime = ((DoubleToken) hlaStartTime.getToken())
                .doubleValue();
        newObject._hlaTimeStep = ((DoubleToken) hlaTimeStep.getToken())
                .doubleValue();
        newObject._hlaLookAHead = ((DoubleToken) hlaLookAHead.getToken())
                .doubleValue();
    } catch (IllegalActionException ex) {
        CloneNotSupportedException ex2 = new CloneNotSupportedException(
                "Failed to get a token.");
        ex2.initCause(ex);
        throw ex2;
    }
    newObject._requireSynchronization = _requireSynchronization;
    newObject._synchronizationPointName = _synchronizationPointName;
    newObject._isCreator = _isCreator;
    newObject._useNextEventRequest = _useNextEventRequest;
    newObject._useTimeAdvancedRequest = _useTimeAdvancedRequest;

    return newObject;
}

/** Initializes the {@link HlaManager} attribute. This method: calls the
 *  _populateHlaAttributeTables() to initialize HLA attributes to publish
 *  or subscribe to; instantiates and initializes the {@link RTIambassador}
 *  and {@link FederateAmbassador} which handle the communication
 *  Federate <-> RTIA <-> RTIG. RTIA and RTIG are both external communicant
 *  processes (see JCERTI); create the HLA/CERTI Federation (if not exists);
 *  allows the Federate to join the Federation; set the Federate time
 *  management policies (regulator and/or contrained); creates a
 *  synchronisation point (if required); and synchronizes the Federate with
 *  a synchronization point (if declared).
 *  @exception IllegalActionException If the container of the class is not
 *  an Actor or If a CERTI exception is raised and has to be displayed to
 *  the user.
 */
public void initialize() throws IllegalActionException {
    super.initialize();

    NamedObj container = getContainer();
    if (!(container instanceof Actor)) {
        throw new IllegalActionException(this,
                "HlaManager has to be contained by an Actor");
    }

    // Get the corresponding director associate to the HlaManager attribute.
    _director = (DEDirector) ((CompositeActor) this.getContainer())
            .getDirector();

    // Initialize HLA attribute tables for publication/subscription.
    _populateHlaAttributeTables();

    // Get a link to the RTI.
    RtiFactory factory = null;
    try {
        factory = RtiFactoryFactory.getRtiFactory();
    } catch (RTIinternalError e) {
        throw new IllegalActionException(this, e, "RTIinternalError ");
    }

    try {
        _rtia = (CertiRtiAmbassador) factory.createRtiAmbassador();
    } catch (RTIinternalError e) {
        throw new IllegalActionException(this, e, "RTIinternalError ");
```

```
}

// Create the Federation or raise a warning it the Federation already exits.
try {
    _rtia.createFederationExecution(_federationName, fedFile.asFile()
            .toURI().toURL());
} catch (FederationExecutionAlreadyExists e) {
    if (_debugging) {
        _debug(this.getDisplayName()
                + " initialize() - WARNING: FederationExecutionAlreadyExists");
    }
} catch (CouldNotOpenFED e) {
    throw new IllegalActionException(this, e, "CouldNotOpenFED ");
} catch (ErrorReadingFED e) {
    throw new IllegalActionException(this, e, "ErrorReadingFED ");
} catch (RTIinternalError e) {
    throw new IllegalActionException(this, e, "RTIinternalError ");
} catch (ConcurrentAccessAttempted e) {
    throw new IllegalActionException(this, e,
            "ConcurrentAccessAttempted ");
} catch (MalformedURLException e) {
    throw new IllegalActionException(this, e, "MalformedURLException ");
}

_federateAmbassador = new PtolemyFederateAmbassadorInner();

// Join the Federation.
try {
    _rtia.joinFederationExecution(_federateName, _federationName,
            _federateAmbassador);
} catch (FederateAlreadyExecutionMember e) {
    throw new IllegalActionException(this, e,
            "FederateAlreadyExecutionMember ");
} catch (FederationExecutionDoesNotExist e) {
    throw new IllegalActionException(this, e,
            "FederationExecutionDoesNotExist ");
} catch (SaveInProgress e) {
    throw new IllegalActionException(this, e, "SaveInProgress ");
} catch (RestoreInProgress e) {
    throw new IllegalActionException(this, e, "RestoreInProgress ");
} catch (RTIinternalError e) {
    throw new IllegalActionException(this, e, "RTIinternalError ");
} catch (ConcurrentAccessAttempted e) {
    throw new IllegalActionException(this, e,
            "ConcurrentAccessAttempted ");
}

// Initialize the Federate Ambassador.
try {
    _federateAmbassador.initialize(_rtia);
} catch (NameNotFound e) {
    throw new IllegalActionException(this, e, "NameNotFound ");
} catch (ObjectClassNotDefined e) {
    throw new IllegalActionException(this, e, "ObjectClassNotDefined ");
} catch (FederateNotExecutionMember e) {
    throw new IllegalActionException(this, e,
            "FederateNotExecutionMember ");
} catch (RTIinternalError e) {
    throw new IllegalActionException(this, e, "RTIinternalError ");
} catch (AttributeNotDefined e) {
    throw new IllegalActionException(this, e, "AttributeNotDefined ");
} catch (SaveInProgress e) {
    throw new IllegalActionException(this, e, "SaveInProgress ");
} catch (RestoreInProgress e) {
    throw new IllegalActionException(this, e, "RestoreInProgress ");
```

```
} catch (ConcurrentAccessAttempted e) {
    throw new IllegalActionException(this, e,
            "ConcurrentAccessAttempted ");
}

// Initialize Federate timing values.
_federateAmbassador.initializeTimeValues(_hlaStartTime, _hlaTimeStep,
        _hlaLookAHead);

// Declare the Federate time constrained (if true).
if (_isTimeConstrained) {
    try {
        _rtia.enableTimeConstrained();
    } catch (TimeConstrainedAlreadyEnabled e) {
        throw new IllegalActionException(this, e,
                "TimeConstrainedAlreadyEnabled ");
    } catch (EnableTimeConstrainedPending e) {
        throw new IllegalActionException(this, e,
                "EnableTimeConstrainedPending ");
    } catch (TimeAdvanceAlreadyInProgress e) {
        throw new IllegalActionException(this, e,
                "TimeAdvanceAlreadyInProgress ");
    } catch (FederateNotExecutionMember e) {
        throw new IllegalActionException(this, e,
                "FederateNotExecutionMember ");
    } catch (SaveInProgress e) {
        throw new IllegalActionException(this, e, "SaveInProgress ");
    } catch (RestoreInProgress e) {
        throw new IllegalActionException(this, e, "RestoreInProgress ");
    } catch (RTIinternalError e) {
        throw new IllegalActionException(this, e, "RTIinternalError ");
    } catch (ConcurrentAccessAttempted e) {
        throw new IllegalActionException(this, e,
                "ConcurrentAccessAttempted ");
    }
}

// Declare the Federate time regulator (if true).
if (_isTimeRegulator) {
    try {
        _rtia.enableTimeRegulation(_federateAmbassador.logicalTimeHLA,
                _federateAmbassador.lookAHeadHLA);
    } catch (TimeRegulationAlreadyEnabled e) {
        throw new IllegalActionException(this, e,
                "TimeRegulationAlreadyEnabled ");
    } catch (EnableTimeRegulationPending e) {
        throw new IllegalActionException(this, e,
                "EnableTimeRegulationPending ");
    } catch (TimeAdvanceAlreadyInProgress e) {
        throw new IllegalActionException(this, e,
                "TimeAdvanceAlreadyInProgress ");
    } catch (InvalidFederationTime e) {
        throw new IllegalActionException(this, e,
                "InvalidFederationTime ");
    } catch (InvalidLookahead e) {
        throw new IllegalActionException(this, e, "InvalidLookahead ");
    } catch (FederateNotExecutionMember e) {
        throw new IllegalActionException(this, e,
                "FederateNotExecutionMember ");
    } catch (SaveInProgress e) {
        throw new IllegalActionException(this, e, "SaveInProgress ");
    } catch (RestoreInProgress e) {
        throw new IllegalActionException(this, e, "RestoreInProgress ");
    } catch (RTIinternalError e) {
        throw new IllegalActionException(this, e, "RTIinternalError ");
```

```
    } catch (ConcurrentAccessAttempted e) {
        throw new IllegalActionException(this, e,
                "ConcurrentAccessAttempted ");
    }
}

// Wait the response of the RTI towards Federate time policies that has
// been declared. The only way to get a response is to invoke the tick()
// method to receive callbacks from the RTI. We use here the tick2()
// method which is blocking and saves more CPU than the tick() method.
if (_isTimeRegulator && _isTimeConstrained) {
    while (!(_federateAmbassador.timeConstrained)) {
        try {
            _rtia.tick2();
        } catch (SpecifiedSaveLabelDoesNotExist e) {
            throw new IllegalActionException(this, e,
                    "SpecifiedSaveLabelDoesNotExist ");
        } catch (ConcurrentAccessAttempted e) {
            throw new IllegalActionException(this, e,
                    "ConcurrentAccessAttempted ");
        } catch (RTIinternalError e) {
            throw new IllegalActionException(this, e,
                    "RTIinternalError ");
        }
    }

    while (!(_federateAmbassador.timeRegulator)) {
        try {
            ((CertiRtiAmbassador) _rtia).tick2();
        } catch (SpecifiedSaveLabelDoesNotExist e) {
            throw new IllegalActionException(this, e,
                    "SpecifiedSaveLabelDoesNotExist ");
        } catch (ConcurrentAccessAttempted e) {
            throw new IllegalActionException(this, e,
                    "ConcurrentAccessAttempted ");
        } catch (RTIinternalError e) {
            throw new IllegalActionException(this, e,
                    "RTIinternalError ");
        }
    }

    if (_debugging) {
        _debug(this.getDisplayName() + " initialize() -"
                + " Time Management policies:" + " is Constrained = "
                + _federateAmbassador.timeConstrained
                + " and is Regulator = "
                + _federateAmbassador.timeRegulator);
    }

    // The following service is required to allow the reception of
    // callbacks from the RTI when a Federate used the Time management.
    try {
        _rtia.enableAsynchronousDelivery();
    } catch (AsynchronousDeliveryAlreadyEnabled e) {
        throw new IllegalActionException(this, e,
                "AsynchronousDeliveryAlreadyEnabled ");
    } catch (FederateNotExecutionMember e) {
        throw new IllegalActionException(this, e,
                "FederateNotExecutionMember ");
    } catch (SaveInProgress e) {
        throw new IllegalActionException(this, e, "SaveInProgress ");
    } catch (RestoreInProgress e) {
        throw new IllegalActionException(this, e, "RestoreInProgress ");
    } catch (RTIinternalError e) {
        throw new IllegalActionException(this, e, "RTIinternalError ");
```

```
    } catch (ConcurrentAccessAttempted e) {
        throw new IllegalActionException(this, e,
                "ConcurrentAccessAttempted ");
    }
}

if (_requireSynchronization) {
    // If the current Federate is the creator then create the
    // synchronization point.
    if (_isCreator) {
        try {
            byte[] rfspTag = EncodingHelpers
                    .encodeString(_synchronizationPointName);
            _rtia.registerFederationSynchronizationPoint(
                    _synchronizationPointName, rfspTag);
        } catch (FederateNotExecutionMember e) {
            throw new IllegalActionException(this, e,
                    "FederateNotExecutionMember ");
        } catch (SaveInProgress e) {
            throw new IllegalActionException(this, e, "SaveInProgress ");
        } catch (RestoreInProgress e) {
            throw new IllegalActionException(this, e,
                    "RestoreInProgress ");
        } catch (RTIinternalError e) {
            throw new IllegalActionException(this, e,
                    "RTIinternalError ");
        } catch (ConcurrentAccessAttempted e) {
            throw new IllegalActionException(this, e,
                    "ConcurrentAccessAttempted ");
        }

        // Wait synchronization point callbacks.
        while (!(_federateAmbassador.synchronizationSuccess)
                && !(_federateAmbassador.synchronizationFailed)) {
            try {
                ((CertiRtiAmbassador) _rtia).tick2();
            } catch (SpecifiedSaveLabelDoesNotExist e) {
                throw new IllegalActionException(this, e,
                        "SpecifiedSaveLabelDoesNotExist ");
            } catch (ConcurrentAccessAttempted e) {
                throw new IllegalActionException(this, e,
                        "ConcurrentAccessAttempted ");
            } catch (RTIinternalError e) {
                throw new IllegalActionException(this, e,
                        "RTIinternalError ");
            }
        }

        if (_federateAmbassador.synchronizationFailed) {
            throw new IllegalActionException(this,
                    "CERTI: Synchronization error ! ");
        }
    } // End block for synchronization point creation case.

    // Wait synchronization point announcement.
    while (!(_federateAmbassador.inPause)) {
        try {
            ((CertiRtiAmbassador) _rtia).tick2();
        } catch (SpecifiedSaveLabelDoesNotExist e) {
            throw new IllegalActionException(this, e,
                    "SpecifiedSaveLabelDoesNotExist ");
        } catch (ConcurrentAccessAttempted e) {
            throw new IllegalActionException(this, e,
                    "ConcurrentAccessAttempted ");
        } catch (RTIinternalError e) {
```

```
                throw new IllegalActionException(this, e,
                        "RTIinternalError ");
            }
        }

        // Satisfied synchronization point.
        try {
            _rtia.synchronizationPointAchieved(_synchronizationPointName);
            if (_debugging) {
                _debug(this.getDisplayName()
                        + " initialize() - Synchronisation point "
                        + _synchronizationPointName + " satisfied !");
            }
        } catch (SynchronizationLabelNotAnnounced e) {
            throw new IllegalActionException(this, e,
                    "SynchronizationLabelNotAnnounced ");
        } catch (FederateNotExecutionMember e) {
            throw new IllegalActionException(this, e,
                    "FederateNotExecutionMember ");
        } catch (SaveInProgress e) {
            throw new IllegalActionException(this, e, "SaveInProgress ");
        } catch (RestoreInProgress e) {
            throw new IllegalActionException(this, e, "RestoreInProgress ");
        } catch (RTIinternalError e) {
            throw new IllegalActionException(this, e, "RTIinternalError ");
        } catch (ConcurrentAccessAttempted e) {
            throw new IllegalActionException(this, e,
                    "ConcurrentAccessAttempted ");
        }

        // Wait federation synchronization.
        while (((PtolemyFederateAmbassadorInner) _federateAmbassador).inPause) {
            if (_debugging) {
                _debug(this.getDisplayName()
                        + " initialize() - Waiting for simulation phase !");
            }

            try {
                ((CertiRtiAmbassador) _rtia).tick2();
            } catch (SpecifiedSaveLabelDoesNotExist e) {
                throw new IllegalActionException(this, e,
                        "SpecifiedSaveLabelDoesNotExist ");
            } catch (ConcurrentAccessAttempted e) {
                throw new IllegalActionException(this, e,
                        "ConcurrentAccessAttempted ");
            } catch (RTIinternalError e) {
                throw new IllegalActionException(this, e,
                        "RTIinternalError ");
            }
        }
    } // End block for synchronization point.

    // GL: FIXME: need to test deeper then remove this call.
    // tick() one time to avoid missing callbacks before the start of the
    // simulation.
    try {
        ((CertiRtiAmbassador) _rtia).tick();
    } catch (ConcurrentAccessAttempted e) {
        throw new IllegalActionException(this, e,
                "ConcurrentAccessAttempted ");
    } catch (RTIinternalError e) {
        throw new IllegalActionException(this, e, "RTIinternalError ");
    }
}
```

```
/** Launch the HLA/CERTI RTIG process as subprocess. The RTIG has to be
 *  launched before the initialization of a Federate.
 *  NOTE: if another HlaManager (e.g. Federate) has already launched a RTIG,
 *  the subprocess creates here is no longer required, then we destroy it.
 *  @exception IllegalActionException If the initialization of the
 *  CertiRtig or the execution of the RTIG as subprocess has failed.
 */
public void preinitialize() throws IllegalActionException {
    super.preinitialize();

    // Try to launch the HLA/CERTI RTIG subprocess.
    _certiRtig = new CertiRtig(this, _debugging);
    _certiRtig.initialize(fedFile.asFile().getAbsolutePath());

    _certiRtig.exec();
    if (_debugging) {
        _debug(this.getDisplayName() + " preinitiliaze() - "
                + "Launch RTIG process");
    }

    if (_certiRtig.isAlreadyLaunched()) {
        _certiRtig.terminateProcess();
        _certiRtig = null;

        if (_debugging) {
            _debug(this.getDisplayName() + " preinitiliaze() - "
                    + "Destroy RTIG process as another one is already "
                    + "launched");
        }
    }
}

/** Propose a time to advance to. This method is the one implementing the
 *  TimeRegulator interface and using the HLA/CERTI Time Management services
 *  (if required). Following HLA and CERTI recommendations, if the Time
 *  Management is required then we have the following behavior:
 *  Case 1: If lookahead = 0
 *   -a) if time-stepped Federate, then the timeAdvanceRequestAvailable()
 *       (TARA) service is used;
 *   -b) if event-based Federate, then the nextEventRequestAvailable()
 *       (NERA) service is used
 *  Case 2: If lookahead > 0
 *   -c) if time-stepped Federate, then timeAdvanceRequest() (TAR) is used;
 *   -d) if event-based Federate, then the nextEventRequest() (NER) is used;
 *  Otherwise the proposedTime is returned.
 *  NOTE: For the Ptolemy II - HLA/CERTI cooperation the default (and correct)
 *  behavior is the case 1 and CERTI has to be compiled with the option
 *  "CERTI_USE_NULL_PRIME_MESSAGE_PROTOCOL".
 *  @param proposedTime The proposed time.
 *  @return The proposed time or a smaller time.
 *  @exception IllegalActionException If this attribute is not
 *  contained by an Actor.
 */
public Time proposeTime(Time proposedTime) throws IllegalActionException {
    Time breakpoint = null;

    // This test is used to avoid exception when the RTIG subprocess is
    // shutdown before the last call of this method.
    // GL: FIXME: see Ptolemy team why this is called again after STOPTIME ?
    if (_rtia == null) {
        if (_debugging) {
            _debug(this.getDisplayName() + " proposeTime() -"
                    + " called but _rtia is null");
        }
        return proposedTime;
```

```
}

// If the proposedTime has already been asked to the HLA/CERTI Federation
// then return it.

// GL: FIXME: Comment this until the clarification with NERA and TARA
// and the use of TICK is made.
/*
 if (_lastProposedTime != null) {
 if (_lastProposedTime.compareTo(proposedTime) == 0) {

 // Even if we avoid the multiple calls of the HLA Time management
 // service for optimization, it could be possible to have events
 // from the Federation in the Federate's priority timestamp queue,
 // so we tick() to get these events (if they exist).
 try {
 _rtia.tick();
 } catch (ConcurrentAccessAttempted e) {
 throw new IllegalActionException(this, e, "ConcurrentAccessAttempted ");
 } catch (RTIinternalError e) {
 throw new IllegalActionException(this, e, "RTIinternalError ");
 }

 return _lastProposedTime;
 }
 }
 */

// If the HLA Time Management is required, ask to the HLA/CERTI
// Federation (the RTI) the authorization to advance its time.
if (_isTimeRegulator && _isTimeConstrained) {
    synchronized (this) {
        // Build a representation of the proposedTime in HLA/CERTI.
        CertiLogicalTime certiProposedTime = new CertiLogicalTime(
                proposedTime.getDoubleValue());

        // Call the corresponding HLA Time Management service.
        try {

            if (_useNextEventRequest) {
                if (_debugging) {
                    _debug(this.getDisplayName()
                            + " proposeTime() - call CERTI NER -"
                            + " nextEventRequest("
                            + certiProposedTime.getTime() + ")");
                }
                _rtia.nextEventRequestAvailable(certiProposedTime);

            } else {
                if (_debugging) {
                    _debug(this.getDisplayName()
                            + " proposeTime() -  call CERTI TAR -"
                            + " timeAdvanceRequest("
                            + certiProposedTime.getTime() + ")");
                }
                _rtia.timeAdvanceRequestAvailable(certiProposedTime);
            }
        } catch (InvalidFederationTime e) {
            throw new IllegalActionException(this, e,
                    "InvalidFederationTime ");
        } catch (FederationTimeAlreadyPassed e) {
            throw new IllegalActionException(this, e,
                    "FederationTimeAlreadyPassed ");
        } catch (TimeAdvanceAlreadyInProgress e) {
            throw new IllegalActionException(this, e,
```

```
            "TimeAdvanceAlreadyInProgress ");
} catch (EnableTimeRegulationPending e) {
    throw new IllegalActionException(this, e,
            "EnableTimeRegulationPending ");
} catch (EnableTimeConstrainedPending e) {
    throw new IllegalActionException(this, e,
            "EnableTimeConstrainedPending ");
} catch (FederateNotExecutionMember e) {
    throw new IllegalActionException(this, e,
            "FederateNotExecutionMember ");
} catch (SaveInProgress e) {
    throw new IllegalActionException(this, e, "SaveInProgress ");
} catch (RestoreInProgress e) {
    throw new IllegalActionException(this, e,
            "RestoreInProgress ");
} catch (RTIinternalError e) {
    throw new IllegalActionException(this, e,
            "RTIinternalError ");
} catch (ConcurrentAccessAttempted e) {
    throw new IllegalActionException(this, e,
            "ConcurrentAccessAttempted ");
} catch (NoSuchElementException e) {
    // GL: FIXME: to investigate.
    if (_debugging) {
        _debug(this.getDisplayName() + " proposeTime() -"
                + " NoSuchElementException " + " for _rtia");
    }
    return proposedTime;
}

// Wait the grant from the HLA/CERTI Federation (from the RTI).
_federateAmbassador.timeAdvanceGrant = false;
while (!(_federateAmbassador.timeAdvanceGrant)) {
    if (_debugging) {
        _debug(this.getDisplayName() + " proposeTime() -"
                + " wait CERTI TAG - " + "timeAdvanceGrant("
                + certiProposedTime.getTime()
                + ") by calling tick2()");
    }

    try {
        _rtia.tick2();
    } catch (SpecifiedSaveLabelDoesNotExist e) {
        throw new IllegalActionException(this, e,
                "SpecifiedSaveLabelDoesNotExist ");
    } catch (ConcurrentAccessAttempted e) {
        throw new IllegalActionException(this, e,
                "ConcurrentAccessAttempted ");
    } catch (RTIinternalError e) {
        throw new IllegalActionException(this, e,
                "RTIinternalError ");
    }
}

// At this step we are sure that the HLA logical time of the
// Federate has been updated (by the reception of the TAG callback
// (timeAdvanceGrant()) and its value is the proposedTime or
// less, so we have a breakpoint time.
try {
    breakpoint = new Time(
            _director,
            ((CertiLogicalTime) _federateAmbassador.logicalTimeHLA)
            .getTime());
} catch (IllegalActionException e) {
    throw new IllegalActionException(this, e,
```

```
                        "The breakpoint time is not a valid Ptolemy time");
                }

                // Stored reflected attributes as events on HLASubscriber actors.
                _putReflectedAttributesOnHlaSubscribers();
            }

            // GL: FIXME: XXX:
            if (_useNextEventRequest) {
                if (_debugging) {
                    _debug(this.getDisplayName()
                            + " proposeTime() - call CERTI NER -"
                            + " nextEventRequest("
                            + breakpoint + ")");
                }

                try {
                    _rtia.nextEventRequest(_federateAmbassador.logicalTimeHLA);
                } catch (Exception e) {
                    throw new IllegalActionException(this, e,
                            "CERTI Exception");
                }

                _federateAmbassador.timeAdvanceGrant = false;
                while (!(_federateAmbassador.timeAdvanceGrant)) {
                    if (_debugging) {
                        _debug(this.getDisplayName()
                                + " proposeTime() - wait CERTI TAG -"
                                + " tick2() is called");
                    }

                    try {
                        _rtia.tick2();
                    } catch (Exception e) {
                        throw new IllegalActionException(this, e,
                                "CERTI Exception ");
                    }
                    _putReflectedAttributesOnHlaSubscribers();
                }
            } // end patch.

        }

        _lastProposedTime = breakpoint;
        return breakpoint;
    }

    /** Update the HLA attribute <i>attributeName</i> with the containment of
     *  the token <i>in</i>. The updated attribute is sent to the HLA/CERTI
     *  Federation.
     *  @param attributeName Name of the HLA attribute to update.
     *  @param in The updated value of the HLA attribute to update.
     *  @param asHLAPtidesEvent Indicate if it will be send to a PtidesPlatform.
     *  @throws IllegalActionException If a CERTI exception is raised then
     *  displayed it to the user.
     */
    void updateHlaAttribute(String attributeName, Token in,
            Boolean asHLAPtidesEvent) throws IllegalActionException {
        Time currentTime = _director.getModelTime();

        // The following operations build the different arguments required
        // to use the updateAttributeValues() (UAV) service provided by HLA/CERTI.

        // Retrieve information of the HLA attribute to publish.
        Object[] tObj = _hlaAttributesToPublish.get(attributeName);
```

```
        // Encode the value to be sent to the CERTI.
        byte[] valAttribute = _encodeHlaValue(in, asHLAPtidesEvent);

        SuppliedAttributes suppAttributes = null;
        try {
            suppAttributes = RtiFactoryFactory.getRtiFactory()
                    .createSuppliedAttributes();
        } catch (RTIinternalError e) {
            throw new IllegalActionException(this, e, "RTIinternalError ");
        }
        suppAttributes.add((Integer) tObj[4], valAttribute);

        byte[] tag = EncodingHelpers.encodeString(((TypedIOPort) tObj[0])
                .getContainer().getName());

        // Create a representation of the current director time for CERTI.
        Double myEpsilon = 0.000000;
        //Double myEpsilon = 0.0000000;

        CertiLogicalTime ct = new CertiLogicalTime(currentTime.getDoubleValue()
                + myEpsilon);
        try {
            _rtia.updateAttributeValues((Integer) tObj[5], suppAttributes, tag,
                    ct);
        } catch (ObjectNotKnown e) {
            throw new IllegalActionException(this, e, "ObjectNotKnown ");
        } catch (AttributeNotDefined e) {
            throw new IllegalActionException(this, e, "AttributeNotDefined ");
        } catch (AttributeNotOwned e) {
            throw new IllegalActionException(this, e, "AttributeNotOwned ");
        } catch (FederateNotExecutionMember e) {
            throw new IllegalActionException(this, e,
                    "FederateNotExecutionMember ");
        } catch (SaveInProgress e) {
            throw new IllegalActionException(this, e, "SaveInProgress ");
        } catch (RestoreInProgress e) {
            throw new IllegalActionException(this, e, "RestoreInProgress ");
        } catch (RTIinternalError e) {
            throw new IllegalActionException(this, e, "RTIinternalError ");
        } catch (ConcurrentAccessAttempted e) {
            throw new IllegalActionException(this, e,
                    "ConcurrentAccessAttempted ");
        } catch (InvalidFederationTime e) {
            throw new IllegalActionException(this, e, "InvalidFederationTime ");
        }
        if (_debugging) {
            _debug(this.getDisplayName() + " publish() -"
                    + " send (UAV) updateAttributeValues "
                    + " current Ptolemy Time=" + currentTime.getDoubleValue()
                    + " HLA attribute \""
                    + ((TypedIOPort) tObj[0]).getContainer().getName()
                    + "\" (timestamp=" + ct.getTime() + ", value="
                    + in.toString() + ")");
        }
    }

    /** Manage the correct termination of the {@link HlaManager}. Call the
     *  HLA services to: unsubscribe to HLA attributes, unpublish HLA attributes,
     *  resign a Federation and destroy a Federation if the current Federate is
     *  the last participant.
     *  @throws IllegalActionException If the parent class throws it
     *  of if a CERTI exception is raised then displayed it to the user.
     */
    public void wrapup() throws IllegalActionException {
```

```
super.wrapup();

if (_debugging) {
    _debug(this.getDisplayName() + " wrapup() - ... so termination");
}

// Unsubscribe to HLA attributes
for (Object[] obj : _hlaAttributesSubscribedTo.values()) {
    try {
        _rtia.unsubscribeObjectClass((Integer) obj[3]);
    } catch (ObjectClassNotDefined e) {
        throw new IllegalActionException(this, e,
                "ObjectClassNotDefined ");
    } catch (ObjectClassNotSubscribed e) {
        throw new IllegalActionException(this, e,
                "ObjectClassNotSubscribed ");
    } catch (FederateNotExecutionMember e) {
        throw new IllegalActionException(this, e,
                "FederateNotExecutionMember ");
    } catch (SaveInProgress e) {
        throw new IllegalActionException(this, e, "SaveInProgress ");
    } catch (RestoreInProgress e) {
        throw new IllegalActionException(this, e, "RestoreInProgress ");
    } catch (RTIinternalError e) {
        throw new IllegalActionException(this, e, "RTIinternalError ");
    } catch (ConcurrentAccessAttempted e) {
        throw new IllegalActionException(this, e,
                "ConcurrentAccessAttempted ");
    }
    if (_debugging) {
        _debug(this.getDisplayName() + " wrapup() - Unsubscribe "
                + ((TypedIOPort) obj[0]).getContainer().getName()
                + "(classHandle = " + obj[3] + ")");
    }
}

// Unpublish HLA attributes.
for (Object[] obj : _hlaAttributesToPublish.values()) {
    try {
        _rtia.unpublishObjectClass((Integer) obj[3]);
    } catch (ObjectClassNotDefined e) {
        throw new IllegalActionException(this, e,
                "ObjectClassNotDefined ");
    } catch (ObjectClassNotPublished e) {
        throw new IllegalActionException(this, e,
                "ObjectClassNotPublished ");
    } catch (OwnershipAcquisitionPending e) {
        throw new IllegalActionException(this, e,
                "OwnershipAcquisitionPending ");
    } catch (FederateNotExecutionMember e) {
        throw new IllegalActionException(this, e,
                "FederateNotExecutionMember ");
    } catch (SaveInProgress e) {
        throw new IllegalActionException(this, e, "SaveInProgress ");
    } catch (RestoreInProgress e) {
        throw new IllegalActionException(this, e, "RestoreInProgress ");
    } catch (RTIinternalError e) {
        throw new IllegalActionException(this, e, "RTIinternalError ");
    } catch (ConcurrentAccessAttempted e) {
        throw new IllegalActionException(this, e,
                "ConcurrentAccessAttempted ");
    }
    if (_debugging) {
        _debug(this.getDisplayName() + " wrapup() - Unpublish "
                + ((TypedIOPort) obj[0]).getContainer().getName()
```

```
                + "(classHandle = " + obj[3] + ")");
    }
}

// Resign HLA/CERTI Federation execution.
try {
    _rtia.resignFederationExecution(ResignAction.
        DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES);
} catch (FederateOwnsAttributes e) {
    throw new IllegalActionException(this, e, "FederateOwnsAttributes ");
} catch (FederateNotExecutionMember e) {
    throw new IllegalActionException(this, e,
            "FederateNotExecutionMember ");
} catch (InvalidResignAction e) {
    throw new IllegalActionException(this, e, "InvalidResignAction ");
} catch (RTIinternalError e) {
    throw new IllegalActionException(this, e, "RTIinternalError ");
} catch (ConcurrentAccessAttempted e) {
    throw new IllegalActionException(this, e,
            "ConcurrentAccessAttempted ");
}
if (_debugging) {
    _debug(this.getDisplayName()
            + " wrapup() - Resign Federation execution");
}

boolean canDestroyRtig = false;
while (!canDestroyRtig) {

    // Destroy federation execution - nofail.
    try {
        _rtia.destroyFederationExecution(_federationName);
    } catch (FederatesCurrentlyJoined e) {
        if (_debugging) {
            _debug(this.getDisplayName()
                    + " wrapup() - WARNING: FederatesCurrentlyJoined");
        }
    } catch (FederationExecutionDoesNotExist e) {
        // GL: FIXME: This should be an IllegalActionExeception
        if (_debugging) {
            _debug(this.getDisplayName()
                    + " wrapup() - WARNING: FederationExecutionDoesNotExist");
        }
        canDestroyRtig = true;
    } catch (RTIinternalError e) {
        throw new IllegalActionException(this, e, "RTIinternalError ");
    } catch (ConcurrentAccessAttempted e) {
        throw new IllegalActionException(this, e,
                "ConcurrentAccessAttempted ");
    }
    if (_debugging) {
        _debug(this.getDisplayName() + " wrapup() - "
                + "Destroy Federation execution - no fail");
    }

    canDestroyRtig = true;
}

// Terminate RTIG subprocess.
if (_certiRtig != null) {
    _certiRtig.terminateProcess();

    if (_debugging) {
        _debug(this.getDisplayName() + " wrapup() - "
                + "Destroy RTIG process");
```

```
        }
    }

    // Clean HLA attribute tables.
    _hlaAttributesToPublish.clear();
    _hlaAttributesSubscribedTo.clear();
    _fromFederationEvents.clear();
    _objectIdToClassHandle.clear();
}

///////////////////////////////////////////////////////////////////
////                         protected variables               ////

/** Table of HLA attributes (and their HLA information) that are published
 *  by the current {@link HlaManager} to the HLA/CERTI Federation. This
 *  table is indexed by the {@link HlaPublisher} actors present in the model.
 */
protected HashMap<String, Object[]> _hlaAttributesToPublish;

/** Table of HLA attributes (and their HLA information) that the current
 *  {@link HlaManager} is subscribed to. This table is indexed by the
 *  {@link HlaSubscriber} actors present in the model.
 */
protected HashMap<String, Object[]> _hlaAttributesSubscribedTo;

/** List of events received from the HLA/CERTI Federation and indexed by the
 *  {@link HlaSubscriber} actors present in the model.
 */
protected HashMap<String, LinkedList<TimedEvent>> _fromFederationEvents;

/** Table of object class handles associate to object ids received by
 *  discoverObjectInstance and reflectAttributesValues services (e.g. from
 *  the RTI).
 */
protected HashMap<Integer, Integer> _objectIdToClassHandle;

///////////////////////////////////////////////////////////////////
////                         private methods                   ////

/** This generic method should call the {@link EncodingHelpers} API provided
 *  by CERTI to handle type decoding operations for HLA value attribute that
 *  has been reflected.
 *  @param type The type to decode the token.
 *  @param buffer The encoded value to decode.
 *  @return The decoded value as an object.
 *  @throws IllegalActionException If the token is not handled or the
 *  decoding has failed.
 */
private Object _decodeHlaValue(Type type, byte[] buffer)
        throws IllegalActionException {

    // GL: FIXME: PTIDES
    if (type instanceof RecordType) {
        Double ret = EncodingHelpers.decodeDouble(buffer);

        return ret;
    } else if (type.equals(BaseType.BOOLEAN)) {
        return EncodingHelpers.decodeBoolean(buffer);
    } else if (type.equals(BaseType.UNSIGNED_BYTE)) {
        return EncodingHelpers.decodeByte(buffer);
    } else if (type.equals(BaseType.DOUBLE)) {
        // GL: FIXME: XXX: This code only work with
        // SDSE/PRISE joystick federate.

        boolean useMessageBuffer = false;
```

```
        if (useMessageBuffer) {
            ByteArrayInputStream bis = new ByteArrayInputStream(buffer);
            ByteArrayOutputStream bos = new ByteArrayOutputStream();

            MessageBuffer mb = new MessageBuffer(bis, bos);
            try {
                mb.receiveData();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (CertiException e) {
                e.printStackTrace();
            }
            double ret = mb.readDouble();
            return ret;
        } else {
            return EncodingHelpers.decodeDouble(buffer);
        }
    } else if (type.equals(BaseType.FLOAT)) {
        return EncodingHelpers.decodeFloat(buffer);
    } else if (type.equals(BaseType.INT)) {
        return EncodingHelpers.decodeInt(buffer);
    } else if (type.equals(BaseType.LONG)) {
        return EncodingHelpers.decodeLong(buffer);
    } else if (type.equals(BaseType.SHORT)) {
        return EncodingHelpers.decodeShort(buffer);
    } else if (type.equals(BaseType.STRING)) {
        return EncodingHelpers.decodeString(buffer);
    } else {
        throw new IllegalActionException(this,
                "The current type received by the HLA/CERTI Federation"
                        + " is not handled by " + this.getDisplayName());
    }
}

/** This generic method should call the {@link EncodingHelpers} API provided
 *  by CERTI to handle type encoding operation for HLA value attribute that
 *  will be published.
 *  @param tok The token to encode.
 *  @return The encoded value as an array of byte.
 *  @throws IllegalActionException If the token is not handled or the
 *  encoding had failed.
 */
private byte[] _encodeHlaValue(Token tok, Boolean asHLAPtidesEvent)
        throws IllegalActionException {
    byte[] encodedValue = null;
    Token t = null;
    double recordTimestamp = -1;
    int recordMicrostep = -1;
    double sourceTimestamp = -1;

    // GL: FIXME: PTIDES
    // This first case handle events from a PtidesPlatform to PtidesPlatform,
    // only network port are supported.
    if (asHLAPtidesEvent) {
        RecordToken rt = (RecordToken) tok;

        recordTimestamp = ((DoubleToken) rt.get("timestamp")).doubleValue();
        recordMicrostep = ((IntToken) rt.get("microstep")).intValue();
        sourceTimestamp = ((DoubleToken) rt.get("sourceTimestamp"))
                .doubleValue();
        t = rt.get("payload");
    } else {
        t = tok;
    }
```

```
BaseType type = (BaseType) t.getType();

if (type.equals(BaseType.BOOLEAN)) {
    encodedValue = EncodingHelpers.encodeBoolean(((BooleanToken) t)
            .booleanValue());
} else if (type.equals(BaseType.UNSIGNED_BYTE)) { // TODO Auto-generated catch
    block

    encodedValue = EncodingHelpers.encodeByte(((UnsignedByteToken) t)
            .byteValue());
} else if (type.equals(BaseType.DOUBLE)) {
    // GL: FIXME: XXX: This code only work with
    // SDSE/PRISE joystick federate.

    ((BooleanToken) isCreator.getToken()).booleanValue();

    boolean useMessageBuffer = false;
    if (useMessageBuffer) {
        ByteArrayOutputStream bos = new ByteArrayOutputStream();

        MessageBuffer mb = new MessageBuffer(null, bos);

        mb.write(((DoubleToken) t).doubleValue());
        try {
            mb.send();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        encodedValue = bos.toByteArray();
    } else {
        encodedValue = EncodingHelpers.encodeDouble(((DoubleToken) t)
                .doubleValue());
    }
} else if (type.equals(BaseType.FLOAT)) {
    encodedValue = EncodingHelpers.encodeFloat(((FloatToken) t)
            .floatValue());
} else if (type.equals(BaseType.INT)) {
    encodedValue = EncodingHelpers.encodeInt(((IntToken) t).intValue());
} else if (type.equals(BaseType.LONG)) {
    encodedValue = EncodingHelpers.encodeLong(((LongToken) t)
            .longValue());
} else if (type.equals(BaseType.SHORT)) {
    encodedValue = EncodingHelpers.encodeShort(((ShortToken) t)
            .shortValue());
} else if (type.equals(BaseType.STRING)) {
    encodedValue = EncodingHelpers.encodeString(((StringToken) t)
            .stringValue());
} else {
    throw new IllegalActionException(this,
            "The current type of the token " + t
            + " is not handled by " + this.getDisplayName());
}

// GL: FIXME: PTIDES
if (asHLAPtidesEvent) {
    Time currentTime = _director.getModelTime();

    if (_debugging) {
        _debug(this.getDisplayName()
                + " _encodeHlaValue() - Encoded as HlaPtidesEvent"
                + " currentTime=" + currentTime.getDoubleValue()
                + " recordTimestamp=" + recordTimestamp);
    }

    HlaPtidesEvent he = new HlaPtidesEvent(recordTimestamp,
```

```
                recordMicrostep, sourceTimestamp, encodedValue);

        return he.getBuffer();
    } else {
        return encodedValue;
    }
}

/** The method {@link _populatedHlaValueTables()} populates the tables
 *  containing information of HLA attributes required to publish and to
 *  subscribe value attributes in a HLA Federation.
 *  @throws IllegalActionException If a HLA attribute is declared twice.
 */
private void _populateHlaAttributeTables() throws IllegalActionException {
    CompositeActor ca = (CompositeActor) this.getContainer();

    List<HlaSubscriber> _hlaSubscribers = null;
    List<HlaPublisher> _hlaPublishers = null;
    _hlaAttributesToPublish.clear();
    _hlaAttributesSubscribedTo.clear();

    _hlaPublishers = ca.entityList(HlaPublisher.class);
    for (HlaPublisher hp : _hlaPublishers) {
        if (_hlaAttributesToPublish.get(hp.getName()) != null) {
            throw new IllegalActionException(this,
                    "A HLA value with the same name is already registered"
                            + " for publication");
        }
        // Only one input port is allowed per HlaPublisher actor.
        TypedIOPort tiop = hp.inputPortList().get(0);

        _hlaAttributesToPublish.put(
                hp.getName(),
                new Object[] {
                    tiop,
                    tiop.getType(),
                    ((StringToken) ((Parameter) hp
                            .getAttribute("classObjectHandle"))
                            .getToken()).stringValue() });
    }

    _hlaSubscribers = ca.entityList(HlaSubscriber.class);
    for (HlaSubscriber hs : _hlaSubscribers) {
        if (_hlaAttributesSubscribedTo.get(hs.getName()) != null) {
            throw new IllegalActionException(this,
                    "A HLA value with the same name is already registered"
                            + " for subscription");
        }
        // Only one output port is allowed per HlaSubscriber actor.
        TypedIOPort tiop = hs.outputPortList().get(0);

        _hlaAttributesSubscribedTo.put(
                hs.getName(),
                new Object[] {
                    tiop,
                    tiop.getType(),
                    ((StringToken) ((Parameter) hs
                            .getAttribute("classObjectHandle"))
                            .getToken()).stringValue() });

        // The events list to store HLA updated values (received by callbacks
        // from the RTI is indexed by the HLA Subscriber actors present in
        // the model.
        _fromFederationEvents.put(hs.getName(),
                new LinkedList<TimedEvent>());
```

```
    }
}

/** This method is called when a time advancement phase is performed. Every
 *  updated HLA attributes received by callbacks (from the RTI) during the
 *  time advancement phase is saved as {@link TimedEvent} and stored in a
 *  queue. Then, every {@link TimedEvent}s are moved from this queue to the
 *  output port of their corresponding {@link HLASubscriber} actors
 *  @throws IllegalActionException If the parent class throws it.
 */
private void _putReflectedAttributesOnHlaSubscribers()
        throws IllegalActionException {
    // Reflected HLA attributes, e.g. update of HLA attributes received by
    // callbacks (from the RTI) from the whole HLA/CERTI Federation are store
    // in the _subscribedValues queue (see reflectAttributeValues() in
    // PtolemyFederateAmbassadorInner class).

    TimedEvent event;
    for (int i = 0; i < _hlaAttributesSubscribedTo.size(); i++) {
        Iterator<Entry<String, LinkedList<TimedEvent>>> it = _fromFederationEvents
                .entrySet().iterator();

        while (it.hasNext()) {
            Map.Entry<String, LinkedList<TimedEvent>> elt = (Map.Entry) it
                    .next();

            // GL: FIXME: Check if multiple events here with same timestamp
            // make sense and can occur, if true we need to update the
            // following code to handle this case.
            if (elt.getValue().size() > 0) {
                event = elt.getValue().getFirst();

                TypedIOPort tiop = (TypedIOPort) ((Object[])
                    _hlaAttributesSubscribedTo
                        .get(elt.getKey()))[0];
                HlaSubscriber hs = (HlaSubscriber) tiop.getContainer();

                hs.putReflectedHlaAttribute(event);
                if (_debugging) {
                    _debug(this.getDisplayName()
                            + " _putReflectedAttributesOnHlaSubscribers() - "
                            + " put Event: " + event.toString() + " in "
                            + hs.getDisplayName());
                }

                elt.getValue().remove(event);
            }
        }
    }
}

///////////////////////////////////////////////////////////////////
////                         private variables                 ////

/** Name of the current Ptolemy federate ({@link HlaManager}).*/
private String _federateName;

/**-Name of the HLA/CERTI federation to create or to join. */
private String _federationName;

/** RTI Ambassador for the Ptolemy Federate. */
private CertiRtiAmbassador _rtia;

/** Federate Ambassador for the Ptolemy Federate. */
private PtolemyFederateAmbassadorInner _federateAmbassador;
```

```
/** Indicates the use of the nextEventRequest() service. */
private Boolean _useNextEventRequest;

/** Indicates the use of the timeAdvanceRequest() service. */
private Boolean _useTimeAdvancedRequest;

/** Indicates the use of the enableTimeConstrained() service. */
private Boolean _isTimeConstrained;

/** Indicates the use of the enableTimeRegulation() service. */
private Boolean _isTimeRegulator;

/** Start time of the Ptolemy Federate HLA logical clock. */
private Double _hlaStartTime;

/** Time step of the Ptolemy Federate. */
private Double _hlaTimeStep;

/** The lookahead value of the Ptolemy Federate. */
private Double _hlaLookAHead;

/** Indicates if the Ptolemy Federate will use a synchronization point. */
private Boolean _requireSynchronization;

/** Name of the synchronization point to create or to reach. */
private String _synchronizationPointName;

/** Indicates if the Ptolemy Federate is the creator of the synchronization
 *  point.
 */
private Boolean _isCreator;

/** Records the last proposed time to avoid multiple HLA time advancement
 *  requests at the same time.
 */
private Time _lastProposedTime;

/** A reference to the enclosing director. */
private DEDirector _director;

/** The RTIG subprocess. */
private CertiRtig _certiRtig;

///////////////////////////////////////////////////////////////////
////                         inner class                       ////

/** This class extends the {@link NullFederateAmbassador} class which
 *  implements the basics HLA services provided by the JCERTI bindings.
 *  @author Gilles Lasnier
 */
private class PtolemyFederateAmbassadorInner extends NullFederateAmbassador {

    ///////////////////////////////////////////////////////////////////
    ////                         public variables                  ////

    /** Indicates if the Federate is declared as time regulator in the
     *  HLA/CERTI Federation. This value is set by callback by the RTI.
     */
    public Boolean timeRegulator;

    /** Indicates if the Federate is declared as time constrained in the
     *  HLA/CERTI Federation. This value is set by callback by the RTI.
     */
    public Boolean timeConstrained;
```

```
/** Indicates if the Federate has received the time advance grant from
 *  the HLA/CERTI Federation. This value is set by callback by the RTI.
 */
public Boolean timeAdvanceGrant;

/** Indicates the current HLA logical time of the Federate. This value
 *  is set by callback by the RTI.
 */
public LogicalTime logicalTimeHLA;

/** Federate time step. */
public LogicalTime timeStepHLA;

/** Indicates if the request of synchronization by the Federate is
 *  validated by the HLA/CERTI Federation. This value is set by callback
 *  by the RTI.
 */
public Boolean synchronizationSuccess;

/** Indicates if the request of synchronization by the Federate
 *  has failed. This value is set by callback by the RTI.
 */
public Boolean synchronizationFailed;

/** Indicates if the Federate is currently synchronize to others. This
 * value is set by callback by the RTI.
 */
public Boolean inPause;

/** The lookahead value set by the user and used by CERTI to handle
 *  time management and to order TSO events.
 */
public LogicalTimeInterval lookAHeadHLA;

////////////////////////////////////////////////////////////////
////                         public methods                 ////

/** Initialize the {@link PtolemyFederateAmbassador} which handles
 *  the communication from RTI -> to RTIA -> to FEDERATE. The
 *  <i>rtia</i> manages the interaction with the external communicant
 *  process RTIA. This method called the Declaration Management
 *  services provide by HLA/CERTI to publish/subscribe to HLA attributes
 *  in a HLA Federation.
 *  @param rtia
 *  @throws NameNotFound
 *  @throws ObjectClassNotDefined
 *  @throws FederateNotExecutionMember
 *  @throws RTIinternalError
 *  @throws AttributeNotDefined
 *  @throws SaveInProgress
 *  @throws RestoreInProgress
 *  @throws ConcurrentAccessAttempted
 *  All those exceptions are from the HLA/CERTI implementation.
 */
public void initialize(RTIambassador rtia) throws NameNotFound,
ObjectClassNotDefined, FederateNotExecutionMember,
RTIinternalError, AttributeNotDefined, SaveInProgress,
RestoreInProgress, ConcurrentAccessAttempted {
    this.timeAdvanceGrant = false;
    this.timeConstrained = false;
    this.timeRegulator = false;
    this.synchronizationSuccess = false;
    this.synchronizationFailed = false;
    this.inPause = false;
```

```
// GL: FIXME: XXX: This method needs to be review and simplified.

// For each HlaPublisher actors deployed in the model we declare
// to the HLA/CERTI Federation a HLA attribute to publish.

// 1. Set classHandle and objAttributeHandle ids for each attribute
// value to publish (i.e. HlaPublisher). Update the HlaPublishers
// table with the information.
Iterator<Entry<String, Object[]>> it = _hlaAttributesToPublish
        .entrySet().iterator();

while (it.hasNext()) {
    Map.Entry<String, Object[]> elt = (Map.Entry) it.next();
    Object[] tObj = (Object[]) elt.getValue();

    // Object class handle and object attribute handle are ids that
    // allow to identify an HLA attribute.
    int classHandle = rtia.getObjectClassHandle((String) tObj[2]);
    int objAttributeHandle = rtia.getAttributeHandle(
            ((TypedIOPort) tObj[0]).getContainer().getName(),
            classHandle);

    // Update HLA attribute information (for publication)
    // from HLA services. In this case, the tObj[] object as
    // the following structure:
    // tObj[0] => input port which receives the token to transform
    //            as an updated value of a HLA attribute,
    // tObj[1] => type of the port (e.g. of the attribute),
    // tObj[2] => object class name of the attribute,
    // tObj[3] => id of the object class to handle,
    // tObj[4] => id of the attribute to handle,
    // tObj[5] => id of the registered attribute's instance.

    // tObj[0 .. 2] are extracted from the Ptolemy model.
    // tObj[3 .. 5] are provided by the RTI (CERTI).

    // All these information are required to publish/unpublish
    // updated value of a HLA attribute.
    elt.setValue(new Object[] { tObj[0], tObj[1], tObj[2],
            classHandle, objAttributeHandle, null });
}

// 2. Create a table of HlaPublishers indexed by their corresponding
// classHandle (no duplication).
HashMap<String, LinkedList<String>> classHandleHlaPublisherTable = null;
classHandleHlaPublisherTable = new HashMap<String, LinkedList<String>>();

Iterator<Entry<String, Object[]>> it2 = _hlaAttributesToPublish
        .entrySet().iterator();

while (it2.hasNext()) {
    Map.Entry<String, Object[]> elt = (Map.Entry) it2.next();
    Object[] tObj = (Object[]) elt.getValue();

    // The classHandle where the HLA attribute belongs to (see FOM).
    String classHandleName = (String) tObj[2];

    if (classHandleHlaPublisherTable.containsKey(classHandleName)) {
        classHandleHlaPublisherTable.get(classHandleName).add(
                elt.getKey());
    } else {
        LinkedList<String> list = new LinkedList<String>();
        list.add(elt.getKey());
        classHandleHlaPublisherTable.put(classHandleName, list);
```

```
    }
}

// 3. Declare to the Federation the HLA attributes to publish. If
// these attributes belongs to the same object class then only
// one registerObjectInstance call is performed.
// Then, update the HlaPublishers table with the new information.
Iterator<Entry<String, LinkedList<String>>> it3 = classHandleHlaPublisherTable
        .entrySet().iterator();

while (it3.hasNext()) {
    Map.Entry<String, LinkedList<String>> elt = (Map.Entry) it3
            .next();
    LinkedList<String> hlaPublishers = elt.getValue();

    int classHandle = rtia.getObjectClassHandle(elt.getKey());

    // The attribute handle set to declare all attributes to publish
    // for one object class.
    AttributeHandleSet _attributesLocal = RtiFactoryFactory
            .getRtiFactory().createAttributeHandleSet();

    // Fill the attribute handle set with all attibute to publish.
    for (String s : hlaPublishers) {
        _attributesLocal.add((Integer) _hlaAttributesToPublish
                .get(s)[4]);
    }

    // Declare to the Federation the HLA attribute(s) to publish.
    try {
        rtia.publishObjectClass(classHandle, _attributesLocal);
    } catch (OwnershipAcquisitionPending e) {
        e.printStackTrace();
    }

    int myObjectInstId = -1;

    // ROI is called ** only once time ** for one federate. This
    // means that only one instance of object class are allowed.
    try {
        myObjectInstId = rtia.registerObjectInstance(classHandle,
                elt.getKey() + "%%" + _federateName);
    } catch (ObjectClassNotPublished e) {
        e.printStackTrace();
    } catch (ObjectAlreadyRegistered e) {
        e.printStackTrace();
    }

    for (String s : hlaPublishers) {
        Object[] tObj = (Object[]) _hlaAttributesToPublish.get(s);

        _hlaAttributesToPublish.put(s,
                new Object[] { tObj[0], tObj[1], tObj[2], tObj[3],
                tObj[4], myObjectInstId });
    }
}

// For each HlaSubscriber actors deployed in the model we declare
// to the HLA/CERTI Federation a HLA attribute to subscribe to.
Iterator<Entry<String, Object[]>> ot = _hlaAttributesSubscribedTo
        .entrySet().iterator();

while (ot.hasNext()) {
    Map.Entry<String, Object[]> elt = (Map.Entry) ot.next();
    Object[] tObj = (Object[]) elt.getValue();
```

```
    int classHandle = rtia.getObjectClassHandle((String) tObj[2]);
    int objAttributeHandle = rtia.getAttributeHandle(
            ((TypedIOPort) tObj[0]).getContainer().getName(),
            classHandle);

    // Update HLA attribute information (for subscription)
    // from HLA services. In this case, the tObj[] object as
    // the following structure:
    // tObj[0] => output port which will produce the event received
    //            as updated value of a HLA attribute,
    // tObj[1] => type of the port (e.g. of the attribute),
    // tObj[2] => object class name of the attribute,
    // tObj[3] => id of the object class handle,
    // tObj[4] => id of the attribute handle.

    // tObj[0 .. 2] are extracted from the Ptolemy model.
    // tObj[3 .. 4] are provided by the RTI (CERTI).

    // All these information are required to subscribe to/unsubscribe
    // to a HLA attribute.
    elt.setValue(new Object[] { tObj[0], tObj[1], tObj[2],
            classHandle, objAttributeHandle });
}

// 2. Create a table of HlaSubscribers indexed by their corresponding
//    object class handle (no duplication).
HashMap<String, LinkedList<String>> classHandleHlaSubscriberTable = null;
classHandleHlaSubscriberTable = new HashMap<String, LinkedList<String>>();

Iterator<Entry<String, Object[]>> it4 = _hlaAttributesSubscribedTo
        .entrySet().iterator();

while (it4.hasNext()) {
    Map.Entry<String, Object[]> elt = (Map.Entry) it4.next();
    Object[] tObj = (Object[]) elt.getValue();

    // The object class name of the HLA attribute (see FOM).
    String classHandleName = (String) tObj[2];

    if (classHandleHlaSubscriberTable.containsKey(classHandleName)) {
        classHandleHlaSubscriberTable.get(classHandleName).add(
                elt.getKey());
    } else {
        LinkedList<String> list = new LinkedList<String>();
        list.add(elt.getKey());
        classHandleHlaSubscriberTable.put(classHandleName, list);
    }
}

// 3. Declare to the Federation the HLA attributes to subscribe to.
Iterator<Entry<String, LinkedList<String>>> it5 =
    classHandleHlaSubscriberTable
        .entrySet().iterator();

while (it5.hasNext()) {
    Map.Entry<String, LinkedList<String>> elt = (Map.Entry) it5
            .next();
    LinkedList<String> hlaSubList = elt.getValue();

    int classHandle = rtia.getObjectClassHandle(elt.getKey());

    // The attribute handle set to declare all subscribed attributes
    // for one object class.
    AttributeHandleSet _attributesLocal = RtiFactoryFactory
```

```
                .getRtiFactory().createAttributeHandleSet();

        for (String s : hlaSubList) {
            _attributesLocal.add((Integer) _hlaAttributesSubscribedTo
                    .get(s)[4]);
        }

        _rtia.subscribeObjectClassAttributes(classHandle,
                _attributesLocal);
    }
}

/** Initialize Federate's timing properties provided by the user.
 *  @param startTime The start time of the Federate logical clock.
 *  @param timeStep The time step of the Federate.
 *  @param lookAHead The contract value used by HLA/CERTI to synchronize
 *  the Federates and to order TSO events.
 */
public void initializeTimeValues(Double startTime, Double timeStep,
        Double lookAHead) {
    logicalTimeHLA = new CertiLogicalTime(startTime);
    lookAHeadHLA = new CertiLogicalTimeInterval(lookAHead);
    timeStepHLA = new CertiLogicalTime(timeStep);

    timeAdvanceGrant = false;
}

// HLA Object Management services (callbacks).

/** Callback to receive updated value of a HLA attribute from the
 *  whole Federation (delivered by the RTI (CERTI)).
 */
public void reflectAttributeValues(int theObject,
        ReflectedAttributes theAttributes, byte[] userSuppliedTag,
        LogicalTime theTime, EventRetractionHandle retractionHandle)
                throws ObjectNotKnown, AttributeNotKnown,
                FederateOwnsAttributes, InvalidFederationTime,
                FederateInternalError {
    try {
        for (int i = 0; i < theAttributes.size(); i++) {
            Iterator<Entry<String, Object[]>> ot = _hlaAttributesSubscribedTo
                    .entrySet().iterator();

            while (ot.hasNext()) {
                Map.Entry<String, Object[]> elt = (Map.Entry) ot.next();
                Object[] tObj = (Object[]) elt.getValue();

                Time ts = null;
                TimedEvent te = null;

                // Get the object class handle corresponding to
                // received "theObject" id.
                int classHandle = _objectIdToClassHandle.get(theObject);

                // The tuple (attributeHandle, classHandle) allows to
                // identify the the object attribute (i.e. the HlaSubscriber)
                // where the updated value has to be stored.
                if (theAttributes.getAttributeHandle(i) == (Integer) tObj[4]
                        && (Integer) tObj[3] == classHandle) {
                    try {
                        // GL: FIXME: PTIDES
                        // This first case handle events from a PtidesPlatform
                        // to PtidesPlatform, only network port are supported.
                        if (tObj[1] instanceof RecordType) {
                            HlaPtidesEvent he = new HlaPtidesEvent(
```

```
                                theAttributes.getValue(i));

                        // GL: FIXME: PTIDES: should be:
                        //ts = new Time(_director, he.getSourceTime());
                        ts = new Time(_director,
                                ((CertiLogicalTime) theTime)
                                .getTime());
                        te = new TimedEvent(ts, new Object[] {
                                (RecordType) tObj[1],
                                _decodeHlaValue(
                                        (RecordType) tObj[1],
                                        he.getValue()),
                                        // GL: FIXME: PTIDES: should be:
                                        // he.getLogicalTime(),
                                        ts.getDoubleValue(),
                                        he.getMicroStep(),
                                        // GL: FIXME: PTIDES: should be:
                                        he.getSourceTime() });
                        //ts.getDoubleValue()});

                        //  System.out.println("RAV " + "has to decode
                            asHlaPtidesEvent");
                        //  System.out.println("RAV " + "logicalTime="+he.
                            getLogicalTime());
                        //  System.out.println("RAV " + "srcTimeStamp="+he.
                            getSourceTime());
                        //  System.out.println("RAV " + "time of RAV=" + ts.
                            getDoubleValue());

                    } else {
                        ts = new Time(_director,
                                ((CertiLogicalTime) theTime)
                                .getTime());
                        te = new TimedEvent(
                                ts,
                                new Object[] {
                                        (BaseType) tObj[1],
                                        _decodeHlaValue(
                                                (BaseType) tObj[1],
                                                theAttributes
                                                .getValue(i)) });
                    }
                } catch (IllegalActionException e) {
                    e.printStackTrace();
                }

                _fromFederationEvents.get(
                        ((TypedIOPort) tObj[0]).getContainer()
                        .getName()).add(te);
                if (_debugging) {
                    _debug(HlaManager.this.getDisplayName()
                            + " INNER"
                            + " reflectAttributeValues() (RAV) - "
                            + "HLA attribute: "
                            + ((TypedIOPort) tObj[0])
                            .getContainer().getName()
                            + "(value=" + te.contents
                            + ", timestamp=" + te.timeStamp
                            + ") has been received");
                }
            }
        }
    }
} catch (ArrayIndexOutOfBounds e) {
    e.printStackTrace();
```

```
    }
}

/** Callback delivered by the RTI (CERTI) to discover attribute instance
 *  of HLA attribute that the Federate is subscribed to.
 */
public void discoverObjectInstance(int theObject, int theObjectClass,
        String objectName) throws CouldNotDiscover,
        ObjectClassNotKnown, FederateInternalError {
    if (_debugging) {
        _debug(HlaManager.this.getDisplayName() + " INNER"
                + " discoverObjectInstance() - the object class "
                + objectName + " has been discovered" + " (theObject="
                + theObject + ", theObjectClass=" + theObjectClass
                + ")");
    }

    _objectIdToClassHandle.put(theObject, theObjectClass);

    // GL: FIXME: XXX: check with PS if this code could be remove also
    // when using HLA v1516
    /*
    try {
      _attributes = null;
      if (_hlaAttributesSubscribedTo.get(objectName) != null) {
        _rtia.requestObjectAttributeValueUpdate(theObject, _attributes);

        if (_debugging) {
          _debug(HlaManager.this.getDisplayName() + " INNER"
              + " discoverObjectInstance() - " + objectName
              + " has been discovered"
              + " (theObject=" + theObject
              + " , theObjectClass=" + theObjectClass + ")");
        }
      }
    } catch (ObjectNotKnown e) {
      e.printStackTrace();
    } catch (AttributeNotDefined e) {
      e.printStackTrace();
    } catch (FederateNotExecutionMember e) {
      e.printStackTrace();
    } catch (SaveInProgress e) {
      e.printStackTrace();
    } catch (RestoreInProgress e) {
      e.printStackTrace();
    } catch (RTIinternalError e) {
      e.printStackTrace();
    } catch (ConcurrentAccessAttempted e) {
      e.printStackTrace();
    } */
}

// HLA Time Management services (callbacks).

/** Callback delivered by the RTI (CERTI) to validate that the Federate
 *  is declared as time-regulator in the HLA Federation.
 */
public void timeRegulationEnabled(LogicalTime theFederateTime)
        throws InvalidFederationTime,
        EnableTimeRegulationWasNotPending, FederateInternalError {
    timeRegulator = true;
    if (_debugging) {
        _debug(HlaManager.this.getDisplayName() + " INNER"
                + " timeRegulationEnabled() - timeRegulator = "
                + timeRegulator);
```

```
    }
}

/** Callback delivered by the RTI (CERTI) to validate that the Federate
 *  is declared as time-constrained in the HLA Federation.
 */
public void timeConstrainedEnabled(LogicalTime theFederateTime)
        throws InvalidFederationTime,
        EnableTimeConstrainedWasNotPending, FederateInternalError {
    timeConstrained = true;
    if (_debugging) {
        _debug(HlaManager.this.getDisplayName() + " INNER"
                + " timeConstrainedEnabled() - timeConstrained = "
                + timeConstrained);
    }
}

/** Callback (TAG) delivered by the RTI (CERTI) to notify that the
 *  Federate is authorized to advance its time to <i>theTime</i>.
 *  This time is the same or smaller than the time specified
 *  when calling the nextEventRequest() or the timeAdvanceRequest()
 *  services.
 */
public void timeAdvanceGrant(LogicalTime theTime)
        throws InvalidFederationTime, TimeAdvanceWasNotInProgress,
        FederateInternalError {
    logicalTimeHLA = theTime;
    timeAdvanceGrant = true;
    if (_debugging) {
        _debug(HlaManager.this.getDisplayName() + " INNER"
                + " timeAdvanceGrant() - TAG("
                + logicalTimeHLA.toString() + ") received");
    }
}

// HLA Federation Management services (callbacks).
// Synchronization point services.

/** Callback delivered by the RTI (CERTI) to notify if the synchronization
 *  point registration has failed.
 */
public void synchronizationPointRegistrationFailed(
        String synchronizationPointLabel) throws FederateInternalError {
    synchronizationFailed = true;
    if (_debugging) {
        _debug(HlaManager.this.getDisplayName() + " INNER"
                + " synchronizationPointRegistrationFailed()"
                + " - synchronizationFailed = " + synchronizationFailed);
    }
}

/** Callback delivered by the RTI (CERTI) to notify if the synchronization
 *  point registration has succeed.
 */
public void synchronizationPointRegistrationSucceeded(
        String synchronizationPointLabel) throws FederateInternalError {
    synchronizationSuccess = true;
    if (_debugging) {
        _debug(HlaManager.this.getDisplayName() + " INNER"
                + " synchronizationPointRegistrationSucceeded()"
                + " - synchronizationSuccess = "
                + synchronizationSuccess);
    }
}
```

```
        /** Callback delivered by the RTI (CERTI) to notify the announcement of
         *  a synchronization point in the HLA Federation.
         */
        public void announceSynchronizationPoint(
                String synchronizationPointLabel, byte[] userSuppliedTag)
                        throws FederateInternalError {
            inPause = true;
            if (_debugging) {
                _debug(HlaManager.this.getDisplayName() + " INNER"
                        + " announceSynchronizationPoint() - inPause = "
                        + inPause);
            }
        }

        /** Callback delivered by the RTI (CERTI) to notify that the Federate is
         *  synchronized to others Federates using the same synchronization point
         *  in the HLA Federation.
         */
        public void federationSynchronized(String synchronizationPointLabel)
                throws FederateInternalError {
            inPause = false;
            if (_debugging) {
                _debug(HlaManager.this.getDisplayName() + " INNER"
                        + " federationSynchronized() - inPause = " + inPause);
            }
        }
    }
}
```