To link to this article : DOI:10.1177/0037549714531054
URL : http://dx.doi.org/10.1177/0037549714531054

To cite this version : Chaudron, Jean-Baptiste and Saussié, David and Siron, Pierre and Adelantado, Martin *Real-time distributed simulations in an HLA framework: Application to aircraft simulation.* (2014) Simulation, vol. 90 (n° 6). pp. 627-643. ISSN 0037-5497

Real-time distributed simulations in an HLA framework: Application to aircraft simulation

Jean-Baptiste Chaudron[1], David Saussié[2], Pierre Siron[1] and Martin Adelantado[3]

Abstract
This paper presents some ongoing research carried out in the context of the PRISE Project (Research Platform for Embedded Systems Engineering). This platform has been designed to evaluate and validate new embedded system concepts and techniques through a special hardware and software environment. Since much actual embedded equipment is not available, corresponding behavior is simulated within a high-level architecture (HLA) federation implemented with a run-time infrastructure (RTI) called CERTI and developed at ONERA. HLA is currently largely used in many simulation applications, but the limited performances of the RTIs raise doubts over the feasibility of HLA federations with real-time requirements. This paper addresses the problem of achieving real-time performances with the HLA standard. Several experiments are discussed using well-known aircraft simulators such as Microsoft Flight Simulator, FlightGear, and X-plane connected with the CERTI RTI. The added value of these activities is to demonstrate that according to a set of innovative solutions, HLA architecture is well suited to achieve hard real-time constraints. Finally, a formal model guaranteeing the schedulability of concurrent processes is also proposed.

Keywords
High-level architecture, HLA, distributed simulation, CERTI middleware, aircraft simulation, real-time simulation, formal model, schedulability

1. Introduction

1.1. Problem statement

The distributed computing paradigm proposes a high performance solution thanks to advances in network technologies. Different programs located on several computers interact all together in order to achieve a global common goal. However, designers and developers of distributed software applications have to face several problems such as heterogeneity of the various hardware components as well as both operating systems and communication protocols. Development of middleware standards like CORBA allows to consistently face these problems.[1] The term middleware describes a software agent operating as an intermediary between distributed processes (Figure 1). This software must be considered in the interoperability framework; it is a connectivity software, which enables the execution of several interacting applications on one or more linked computers.

Modern flight simulation techniques and implementations often result in many sophisticated and complex calculations that require a high level of computing power. Several flight simulator applications often require their services to be delivered with respect to a given instant of time (deadline). This issue constitutes the problematic of real-time systems which are defined as systems in which the correctness of the system not only depends on the logical results of computation, but also on the time at which these results are produced.[2] Real-time systems are broadly classified into two categories based on the nature of the deadline, namely, *hard real-time systems*, in which the consequences of not executing a task before its deadline

[1]Institut Supérieur de l'Aéronautique et de l'Espace (ISAE), Mathematics, Computer Science and Control Theory Department, Toulouse, France
[2]École Polytechnique de Montréal, Electrical Engineering Department, Montréal, Québec, Canada
[3]French Aeronautics and Space Research Center (ONERA), Information Modeling and Processing Department, Toulouse, France

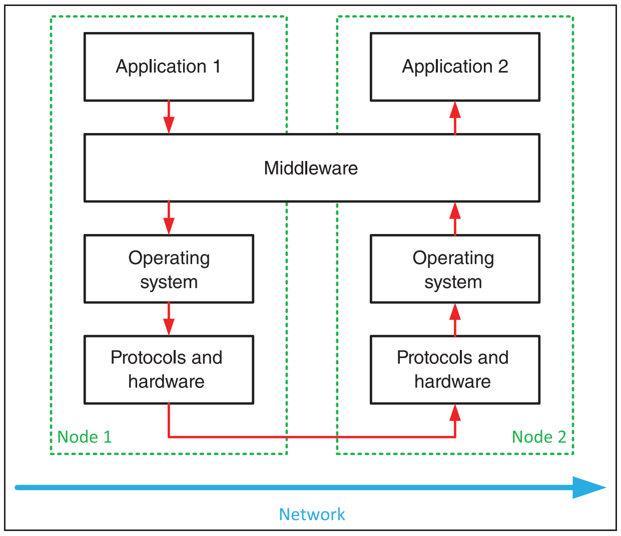

Figure 1. Middleware illustration.

may be catastrophic and *soft real-time systems*, in which the utility of results produced by a task with a soft deadline decreases over time after the deadline expires. Examples of typical hard real-time systems are flight controls and nuclear plant controls. Telephone switching system and image processing applications are examples of soft real-time systems. The application presented in this article is concerned with both types of real-time system characteristics.

Traditional standards and middleware architectures are not suitable for supporting real-time constraints. Real-time aircraft software and hardware components interconnected with middleware such as CORBA[3] have led to advances in current standards to include real-time properties (e.g. real-time CORBA[4] or more recently DDS[5]).

Our work is focused on the use of the High Level Architecture (HLA) IEEE 1516-2000 standard[6–8] to develop, interconnect and maintain a flight simulator. (In August 2010, a new revision of the standard (IEEE 1516-2010) was published by SISO/IEEE, which is not treated here.) However, recent works to include real-time specifications and properties to HLA standard are less advanced than other ones.[9] This article explains how we proceed to implement and test this simulator and how we validate real-time behavior on our computing platform. The use of a distributed simulation architecture to study distributed embedded systems and hardware should provide a more natural and flexible framework for new researches in the domain.

1.2. Background

Simulation is a well-established technique used in the man–machine system area for training, evaluation of performance and research. Flight simulation re-creates how an aircraft flies under the action of aerodynamic, thrust and gravity forces in the standard atmosphere (with possible models for wind and turbulence).[10] To achieve this goal, a flight simulator must first rely on a solid mathematical description of the aircraft and its environment; the more accurate the model, the more realistic and reliable the simulation will be. A digital computer running a real-time operating system then computes this model. The simulation can finally be completed with input organs (e.g. yoke-pedal systems, joysticks), display screens, cockpit-like environment and mechanical devices reproducing the aircraft motion (e.g. Stewart platform).

The choice of a distributed standard and its underlying middleware is an important requirement to obtain a high fidelity, valid and scalable real-time flight simulation. This involves choosing an operating system, a programming language and hardware in compliance with the selected middleware. Many studies and integration simulations are elements of the Airbus industrial process,[11] but the different models are proprietary (and sometimes certified) as well as the run-time infrastructure (RTI). Previous works focused on the DDS standard for a flight simulator basis[12] or the HLA standard.[13] In Huiskamp et al.[14] and Kuijpers et al.,[15] the authors describe a comprehensive methodology to enable the interoperability between various components for the development of simulators. However, the method used to ensure the real-time behavior is not clearly detailed and HLA middleware is not specified.

The RTI (HLA underlying middleware) is the distributed software used for interconnecting various federates to a global federation execution. In Lemmers et al., the authors use the RTI-NG, which was the first RTI developed and used by the US Department of Defense;[13,16] this RTI is no longer maintained. Since then, several approaches have been investigated to add real-time properties to the HLA standard and underlying software. These works include optimized time management services,[17] multi-threaded synchronous processes for RTI[17–19] and global scheduling services.[18,19] These different techniques allow an improved use of system resources, better scalability and also a higher reactivity of services provided by the RTI.

1.3. Our approach

Our work takes place in a global project named PRISE (Plate-forme de Recherche et d'Ingénierie des Systèmes Embarqués (Research Platform for Embedded Systems Engineering)). The main focus of this project is to study new embedded system concepts and techniques through a special hardware and software environment. Our platform is built around the following components:

- Hardware: Four real-time nodes with Opteron six core processors, two graphical HP computer stations with Intel Xeon processors and high

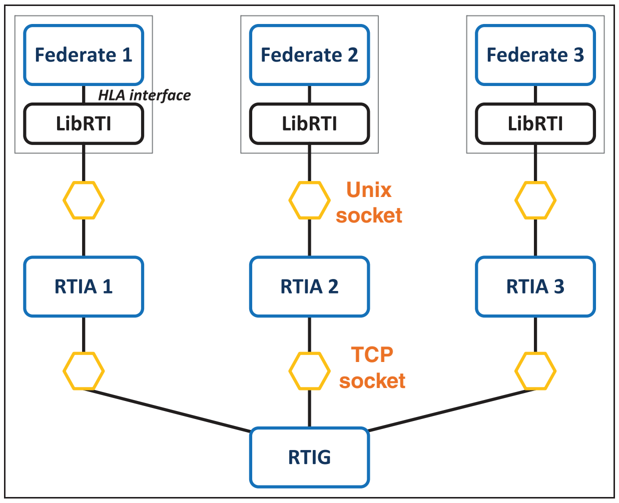

Figure 2. CERTI architecture.

performance GP-GPU, an Ethernet gigabit switch on a dedicated network, and also two input organs (Yoke/Throttle/Pedal systems).

- Software: Linux Red Hawk Operating System[20] compliant with POSIX real-time standard.[21] This RTOS has been already used in the simulation domain by TNO laboratory to run their own RTI implemented in C++; this operating system is suitable for real-time computing.[22]
- A distributed clock technology allowing distributing same clock reference to each node.[23]

Since 1996, the French Aerospace Laboratory (ONERA) has been developing his own open-source middleware RTI, namely CERTI, compliant with the HLA standard.[24] CERTI runs under several operating systems including Linux, Mac OS X, and Windows. CERTI is recognizable through its original architecture of communicating processes (Figure 2); it is a distributed system involving a global process (RTIG (run-time infrastructure gateway)), a local one (RTIA (run-time infrastructure ambassador)), as well as a library (libRTI) linked with each federate. Each federate process interacts locally with an RTIA through a Unix-domain socket. The RTIA processes exchange messages over the network with the RTIG process via TCP and/or UDP sockets in order to run the distributed algorithms associated with the RTI services.

One of the many benefits of choosing CERTI as middleware is the complete control over its implementation; this greatly facilitates the integration of changes in the source code to ensure temporal predictability. The suitability of CERTI to meet real-time constraints was first tested in studies for ONERA/CNES satellite formation flying.[25] They showed that CERTI (in its original version) is able to manage multiple real-time federates with short execution and communication cycles. In the present paper, we mainly focus on the real-time characteristics and properties of a flight simulator developed in the HLA standard framework. We propose a comprehensive bottom-up approach from the specifications of the flight simulator (including real-time requirements) to the validation by using formal approaches and experiments on a dedicated hardware/software architecture.

1.4. *Paper organization*

The paper is structured as follows. Section 2 describes the architecture of the flight simulator as well as the different characteristics of each component. Section 3 outlines the two run-time models, i.e. data flow and time management models. The real-time formal validation is discussed in Section 4. Finally, Section 5 gathers experimental results on the PRISE platform.

2. Simulation architecture

This section is devoted to the description of the global simulation (federation) and its components (federates). As a whole, the federation is a hybrid system whose different parts are described either by discrete-event models, or by continuous-time models. The need for a modular and scalable platform led us to the use of such a distributed simulation where the model of the aircraft is split into several different subsystems. Indeed, if the first version of our federation is essentially software-oriented, it will be possible to replace some federates by dedicated hardware equipment, i.e. small-scale models of control surfaces or embedded systems.

2.1. *Global view*

The PRISE HLA Federation is composed of nine federates, each representing a specific part of the aircraft or the environment itself (Figure 3).

We provide here a brief description of the federates:

Federate 1: Pilot Inputs
The Pilot Inputs federate acquires the pilot orders transmitted by a yoke/throttle/pedals system. The elevator, aileron, and rudder axes can then be commanded through the flight control laws. The engine thrust is not directly controlled as the auto-throttle implemented in the flight controller federate alleviates the pilot workload.
Federate 2: Flight Controller
The Flight Controller federate is in charge of the aircraft control; it implements the classical autopilot functions (e.g. speed and altitude hold control systems) as well as the flight control functions (i.e. control and stability augmentation systems). The aircraft can then be turned in automatic mode where no pilot is needed or in manual mode where the pilot interacts through the yoke/throttle/pedals system. The design of the flight control is a huge task and truly is a complex problem in itself; only essential functions were

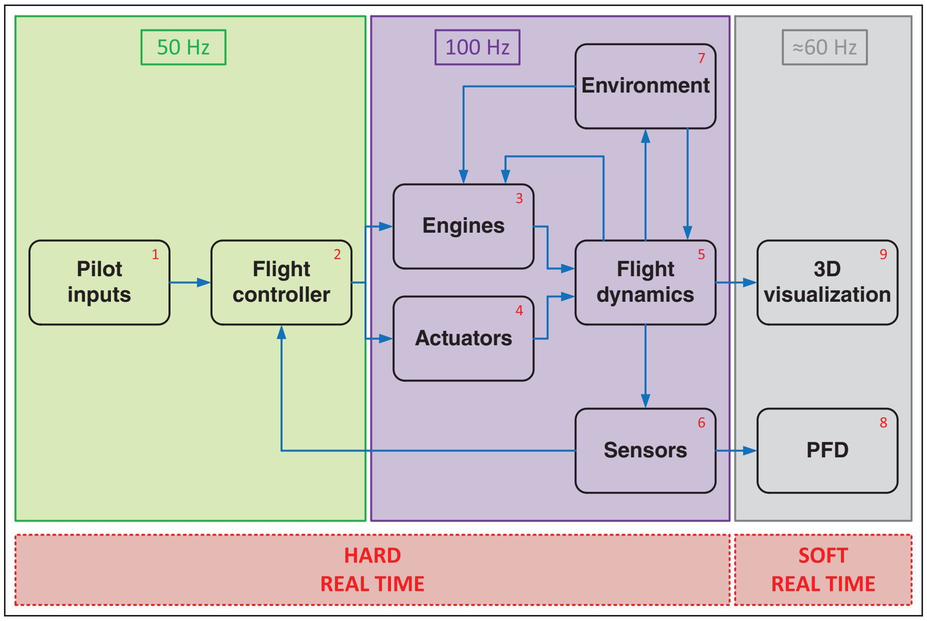

Figure 3. PRISE simulation architecture.

first developed to operate the aircraft with the prospect to enrich them as the PRISE project continues.

Federates 3 and 4: Engines and Actuators

The Actuators federate gathers all the control surface actuators whose deflections change the aerodynamic forces and thus influence the aircraft motion. We consider here left and right ailerons, left and right elevators and rudder. Each control surface is modeled by a second-order system with position and rate saturations to enforce realism. Delay, bias, and hysteresis phenomena are also considered. The Engines federate simulates two high-bypass turbofan engines whose characteristics change with the atmospheric conditions (i.e. temperature) and the aircraft Mach number. Both federates receive respectively angular deflection and throttle commands from the flight controller federate.

Federate 5: Flight Dynamics

The Flight Dynamics federate represents the core of the simulation model as it computes the equations of motion. Under the action of aerodynamic, gravity, and propulsion forces, the aircraft state evolves accordingly. As a whole, the flying equations are a set of 12 first-order differential equations. The aerodynamic coefficients are implemented in the form of look-up tables with many entries; therefore, computing the aerodynamic forces can be a demanding computational task. The reader is referred to Nelson and Stevens and Lewis for a complete description of these equations.[26,27]

Federate 6: Sensors

To perform control, the flight controller needs measurements of the aircraft state that are available through sensors. The Sensors federate simulates the different sensors (e.g. IRU (inertial reference unit)/IMU (inertial measurement unit)/ADIRU (air data inertial reference unit)). In a control loop, sensors are not necessarily neutral elements and they have their own dynamics; here they are modeled as first- or second-order low-pass Butterworth filters whose cutoff frequency depends on the nature of the measurement (e.g. rates, accelerations). Delay, bias, drift, and noise phenomena are also implemented to augment realism.

Federate 7: Environment

The Environment federate simulates the US Standard Atmosphere 1976.[28] Based on the altitude provided by the Flight Dynamics federate, it calculates the corresponding atmospheric variables (temperature, pressure, air density, and sound speed) and feeds them back to other federates. Different types of wind and turbulence are also simulated (i.e. wind shear, wind gust, Dryden and Von Karman turbulences) to reproduce realistic weather conditions.

Federates 8 and 9: Primary Flight Display and 3D Visualization

The sensor outputs are used by the Primary Flight Display (PFD) federate to display essential flight information in a cockpit-like interface (e.g. speed, vertical speed, attitude, altitude, and heading). The actual aircraft position and orientation are fed to the 3D Visualization federate that takes in charge the display in a virtual environment such as Flight Simulator,[29] FlightGear[30] and X-plane.[31] The Virtual Air project provides plug-ins to integrate these software components in an HLA federation.[32]

The federation is divided into two parts:

- The first part is submitted to *hard real-time* constraints and has to ensure that all deadlines are met for each federate. The Pilot Inputs and Flight

Controller federates have a 50 Hz refresh rate, corresponding to an average frequency of the usual avionics system. The other federates (Engines, Actuators, Flight Dynamics, and Sensors) have a 100 Hz refresh rate; they simulate continuous-time systems modeled by differential equations and solved by numerical methods (Section 2.2).

- The second part deals with *soft real-time* constraints; the goal is to meet a certain subset of deadlines in order to optimize some application specific criteria. In our case, the 3D visualization and PFD federates have an average refresh rate of 60 Hz in order to be fluid for human eyes.[33]

2.2. Implementation

Federates can be represented by continuous-time models, discrete-event models or possibly both (hybrid models). Federates 3–7 model continuous-time systems that can be roughly described by ordinary differential equations (ODE).[27] They can be rearranged in a set of first-order differential equations, i.e. a state-space model:

$$\dot{x}(t) = f(x(t), u(t), t) \tag{1}$$

$$y(t) = g(x(t), u(t), t) \tag{2}$$

$$x(t_0) = x_0 \tag{3}$$

where $x \in \mathbb{R}^n$ denotes the state vector, $u \in \mathbb{R}^m$ the input vector, $y \in \mathbb{R}^p$ the output vector and t the time. In the general case, the functions $f : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^n$ and $g : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^p$ are nonlinear vector functions, resp. named state and output functions. Basically, the state equation (1) describes the dynamical behavior of the system with its internal description (the state vector x) whereas the output equation (2) yields all the variables of interest (e.g. the quantities measured by the sensors) based upon the states and the inputs. The system is solved with the initial conditions (3).

Most of the time, these systems cannot be solved analytically, and, regardless, in a digital environment, values do not evolve continuously with time. We then have to resort to integration methods (e.g. Euler, Heun, Adams–Basforth, Runge–Kutta methods), which perform interpolation at each integration step to approximate the derivative at a discrete subset of time points. In the case of explicit Runge–Kutta (RK) methods, this yields recurrence equations of the form:

$$x(t_{n+1}) = x(t_n) + \sum_{i=1}^{k} c_i R_i \tag{4}$$

where

$$R_1 = \Delta t.f(x(t_n), u(t_n), t_n) \tag{5}$$

$$R_i = \Delta t.f(x(t_n) + \sum_{j=1}^{i} a_{i,j} R_j, u(t_n + \Delta t.b_i), t_n + \Delta t.b_i) \tag{6}$$

with $a_{i,j}$, b_i and c_i are algorithm parameters and Δt the integration step ($t_n = n\Delta t$). The quality of the solution usually increases with the order of the method k. The simplest RK method is the forward Euler method given by the formula

$$x(t_{n+1}) = x(t_n) + \Delta t.f(x(t_n), u(t_n), t_n) \tag{7}$$

The classical RK method, referred to as RK4, is given by

$$x(t_{n+1}) = x(t_n) + \frac{1}{6}(R_1 + 2R_2 + 2R_3 + R_4) \tag{8}$$

with

$$R_1 = \Delta t.f(x(t_n), u(t_n), t_n) \tag{9}$$

$$R_2 = \Delta t.f\left(x\left(t_n + \frac{\Delta t}{2} R_1\right), u\left(t_n + \frac{\Delta t}{2}\right), t_n + \frac{\Delta t}{2}\right) \tag{10}$$

$$R_3 = \Delta t.f\left(x\left(t_n + \frac{\Delta t}{2} R_2\right), u\left(t_n + \frac{\Delta t}{2}\right), t_n + \frac{\Delta t}{2}\right) \tag{11}$$

$$R_4 = \Delta t.f(x(t_n + \Delta t R_3), u(t_n + \Delta t), t_n + \Delta t) \tag{12}$$

The smaller the integration step Δt, the more accurate the solution. For each time step Δt of the real-time simulation, the precision can still be improved within the federate; thus the time step is locally decreased, improving the accuracy of the solving. The recurrence equation (4) is then repeated N times with decreased time step $\Delta t/N$. This way, we avoid using too small steps inside the federation (leading to an increase of HLA messages). Nevertheless we suppose that the inputs coming from the Actuators and Engines federates remain constant during one time step, Δt, which is not completely rigorous. Comparisons with equivalent Simulink models and variable-step methods showed that an RK4 method with time step $\Delta t = 0.01$s was sufficient to deliver reliable results. Indeed, a higher rate would imply more exchanged messages, and so the real-time constraints would hardly be satisfied.

Contrary to the former federates which represent continuous-time processes, Federate 2 simulates discrete-time processes. Indeed, flight controls as well as other avionic functions, are intended to run on a digital platform with an a priori execution rate (actually, different execution rates depending on the executed tasks). Generally, flight controllers are designed in the continuous-time domain, then digitalized with bilinear or Tustin transforms.[34] The control system engineer must then choose an adequate execution rate (compatible with the hardware)

Table 1. Object/Attribute table published by the Flight Dynamics federate.

Attribute Table

Object	Attribute	Data-type	Card.	Units	Update type	T/A	U/R
AIRCRAFT_ POSITION	LONGITUDE	Double	I	deg	Periodic	TA	UR
	LATITUDE	Double	I	deg	Periodic	TA	UR
	ALTITUDE	Double	I	m	Periodic	TA	UR
AIRCRAFT_ ORIENTATION	PHI	Double	I	rad	Periodic	TA	UR
	THETA	Double	I	rad	Periodic	TA	UR
	PSI	Double	I	rad	Periodic	TA	UR
AIRCRAFT_ VELOCITY	SPEED	Double	I	m/s	Periodic	TA	UR
	VERTICAL_SPEED	Double	I	m/s	Periodic	TA	UR
	ALPHA	Double	I	rad	Periodic	TA	UR
	BETA	Double	I	rad	Periodic	TA	UR
	MACH	Double	I	$\varnothing$	Periodic	TA	UR
AIRCRAFT_ ANGULAR_ VELOCITY	ROLL	Double	I	rad/s	Periodic	TA	UR
	PITCH	Double	I	rad/s	Periodic	TA	UR
	YAW	Double	I	rad/s	Periodic	TA	UR
AIRCRAFT_ ACCELERATION	ACC_X	Double	I	m/s^2	Periodic	TA	UR
	ACC_Y	Double	I	m/s^2	Periodic	TA	UR
	ACC_Z	Double	I	m/s^2	Periodic	TA	UR

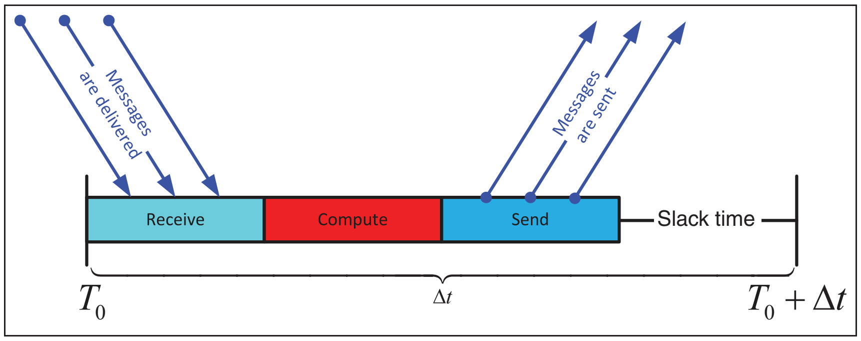

Figure 4. Time step.

such that the digital controllers will still guarantee closed-loop stability and performance.

2.3. *Object model template*

Messages exchanged between federates are based upon the global federation object model (HLA FOM) compliant with HLA Object Model Template (OMT).[8] This object model provides the global definitions of all objects (and their attributes) that are available for all federates. As an example, Table 1 shows the object/attribute table for the objects published by the Flight Dynamics federate. Thereafter, the Sensors, 3D Visualization, Engines, and Environment federates subscribe to all/part of the attributes.

3. Run-time execution mode

3.1. *Towards periodic federates*

Fujimoto and McLean introduced the concept of repeatability within real-time simulations.[35,36] Federates engaged in a real-time simulation repeat the same pattern of execution periodically with a time step noted Δt. During each step, federates carry out four phases: (a) the reception phase where messages are delivered, (b) the calculation phase where internal state is updated, (c) the transmission phase where messages are sent, and (d) the slack time phase (Figure 4).

We focus on two different run-time modes based on periodic federates; each has its strengths and weaknesses. A previous work on satellite formation flying has highlighted the necessity of adding a synchronization phase to maintain consistency between each cycle through a common wall clock time (WCT).[37]

The first execution mode is the data flow model described in Section 3.2. This execution mode is only scheduled by the communication flow between each federate; indeed each federate waits for a new input to run its local algorithm and provides its own output to the rest of the federation. This approach could only be used on synchronous distributed systems like PRISE Red Hawk RCIM synchronized nodes.

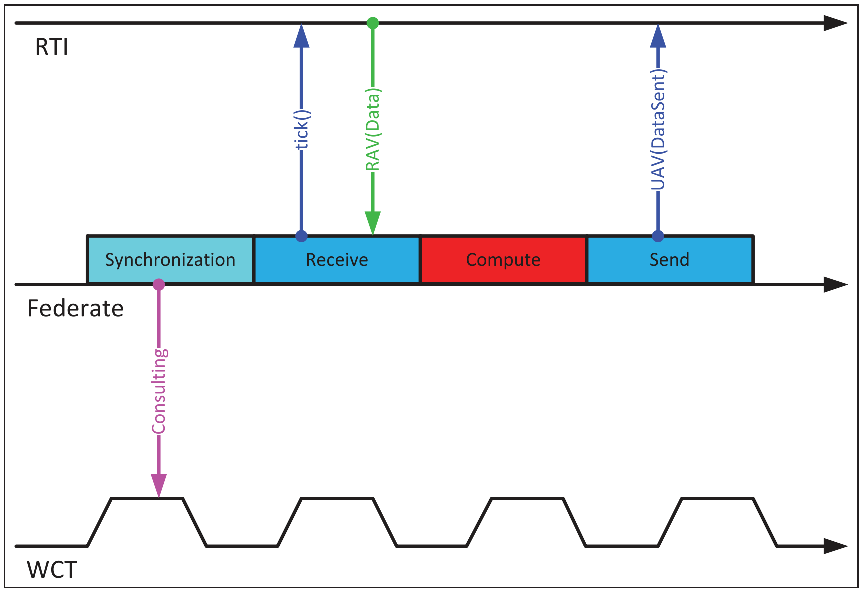

Figure 5. Periodic data flow federate step.

The second execution mode (Section 3.3) uses HLA time-management mechanisms, as in the original Fujimoto and McLean works on real-time simulations. During the run-time, each federate's computations and communications are scheduled by time management principles and algorithms. A suitable deployment of these techniques ensures a consistent temporal behavior on a common time reference: the *simulated time*. This approach is probably the best way to maintain consistency between federates located on asynchronous computers (no common WCT needed).

3.2. Data flow model

Federates communicate using HLA basic *publish and subscribe* principles through RTI service calls like `updateAttributeValues` (Figure 5). The receiver federate is waiting for a `reflectAttributesValues` callback during the reception phase after invoking a specific CERTI `tick` service in its blocking version.[25] After receiving all the expected attribute values for the current time step, each federate runs its own algorithm.

The main interest of this run-time execution mode is the simplicity of modeling its behavior with a formal model compliant with real-time scheduling policies and techniques (Section 4.3). Nevertheless, this approach is hardly suitable for adding new federates or for plugging existing federates to another federation execution. Most of all, there is no safety guarantee during the run-time. If the application is not well scheduled, a federate can remain blocked (i.e. waiting for an expected data). Special attention must be paid to ensure a reliable execution of the whole federation.

3.3. Time management model

Time management mechanisms provided by the HLA standard are one of the main benefits of this standard architecture.[38] These services allow a consistent global time throughout the whole simulation by using different methods and algorithms. Specifically, each simulation message is assigned a time-stamp, and the RTI ensures that messages are delivered to each federate in time-stamp order, and no message is delivered to a federate in its past. The main operation required to implement time management services is the calculation of the greatest available logical time (GALT) of each federate (called lower bound on time-stamp (LBTS) in the HLA US DoD 1.3 standard). The GALT value is critical because any message with time-stamp less than GALT can then be delivered safely to the federate while guaranteeing time-stamp order delivery. Real-time simulations could use time management to ensure the good behavior of the whole simulation.[39] According to the HLA standard, all federates concerned by hard real-time constraints are both regulators and constrained; the soft real-time federates (PFD and 3D Visualization) are only constrained because they are only subscribers.

The simplest functional behavior that can be used for time-management model is depicted in Figure 6. The federate asks to advance to the next cycle by invoking the `timeAdvanceRequest` service (`TAR`). The RTI grants the logical time advance (guaranteeing causality constraints) by first invoking all the available `reflectAttributeValue` callbacks (`RAV`) and finally by accepting the time advance through the invocation of the `timeAdvanceGrant` callback (`TAR`). Once

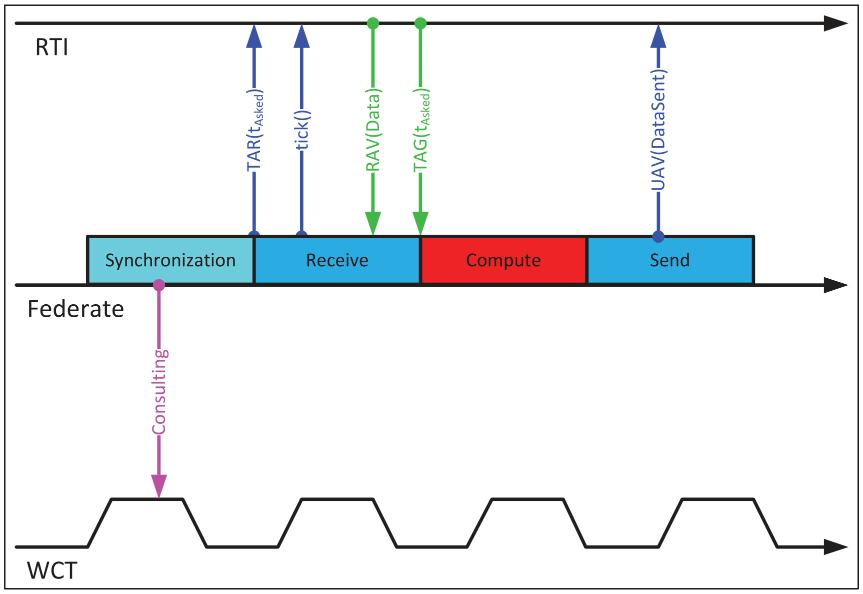

Figure 6. Periodic time managed federate step.

the federate has finished updating its state, it invokes the `updateAttributeValues` service (`TAR`) to send the new attribute values.

Due to the presence of several loops within the federation (Figure 3), this functional behavior has to be adapted to allow multiple reception/update phases within a same cycle. Indeed, for each step Δt, some federates have to receive data updated for the current cycle and then data updated for the next cycle (in simulated time). A typical example is as follows: the Flight Dynamics federate publish the current altitude; the Environment federate receives it and publishes the corresponding atmosphere conditions (e.g. air density, wind); the Flight Dynamics federate receives them and can compute the corresponding aerodynamic forces. The mixing of `nextEventRequest` (`NER`) and `timeAdvanceRequest` (`RAV`) services helped solving the loop cases (not presented here).

Finally, simulated time advance by calling `NER` or `TAR` services can be correlated by consulting the WCT (WCT RCIM hardware clock) to ensure the respect of real-time constraints for each federate (Figure 6).

4. Real-time formal validation

4.1. Necessity of a formal model

A formal model guaranteeing the schedulability of concurrent processes is essential to validate real-time periodic federates and to ensure real-time communications with CERTI. One commonly proposed way for designing hard real-time systems is to build the system from a number of periodic tasks, each assigned static priorities, and dispatched at run-time according to a static priority preemptive scheduling algorithm. The main motivation of *scheduling theory* research using this approach has been to derive an analysis that can bind the behavior of the tasks at run-time. Original work by Liu and Layland provides an a priori analysis to determine if a set of periodic tasks would be guaranteed to meet their deadlines (on a monoprocessor architecture).[40] They proposed a definition of a periodic task based on timing parameters. A periodic task τ_i is a quadruplet $<\boldsymbol{r_i}, \boldsymbol{C_i}, \boldsymbol{D_i}, \boldsymbol{P_i}>$ with:

- $\boldsymbol{r_i}$ the time of initial activation of the task;
- $\boldsymbol{C_i}$ the worst-case execution time;
- $\boldsymbol{D_i}$the deadline;
- $\boldsymbol{P_i}$ the period.

Data flow federates (Section 3.2) simulate communicating periodic processes that could be considered periodic tasks that communicate by using HLA principles; thus HLA communications could be represented by periodic messages. Let $\boldsymbol{m_{i,j}}$ denote a periodic message representing data exchange between two federates **FEDi** and **FEDj**. A periodic message $\boldsymbol{m_{i,j}}$ could be then described by a quadruplet $<\boldsymbol{r_{i,j}}, \boldsymbol{C_{i,j}}, \boldsymbol{D_{i,j}}, \boldsymbol{P_{i,j}}>$ with:

- $\boldsymbol{r_{i,j}}$ the time of initial sending of the message by the federate;
- $\boldsymbol{C_{i,j}}$ the worst case transit time (WCTT) through CERTI;
- $\boldsymbol{D_{i,j}}$ the deadline for message transmission;
- $\boldsymbol{P_{i,j}}$ the period of production of message by federate.

Under these assumptions, we could use this formalism to describe our data flow application model. The single processor approach[37] (based on simple precedence

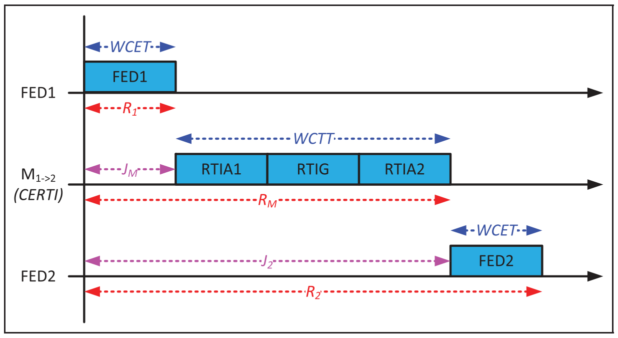

Figure 7. Periodic time managed federate step.

constraint and Deadline Monotonic algorithm) could be extended by using Tindell and Clark's holistic techniques[41] to ensure end-to-end validation of our simulation. Indeed their work allows taking into account data dependency between the scheduling of tasks and messages in distributed real-time systems. The principle is rather simple: delays due to data dependency (messages or functional precedences) between tasks could be represented by the *release jitter* of a task (noted J_i for federate i). Indeed, the WCRT (noted R_i for federate i) is the longest time ever taken by a task to complete its required computation. Figure 7 illustrates an example of two federates **FED1** and **FED2** and corresponding release jitter and worst-case response time.

For a given system, a recursive algorithm finds the WCRT for the whole set of tasks and messages. Finally, if every WCRT is always less than the corresponding deadline (for tasks and messages), we ensure the correct behavior of our distributed application. However, this model is not suited to time management modeling; indeed time management involves more complex mechanisms, which are difficult to combine with a formal scheduling model (Section 4.3).

4.2. *Run-time execution an communication characteristics*

The calculation of worst-case execution time (WCET) is a key issue for successfully schedule processes; it allows determining the C_i parameter value for a task (Section 4.1). Calculation of the WCET (compute section in Figures 5 and 6) should take into account specific calculations made by the federate. As described in Section 2.2, the WCET depends on the number of iterations N chosen for local algorithms. Measurements were made for the Flight Dynamics federate (Table 2), Actuators federate (Table 3), Sensors federate (Table 4), and Engines federate (Table 5). The Flight Controller federate WCET evaluation does not depend on a number of iterations of the local algorithm (Table 6).

Calculation of the worst case transit time (WCTT) values for all messages through CERTI must take into account three phases (Figure 8). Phase 1 is the copy on local host Unix domain socket and the local computation of the sender federate associated RTIA process. Phase 2 describes

Table 2. Flight Dynamics federate WCET.

Flight Dynamics federate	Min. (ms)	Mean (ms)	Max. (ms)	Std dev. (ms)
100 iterations per calculation	0.96	0.963	1.004	0.005
500 iterations per calculation	4.743	4.753	4.78	0.0047
1000 iterations per calculation	9.58	9.704	9.751	0.048

Table 3. Actuators federate WCET.

Actuators federate	Min. (ms)	Mean (ms)	Max. (ms)	Std dev. (ms)
100 iterations per calculation	0.022	0.0224	0.024	0.0005
500 iterations per calculation	0.11	0.112	0.123	0.002
1000 iterations per calculation	0.217	0.221	0.244	0.0041

Table 4. Sensors federate WCET.

Sensors federate	Min. (ms)	Mean (ms)	Max. (ms)	Std dev. (ms)
100 iterations per calculation	0.077	0.0775	0.088	0.0016
500 iterations per calculation	0.377	0.378	0.389	0.002
1000 iterations per calculation	0.753	0.755	0.764	0.0016

Table 5. Engines federate WCET.

Engines federate	Min. (ms)	Mean (ms)	Max. (ms)	Std Dev. (ms)
100 iterations per calculation	0.024	0.025	0.047	0.0028
500 iterations per calculation	0.123	0.124	0.145	0.0028
1000 iterations per calculation	0.245	0.246	0.27	0.0029

Table 6. Controller federate WCET.

Controller federate	Min. (ms)	Mean (ms)	Max. (ms)	Std dev. (ms)
Calculation	0.01	0.07	0.3	0.005

Table 7. CERTI WCTT evaluation.

Data size	Min. (ms)	Mean (ms)	Max. (ms)	Std dev. (ms)
512 Bits	0.218	0.248	0.308	0.015
1024 Bits	0.222	0.230	0.312	0.013
5120 Bits	0.284	0.315	0.364	0.017
10240 Bits	0.349	0.381	0.446	0.016

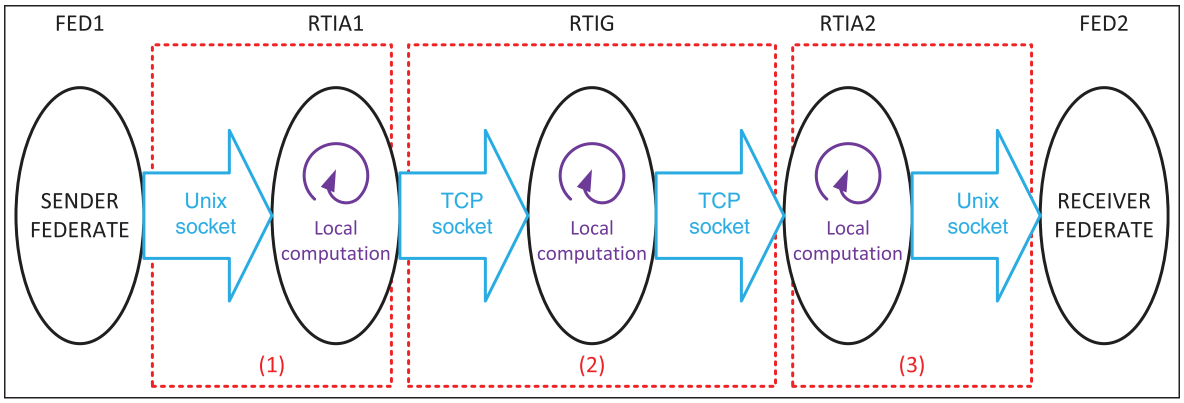

Figure 8. CERTI communication steps.

the time to read and write on different communication TCP or UDP sockets over the network or on the local host, and the time needed for RTIG local computation. Phase 3 is the copy on local host Unix domain socket and the local computation of receiver federate associated RTIA process.

Table 7 gathers experimental measurements of CERTI WCTT with respect to data size on one single PRISE Red Hawk node.

4.3. *Data flow validation*

We can now validate the data flow model. As there are seven periodic federates in the federation (the hard real-time part), the real-time formal model can be simply represented by seven tasks consistent with the numbering used in Figure 3:

$$\begin{aligned}
&\mathbf{FED1}: \ <0, C_1, \mathbf{20}, \mathbf{20}>;\\
&\mathbf{FED2}: \ <0, C_2, \mathbf{20}, \mathbf{20}>;\\
&\mathbf{FED3}: \ <0, C_3, \mathbf{10}, \mathbf{10}>;\\
&\mathbf{FED4}: \ <0, C_4, \mathbf{10}, \mathbf{10}>;\\
&\mathbf{FED5}: \ <0, C_5, \mathbf{10}, \mathbf{10}>;\\
&\mathbf{FED6}: \ <0, C_6, \mathbf{10}, \mathbf{10}>;\\
&\mathbf{FED7}: \ <0, C_7, \mathbf{10}, \mathbf{10}>.
\end{aligned}$$

The **Joystick** and **Flight Controller** federates (resp. **FED1** and **FED2**) have a refresh rate of 50 Hz, so their

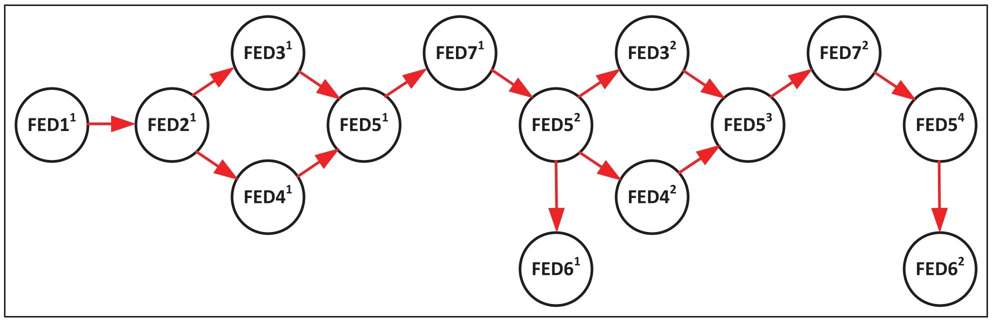

Figure 9. Precedence tree for PRISE application.

period ($\boldsymbol{P_i}$ parameter) and deadline ($\boldsymbol{D_i}$ parameter) is 20 ms. The five other federates have a computing frequency of 100 Hz, so their corresponding period and deadline are 10 ms.

However, these communicating tasks do not run at the same rate, which is not compliant with simple precedence scheme to represent HLA communication. To solve this problem, the task graph has to be unfolded. In this sense, each previous task is subdivided into a set of sub-tasks. The period of each sub-task is then equal to the hyper-period of the set of basic tasks (i.e. the *least common multiple* of all task periods). Here, the least common multiple is equal to 20, and we obtain a set of 12 sub-tasks:

$$\begin{aligned}
&\mathbf{FED1^1}: \; < 0, \boldsymbol{C_1}, \mathbf{20}, \mathbf{20} >;\\
&\mathbf{FED2^1}: \; < 0, \boldsymbol{C_2}, \mathbf{20}, \mathbf{20} >;\\
&\mathbf{FED3^1}: \; < 0, \boldsymbol{C_3}, \mathbf{10}, \mathbf{20} >;\\
&\mathbf{FED3^2}: \; < 0, \boldsymbol{C_3}, \mathbf{20}, \mathbf{20} >;\\
&\mathbf{FED4^1}: \; < 0, \boldsymbol{C_4}, \mathbf{10}, \mathbf{20} >;\\
&\mathbf{FED4^2}: \; < 0, \boldsymbol{C_4}, \mathbf{20}, \mathbf{20} >;\\
&\mathbf{FED5^1}: \; < 0, \boldsymbol{C_5}, \mathbf{10}, \mathbf{20} >;\\
&\mathbf{FED5^2}: \; < 0, \boldsymbol{C_5}, \mathbf{20}, \mathbf{20} >;\\
&\mathbf{FED6^1}: \; < 0, \boldsymbol{C6}, \mathbf{10}, \mathbf{20} >;\\
&\mathbf{FED6^2}: \; < 0, \boldsymbol{C_6}, \mathbf{20}, \mathbf{20} >;\\
&\mathbf{FED7^1}: \; < 0, \boldsymbol{C_7}, \mathbf{10}, \mathbf{20} >;\\
&\mathbf{FED7^2}: \; < 0, \boldsymbol{C_7}, \mathbf{20}, \mathbf{20} >;
\end{aligned}$$

As mentioned before, Flight Dynamics and Environment federates (resp. 5 and 7) have cyclic dependency. Therefore, the Flight Dynamics federate sends **UAV** to the Environment federate to update its position and the Environment federate returns a **UAV** containing air density and wind so that the Flight Dynamics federate can compute the aerodynamic forces for the current step. We solve this cyclic dependence by doubling the rate of Flight Dynamics federate, i.e. four iterations during the hyper period. We assume that each $\boldsymbol{C_i}$ is given by the run-time evaluation tables presented in Section 4.2. The WCET of each task must be equal or greater than the measured computation time. Indeed, we consider the Flight Dynamics federate to run with 100 iterations per calculation, i.e. $\boldsymbol{C_5} = 1$ms. The Sensors, Engines and Actuators federates run with 500 iterations per calculation, i.e. $\boldsymbol{C_3} = \boldsymbol{C_4} = 0.2$ ms and $\boldsymbol{C_6} = 0.4$ ms. The Controller federate computation maximum duration is $\boldsymbol{C_2} = 0.3$ ms. The Joystick and Environment federates run very fast (relevant to microsecond measurement), for formal model we choose $\boldsymbol{C_1} = \boldsymbol{C_7} = 0.1$ ms. Finally, the formal model contains 14 real-time communicating tasks and the corresponding precedence tree is depicted in Figure 9:

$$\begin{aligned}
&\mathbf{FED1^1}: \; < 0, 0.1, 20, 20 >;\\
&\mathbf{FED2^1}: \; < 0, 0.3, 20, 20 >;\\
&\mathbf{FED3^1}: \; < 0, 0.2, 10, 20 >;\\
&\mathbf{FED3^2}: \; < 0, 0.2, 20, 20 >;\\
&\mathbf{FED4^1}: \; < 0, 0.2, 10, 20 >;\\
&\mathbf{FED4^2}: \; < 0, 0.2, 20, 20 >;\\
&\mathbf{FED5^1}: \; < 0, 1, 5, 20 >;\\
&\mathbf{FED5^2}: \; < 0, 1, 10, 20 >;\\
&\mathbf{FED5^3}: \; < 0, 1, 15, 20 >;\\
&\mathbf{FED5^4}: \; < 0, 1, 20, 20 >;\\
&\mathbf{FED6^1}: \; < 0, 0.4, 10, 20 >;\\
&\mathbf{FED6^2}: \; < 0, 0.4, 20, 20 >;\\
&\mathbf{FED7^1}: \; < 0, 0.1, 10, 20 >;\\
&\mathbf{FED7^2}: \; < 0, 0.1, 20, 20 >.
\end{aligned}$$

Each precedence depicted in Figure 9 represents a CERTI communication. According to Tindell and Clark,[41] these precedences are formally described by periodic messages similar to the Liu and Layland task model.[40] Actually, the largest message is the one published by the Flight Dynamics federate (Table 1). This message is composed of 17 double (usually encoded on 8 octets), so global message size is less than 5120 bytes. Table 7 shows that the WCTT of each message through CERTI is 0.3 ms.

This topology can be checked for different WCET (for federate) and WCTT (for CERTI communication on a local host or over a network) by using the CHEDDAR open source scheduling verifier tool.[42] On a single PRISE node, we verify that every WCRT $\boldsymbol{R_i}$ (resp. $\boldsymbol{R_{i,j}}$) is always less than corresponding deadlines $\boldsymbol{D_i}$ (resp. $\boldsymbol{D_{i,j}}$). This formal validation is also validated by run-time verification with high precision RCIM timer (Section 5.2).

4.3. *Time management validation*

The internal CERTI time management algorithm is based on the first generation HLA algorithm, which is based on Null Message algorithm from Chandy and Misra (not explained here).[43] The main shortcoming of this approach for real-time or/and high performance simulation is the communication overhead implied by additional exchanges of NULL messages between all simulators. Although this method avoids deadlock in a conservative federation, it generates some overhead. Nevertheless, this overhead is compensated by the better synchronization that these services enforce between federates. CERTIs algorithm and methods for real-time time management mechanisms can be found in Chaudron et al.,[44] with elements for WCET computation and formal proof discussion. Investigations to these key points are an on-going effort.

5. Practical implementation and experimental results

In the section, we provide the latest updates in CERTI to enforce real-time behavior, such as CPU affinity and priority management. Temporal behaviors are then validated for both data flow and time management models.

5.1. *CERTI real-time API*

Current CERTI version does not provide any service or mechanism to ensure a real-time behavior of a simulation (federation). To manage every part of a federation and to be compliant with formal techniques and scheduling techniques, different methods were recently added to the CERTI API (for the Linux operating system). The new services ensure a correct predictability for CERTI communications (WCTT) and federate computation (WCET).

New functions were first implemented to allow using affinity mechanism. CPU affinity is a scheduler property that *assigns* a process (federate, RTIA, or RTIG) to a given set of CPUs on the system (Figure 10). The Linux scheduler honors the given CPU affinity and the process will not run on any other CPU.

Another interface allows the management of priority for CERTI processes (including federates, RTIAs, and RTIG) (Figure 11). For example, real-time Linux operating systems allow 100 priority levels to run critical tasks (SCHED_PRIORITY). Modification of priority relies on the choice of real-time scheduling algorithms under the POSIX/Linux framework: two scheduling algorithms, namely SCHED_FIFO and SCHED_RR, are intended for time-critical applications that need accurate control over the way in which runnable processes are selected for execution.

Finally, the `mlockall` mechanism is used for the RTIG process, each federate, and their respective RTIA processes in order to disable memory paging into the address space of

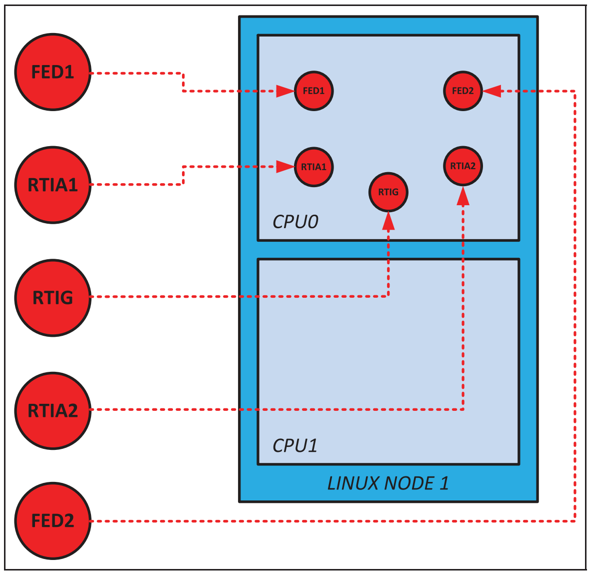

Figure 10. CERTI affinity mechanism illustration.

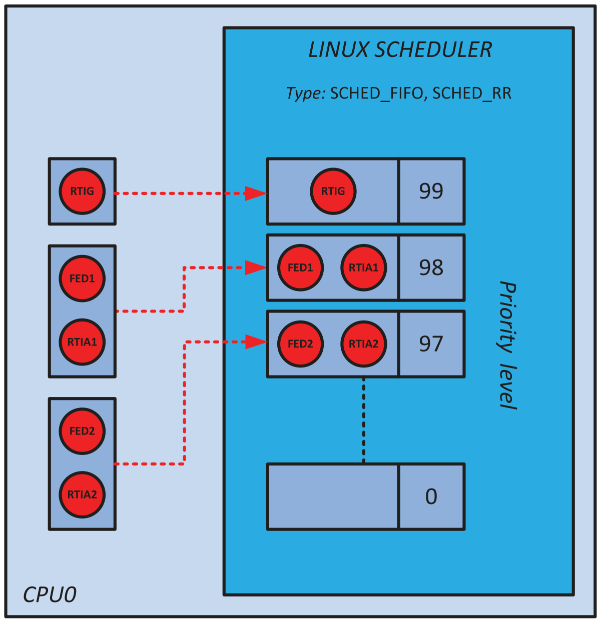

Figure 11. CERTI scheduler configuration mechanism illustration.

the calling process. This includes the pages of the code, data and stack segment, as well as shared libraries, user space and kernel data, shared memory and memory-mapped files. All mapped pages are guaranteed to be resident in RAM when the `mlockall` system call returns successfully. Moreover, they are guaranteed to remain in RAM. Our real-time flight simulator application requires deterministic timing, and, like scheduling, memory paging is one major cause of unexpected program execution delays.

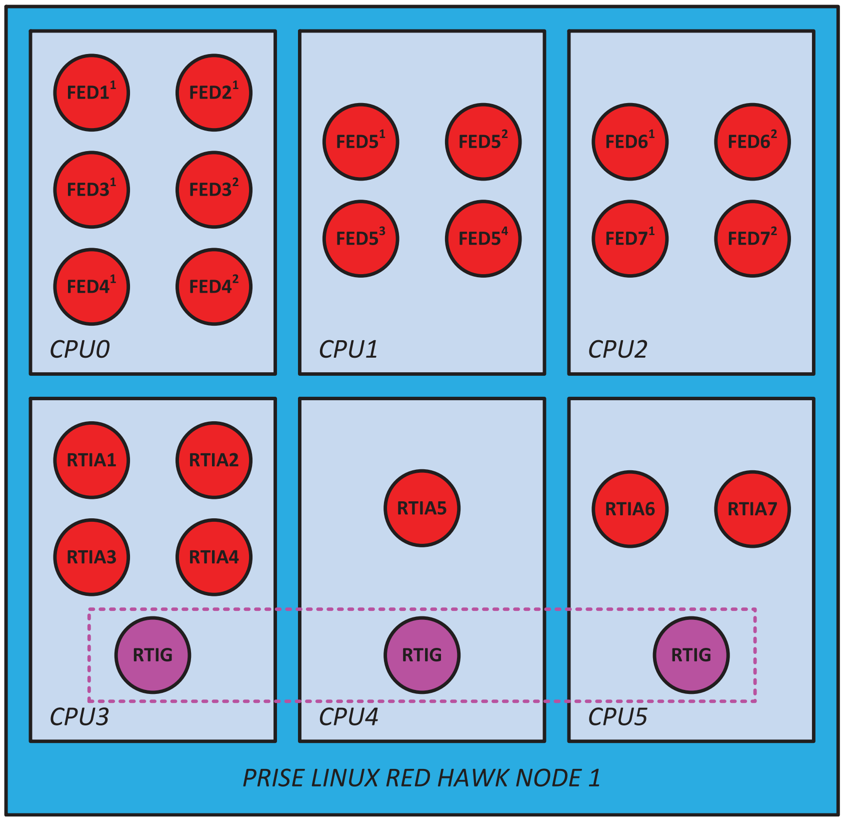

Figure 12. Illustration of CERTI communicating processes topology on a single PRISE node.

We chose a distribution topology on a single multiprocessor Red Hawk node of the PRISE platform. Each federate, its corresponding RTIA and global RTIG are assigned to a given CPU as depicted in Figure 12. RTIG is a particular process that could run on three CPUs with the highest priority on each one to ensure it always runs when it is needed.

Federates (application computing processes) and CERTI communicating processes (both RTIAs and RTIG) are separated in order to validate real-time temporal behavior with Tindell and Clark holistic (Tindell & Clark, 1994). The Deadline Monotonic algorithm associated with simple precedence constraint to assign the priority of each federate subtask on a given CPU is applied. The same method is applied to the corresponding RTIAs. The RTIG is the highest priority task running on CPU 3, 4, and 5. Tindell and Clark's techniques can easily be updated on other distributed topologies if we are able to separate computing processes (federates) and communicating processes (RTIAs, RTIG) on a given architecture.

5.2. *Simulation results*

We first validate the behavior of the data flow and time management models by simulating and comparing the time-responses of the aircraft to an altitude change. From an initial cruising altitude of 10,000 m above sea, we command the aircraft to reach an altitude of 9000 m. The autopilot is turned on and the autothrottle maintains an airspeed of 250 m/s. As shown in Figure 13, the aircraft smoothly reaches the new cruising altitude while maintaining the airspeed reference. The time-responses are strictly identical and similar to the expected behavior provided by an equivalent model running under Matlab/Simulink software.

As shown in precedence tree of Figure 9, an execution period of 20 ms can be split into two cycles of 10 ms. With a refresh rate of 50 Hz, the Joystick and Controller federates run only in the first cycle. Each cycle must respect the 10 ms deadline to satisfy the real-time constraints.

The data flow model measurements are presented in Figure 14 and Table 8. All cycles respect the 10 ms deadline. As expected, cycle 1 is longer than cycle 2; the execution of federates 1 and 2 in cycle 1 generates more messages (**`UAV`** and **`tick`**). The real-time constraints are largely satisfied with an average duration of 2.19 ms and 4.78 ms for cycles 1 and 2. With a very low standard deviation, the behavior of each cycle is very regular.

For the time management model, all computed cycles respect the 10 ms deadline; the global behavior is also very

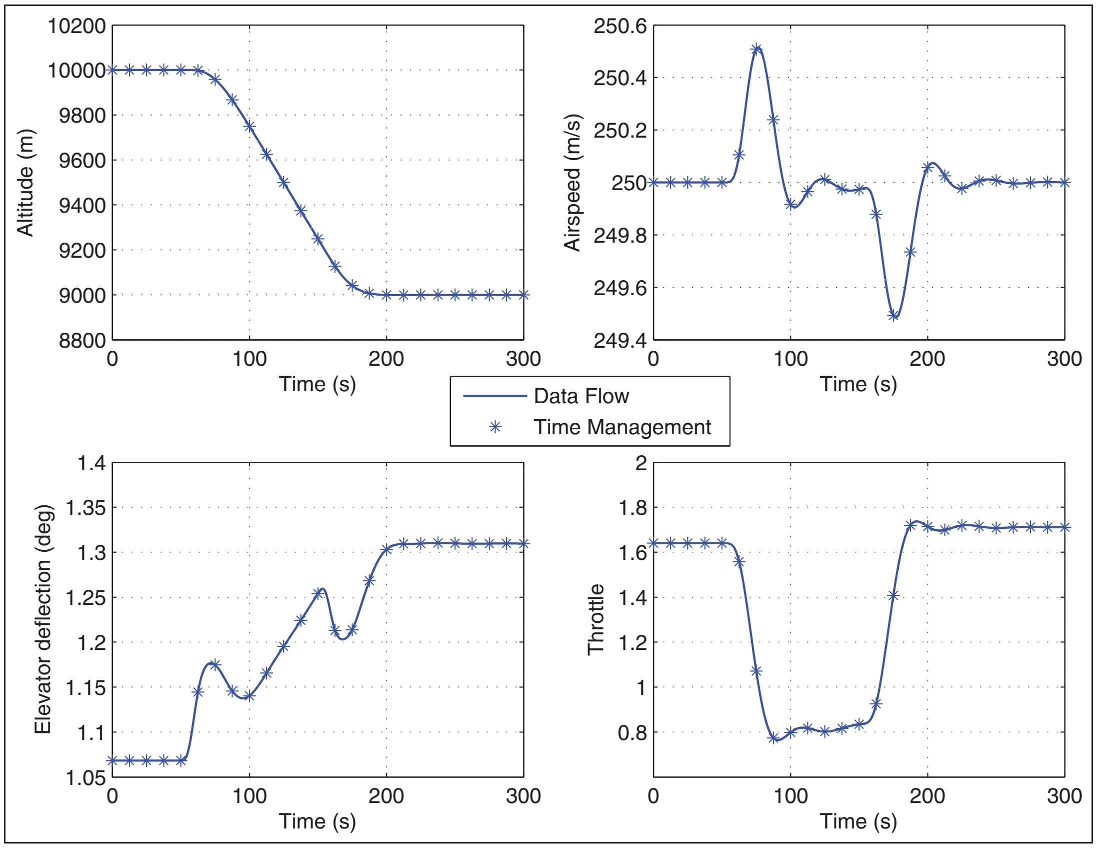

Figure 13. Time-responses for an altitude change.

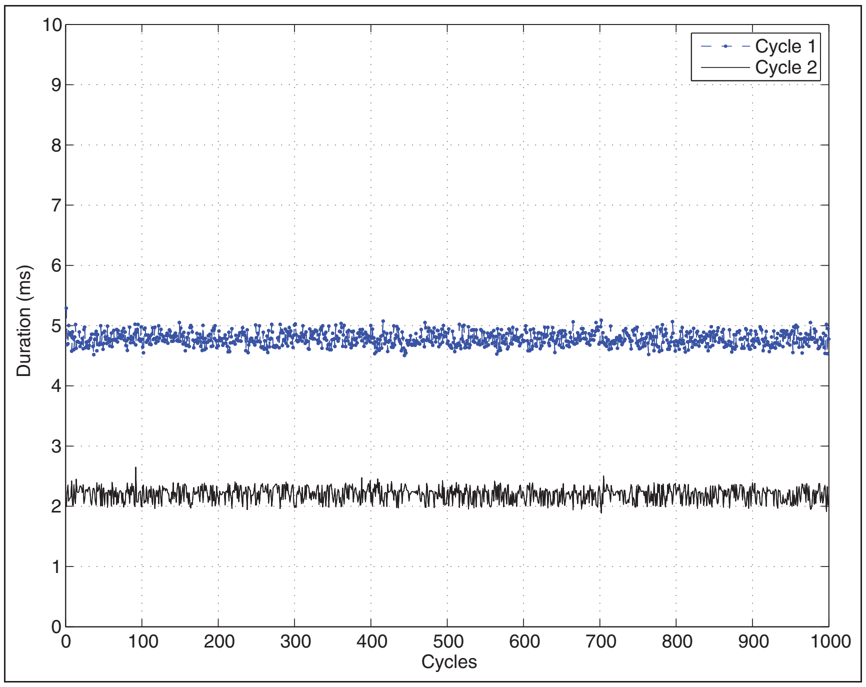

Figure 14. Temporal behavior of time management model.

Table 8. Global measurements for data flow cycles.

Data flow model	Min. (ms)	Mean (ms)	Max. (ms)	Std dev. (ms)
1st cycle	4.50	4.78	5.30	0.11
2nd cycle	1.88	2.19	2.65	0.12

Table 9. Global measurements for time management cycles.

Time management model	Min. (ms)	Mean (ms)	Max. (ms)	Std dev. (ms)
1st cycle	9.13	9.54	9.99	0.11
2nd cycle	6.28	6.55	6.85	0.07

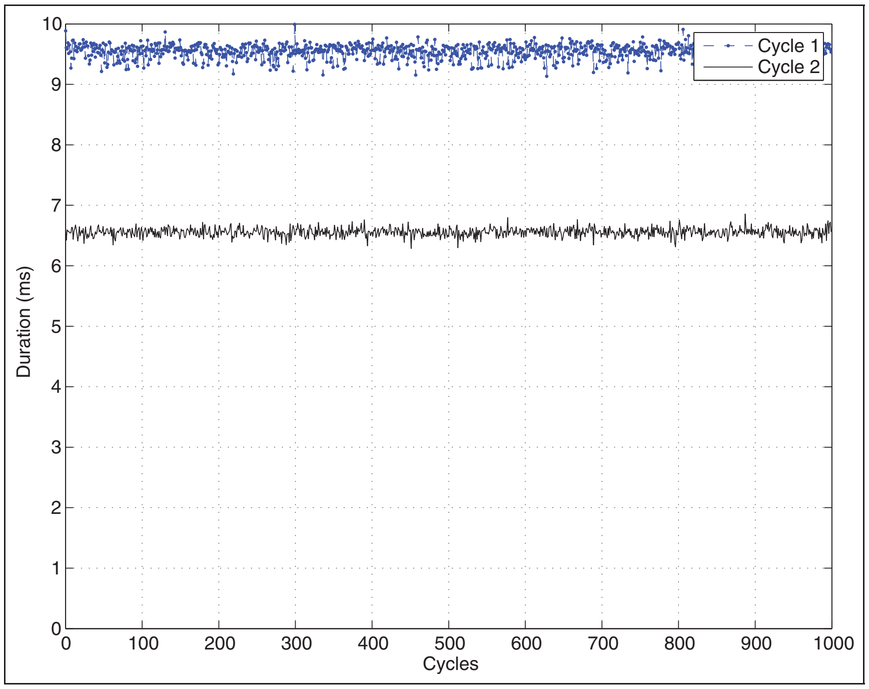

Figure 15. Temporal behavior of the data flow model.

regular (Figure 15 and Table 9). The combination of NER and TAR (HLA services calls) for each federate step seems to generate some overhead. The cycles are longer compared to those of the data flow model, but the real-time specifications are still satisfied.

6. Conclusion and perspectives

The first but complete step of the PRISE project has required the mastering of many skills, from the realistic implementation of an aircraft and its environment to the extension of HLA distributed simulation to real-time.

The real-time analysis has required the modeling of several aspects of a distributed simulation. Different static scheduling and run-time analysis have been studied under different hypotheses (single processor, distributed synchronous processors, distributed asynchronous processors). We have finally developed and maintained many tools to manage the allocation of both federates and CERTI processes over PRISE CPUs; the priority of each process has also been modified to comply with the scheduling technique used. This methodology could still be applied in spite of a change in the federation.

Two run-time models have been introduced. On one hand, the data flow model proved very effective at satisfying the real-time constraints, but adding new federates to the federation would involve a heavier reprogramming task; indeed, one must explicitly program the wait and the reception of newly available data, and as the publication and reception messages are not time-stamped, one must pay greater attention to ensure logical time behavior of the federation. On the other hand, even if the time-management model had longer duty cycles because of the HLA communication messages, it still fulfilled our hard real-time requirements. Moreover, the addition of new federates should not be a heavy computational burden, as it is sufficient to program the adequate time advance request, and RTIG will deliver automatically the attributes with the correct time stamps.

The PRISE platform has become a key element for the study of embedded systems at ISAE. Future projects include the enrichment of the aircraft simulation with new federates and formation flying. Moreover, the hard real-time properties of our architecture could allow the use of real physical actuators, sensors, embedded calculators, or even a real avionic network.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

1. Object Management Group. *Minimum CORBA—joint revised*. OMG document orbos/98-08-04, 1998.
2. Stankovic JA. Misconceptions about real-time computing. *IEEE Comp J* 1988; 21(10): 10–19.
3. Harrisson TH, Levine DL and Schmidt DC. The design and performance of a real-time CORBA event service. In: *Proceedings of the OOPSLA '97 conference*, Atlanta, GA, October 5–9, 1997; pp.184–200.
4. Object Management Group. *Real-time CORBA specifications*. Version 1.2, OMG document formal/05-01-04, 2005.
5. Object Management Group. *Data distribution service for real-time systems*. Version 1.3, OMG Document formal/07-01-01, 2007.
6. The Institute of Electrical and Electronics Engineers (IEEE) Computer Society. IEEE standard for modeling and simulation (M&S) high level architecture (HLA)—Federate interface specification. Simulation Interoperability Standards Committee, 2000.
7. The Institute of Electrical and Electronics Engineers (IEEE) Computer Society. IEEE standard for modeling and simulation (M&S) high level architecture (HLA)—Framework and rules. Simulation Interoperability Standards Committee, 2000.
8. The Institute of Electrical and Electronics Engineers (IEEE) Computer Society. IEEE standard for modeling and simulation (M&S) high level architecture (HLA)—Object model template (OMT) specification. Simulation Interoperability Standards Committee, 2000.
9. Zao H. *HLA Streaming and real-time extensions*. PhD Thesis, School of Information Technology Engineering, University of Ottawa, Ontario, Canada, 2001.
10. Allerton D. *Principles of Flight Simulation*. Reston, VA: AIAA, 2009.
11. Airbus Avionics and Simulation Organisation. Avionics and simulation products. EDYY workshop presentation for engineering schools, 2008.
12. Zheng S, He J, Jin J, et al. DDS based high fidelity flight simulator. In: *Proceedings of the 2009 WASE international conference on information engineering*, Taiyuan, Shanxi, China, July 10–11, 2009, pp.548–551.
13. Lemmers A, Kuiper P and Verhage R. Performance of a component-based flight simulator architecture using the HLA paradigm. In: *Proceedings of the AIAA modeling and simulation technologies conference*, Monterey, CA, August 5–8, 2002 (paper no. AIAA-2002-4476).
14. Huiskamp W, Janssen H and Jense H. An HLA based flight simulation architecture. In: *Proceedings of AIAA modeling and simulation technologies conference*, Denver, CO, August 14–17, 2000 (paper no. AIAA-2000-4401).
15. Kuijpers N, Van Gool P and Jense H. A component architecture for simulator development. In: *Proceedings of the spring simulation interoperability workshop*, Orlando, FL, March 9–13, 1998 (paper no. 98S-SIW-238).
16. Bachinsky S, Noseworthy J and Hodum F. Implementation of the next generation RTI. In: *Proceedings of the spring simulation interoperability workshop*, Orlando, FL, March 23–27, 1999 (paper no. 99S-SIW-118).
17. McLean T, Fujimoto RM and Fitzgibbons B. Middleware for real-time distributed simulations. *Concur Comput Prac Exp* 2004; 16(15): 1483–1501.
18. Zao H and Georganas ND. Architecture proposal for real-time RTI. In: *Proceedings of the Simulation Interoperability Standards Organization (SISO) simulation interoperability workshop*, Orlando, FL, September 9–14, 2001 (paper no. 01F-SIW-025).
19. Boukerche A and Kaiyuan L. A novel approach to real-time RTI based distributed simulation system. In: *Proceedings of the 38th annual symposium on simulation,* San Diego, CA, April 4–6, 2005, pp.267–274.
20. Baietto J. *Real-time Linux: The RedHawk approach*. Concurrent Computer Corporation, White Paper.
21. Gallmeister BO. *POSIX. 4: Programming for the real world*. Sebastopol, CA: O'Reilly & Associates, 1995.
22. Jansen R, Huiskamp W, Boomgaardt J, et al. Real-time scheduling of HLA simulator components. In: *Proceedings of the spring simulation interoperability workshop,* Arlington, VA, April 18–23, 2004 (paper no. 04S-SIW-30).
23. Concurrent Computer Corporation. Real-time clock and interrupt module user's guide. 2001.
24. Noulard E, Rousselot JY and Siron P. CERTI: An open Source RTI, why and how. In: *Proceedings of the spring simulation interoperability workshop*, San Diego, CA, March 23–27, 2009 (paper no. 09S-SIW-015).
25. Noulard E, D'Ausbourg B and Siron P. Running real time distributed simulations under Linux and CERTI. In: *Proceedings of the European simulation interoperability workshop,* Edinburg, Scotland, June 16–19, 2008 (paper no. 08E-SIW-061).
26. Nelson RC. *Flight Stability and Automatic Control*. 2nd edition.New York: WCB/McGraw Hill, 1998.
27. Stevens BL and Lewis FL. *Aircraft Control and Simulation*. 2nd edition: Hoboken, NJ: Wiley, 2003.

28. *Standard US atmosphere*. Washington DC: Government US Printing Office, 1976.
29. Microsoft Flight Simulator X, http://www.microsoft.com/games/flightsimulatorx.30.
30. FlightGear Flight Simulator, http://www.flightgear.org/.
31. X-Plane, http://www.x-plane.com/.
32. Virtual Air, http://sourceforge.net/projects/virtualair/.
33. Arthur KW and Booth KS. Evaluating 3D task performance for fish tank virtual worlds. *ACM Trans Inform Sys* 1993; 11(3): 239–265.
34. Ogata K. *Discrete-time Control Systems*. 2nd edition. Englewood Cliffs, NJ: Prentice Hall, 1994.
35. Fujimoto RM and McLean T. Repeatability in real-time distributed simulation executions. In: *PADS '00: Proceedings of the fourteenth workshop on parallel and distributed simulation*, Bologna, Italy, May 28–31, 2000, pp.23–32.
36. McLean T. Hard real-time simulations using HLA. In: *Proceedings of the Simulation Interoperability Standards Organization (SISO) simulation interoperability workshop*, Orlando, FL, 2001.
37. Chaudron JB, Siron P and Adelantado M. Towards an HLA run-time infrastructure with hard real-time capabilities. In: *Proceedings of the European simulation interoperability workshop*, Ottawa, Canada, Ottawa, Canada, July 12–14, 2010. Available at: oatao.univ-toulouse.fr/4148/1/Siron_4148.pdf.
38. Fujimoto RM. Time management in the high level architecture. *Simulation* 1998; 71(6): 388–400.
39. Phillips RG and Crues EZ. Time management issues and approaches for real time HLA based simulations. In: *Proceedings of the fall simulation interoperability workshop*, Orlando, FL, September 18–23, 2005 (paper no. 05F-SIW-058).
40. Liu CL and Layland JW. Scheduling algorithms for multiprogramming in a hard real-time environment. *J Assoc Comput Machin* 1973; 20(1): 46–61.
41. Tindell K and Clark J. Holistic schedulabitlity for distributed hard real-time systems. *J Microproc Microprog* 1994; 40: 117–134.
42. Singhoff F, Legrand J, Nana L, et al. Cheddar: A flexible real time scheduling framework. In: *Proceedings of the 2004 annual ACM SIGAda international conference on Ada*, Atlanta, GA, November 14–18, 2004, pp.1–8.
43. Chandy KM and Misra J. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans Software Engin* 1979; 5(5): 440–452.
44. Chaudron JB, Siron P and Noulard E. Design and modelling techniques applied to real-time RTI time management. In: *Proceedings of the spring simulation interoperability workshop*, Boston, MA, April 4–8, 2011.