Towards an HLA Run-time Infrastructure with Hard Real-time Capabilities

Martin Adelantado
Pierre Siron
Jean-Baptiste Chaudron
ONERA/DTIM
2 avenue E. Belin
31055 Toulouse Cedex
France
martin.adelantado@onera.fr, pierre.siron@onera.fr, jean-baptiste.chaudron@onera.fr

Pierre Siron
Jean-Baptiste Chaudron
Université de Toulouse, ISAE
10 avenue E. Belin
31055 Toulouse Cedex
France
pierre.siron@isae.fr

Keywords:
High Level Architecture, Distributed Simulation, Real time Systems, Scheduling Algorithms, Real-time Memory Handling.

ABSTRACT: *Our work takes place in the context of the HLA standard and its application in real-time systems context. The HLA standard is inadequate for taking into consideration the different constraints involved in real-time computer systems. Many works have been invested in order to providing real-time capabilities to Run Time Infrastructures (RTI) to run real time simulation. Most of these initiatives focus on major issues including QoS guarantee, Worst Case Transit Time (WCTT) knowledge and scheduling services provided by the underlying operating systems. Even if our ultimate objective is to achieve real-time capabilities for distributed HLA federations executions, this paper describes a preliminary work focusing on achieving hard real-time properties for HLA federations running on a single computer under Linux operating systems. Our paper proposes a novel global bottom up approach for designing real-time Run time Infrastructures and a formal model for validation of uni processor to (then) distributed real-time simulation with CERTI.*

1. Introduction

Modern systems become more and more complex with an increasing number of both components and interactions between them. These different applications often require their services to be delivered with the respect to a given period of time (deadline). This focus is the problematic of real-time system which are defined as those 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 [1]. A real-time application is usually comprised of a set of cooperating tasks which are activated at regular intervals and/or on particular events. They also need a reliable prediction of the worst-case scenario that can arise and to know how to deal with it efficiently and effectively. Apart from satisfying the timing constraints, another important characteristic of real-time systems is the notion of predictability.

Real-time systems are broadly classified into two categories based on the nature of deadline, namely, *hard real-time systems*, in which the consequences of not executing a task before its deadline 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 hard real-time systems are avionic control and nuclear plant control. Telephone switching system and image processing applications are examples for soft real-time systems.

The emergence of computer networks have led to implementation of technologies to control calculations located on different computers which can communicate over a global (or local) network. The uses of these technologies have increased and this has necessitated the development of standards, such as CORBA [2] for client-server paradigm, to respond consistently to problems involved by this distribution

(heterogeneous computers, taking into account the network).

In the field of simulation, the use of network technology has led to an emergence of specific simulations, called distributed simulations which involve several different simulations connected by one or more computer networks. In a distributed simulation, interoperability between distributed components is essential to ensure consistent behavior. In this sense, all distributed players must communicate well and interact following a common framework which is set by a middleware compliant with a standard of distributed simulation, like the High Level Architecture (HLA) in DMSO 1.3 version [3], or in IEEE 1516 version [4].

Middleware in computing terms is used to describe a software agent acting as an intermediary between different distributed processes. This software has to be seen in the domain of interoperability. It is a connectivity software which allows, usually, several applications to run on one or several computers, and to interact across a network (see Figure 1).

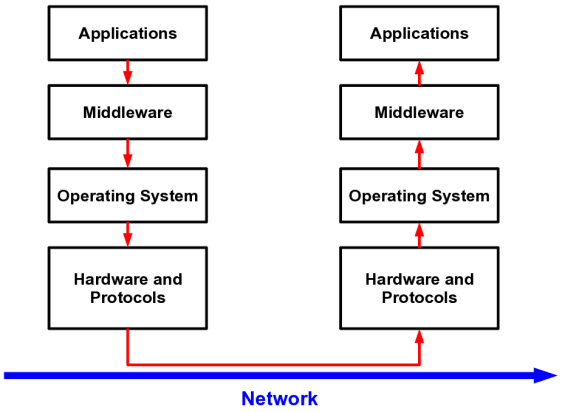

Figure 1: Middleware vision

The middleware involved in CORBA standard is called the ORB (Object Request Broker) and the one implies in HLA standard is named the RTI (Run Time Infrastructure). The RTI is the software implementation of the HLA Interface Specification. It is a middleware for the proper functioning of distributed simulation in accordance with the principles and specifications from HLA standard.

For years, the French Aerospace Laboratory (ONERA) was developing his own middleware RTI compliant with HLA standard called CERTI [5] [6], running under several operating systems including Linux and Windows. This middleware is available from the web site http://www.cert.fr/CERTI. We will use this RTI for our work in order to investigate how we can get real time properties to an HLA real-time simulation.

This paper firstly introduces organizational issues of the problem description focusing on running real-time simulation with HLA. We then discuss of related and current works issues about this problematic. Next, we give a detailed presentation of our work including a formal model to validate real-time simulations. To finish, we present our future works and a conclusion.

2. Problem Description

2.1 Does HLA standard support real-time?

Traditional standards and middleware architectures are not very suitable for supporting real-time constraints. The advantages and the success of these techniques for distributed computing and the emergence of interest for real-time systems implies that research community tries to adapt current middleware standards to include real-times properties. For example, Real-Time CORBA is an enhancement of CORBA. It was designed by the Real-Time Special Interest Group of the Object Management Group (RTSIG-OMG), with participation of several companies in the field of the embedded systems, like Boeing and Objective Interface for example. The Real-Time CORBA specifications [7] allow the management of hardware resources whereas CORBA is an intermediate layer between the operating systems and the applications. One of the key specifications is the end-to-end predictability. To reach this goal, Real-Time CORBA supports fixed priority scheduling. This scheduling method defines static priority levels for each thread. These priorities, despite their value at the initialization, could not be modified during their lifetime.

Works to include real-time specifications and properties to HLA standard are less advanced than CORBA ones. Indeed, HLA does not currently address real-time simulation and HLA compliant simulation could not require any quality of service from the underlying middleware (RTI). Firstly, HLA does not provide interfaces to specify end to end prediction requirement for federate. Secondly, HLA does not allow the management of underlying Operating System(s) in term of priority or resource. Thirdly, in distributed case, HLA only supports two transportation types : the reliable one and the best-effort one, usually encoded with the TCP and UDP network protocols which are not suitable for real-time constraints. These different limitations have crucial impact for real-time simulation systems where the amount and predictability of RTI overhead is an important design factor.

2.2 Different action levels for a real time simulation

The temporal properties of distributed real-time simulation are obtained from a complex combination of the application structure, the HLA middleware used (the standard implementation in a chosen language), the software infrastructure (operating systems and communication protocols) and finally the physical infrastructure (type of computers, type of networks and distribution topology). These different levels imply to answer some relevant questions :

1. ***Hardware level*** : What material should we use? Is it good enough to meet the expectations of the intended application?
2. ***Software level*** :What programming language should I use? Which operating system is best?
3. ***Middleware level*** : What type of middleware we want to use? What is it operating mechanisms involved? What services should it offer?
4. ***Application level*** : What we want to model with the simulation? What kind of tasks are implies in the problem?
5. ***Formal level*** : What formal methods can be applied to verify that the system will perform well according to the requirements of the designer?

2.3 CERTI

We claim that the choice of a RTI (Middleware level) is a very important part for real-time simulation problem because that implies which operating system, which programming language and which hardware could be used for compliance with RTI (Hardware and Software level).

In our approach, to not have to do all work from scratch, we will rely on and extend an existing middleware for real-time purpose. Our choice is to use CERTI Open Source RTI managed and maintained by ONERA team. It is a RTI who is recognizable through its original architecture of communicating processes. It is a distributed system involving two processes, a local one (RTIA) and a global one (RTIG), as well as a library (libRTI) linked with each federate. The CERTI architecture is depicted in Figure 2. Each federate process interacts locally with an RTI Ambassador process (RTIA) through a Unix-domain socket. The RTIA processes exchange messages over the network, in particular with the RTIG process, via TCP (and also UDP) sockets, in order to run the various distributed algorithms associated with the RTI services. This particular architecture will have a significant impact on the evaluation of CERTI for use in a real-time context in particular for scheduling part as we will see in section 4.2 .

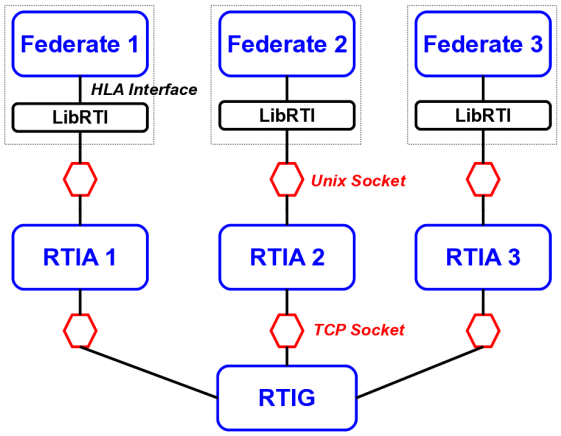

Figure 2: CERTI architecture

2.4 Underlying Software and Hardware

In this paper, for hardware level, we firstly want to validate our approach on a single processor based hardware architecture. Real-time problem in distributed case is much more complex and requires consideration of the communication resource (network messages) in the formal model (see section 6.1) .

For the software level, we choose *Linux Red Hawk,* an operating system compliant with POSIX standard for real-time [8]. It is a Real-Time Operating System (RTOS) and must overcome the uncertainty in time, it is not necessarily faster (more efficient) than a conventional operating system called "time sharing" but must help to add determinism to Os calls from the middleware.

This RTOS have been already used in the simulation domain by TNO laboratory which use this OS to run their own RTI also implemented in C++. Their experiments have been well concluding for the real-time context [9].

3. Previous and Related Works

3.1 Toward periodic federate

The concept of repeatability within real-time simulations has been introduced by Fujimoto and McLean [10] [11]. 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 four phases (see Figure 3): (1) reception phase , (2) a calculation phase, (3) a transmission phase and (4) a slack time phase.

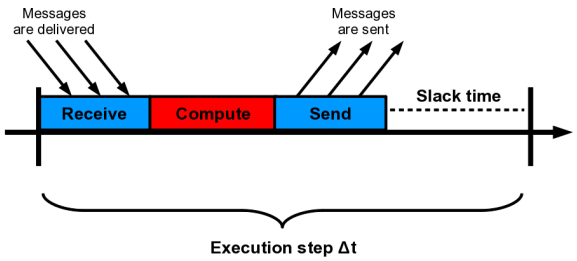

Figure 3: Periodic federate

A similar approach was proposed by ONERA in a collaborative study with the CNES laboratory (Centre National d'Etudes Spatiales) [12]. Each federate involved in the simulation represents an embedded system which performs calculations with periodic cycles. This study has highlighted the necessity of adding a synchronization phase to other phases for each execution step of repeatable federates to be sure to maintain consistency between each cycles (see Figure 4).

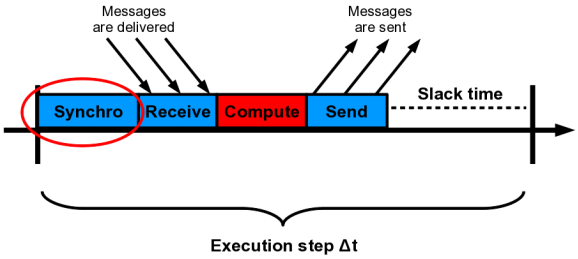

Figure 4: Periodic federate with synchronization phase

Usually this synchronization phase is made (for each federate) by consulting local wall clock time. In Fujimoto and McLean works, it is implicitly made by time management mechanisms in which synchronization and reception phases are made in the same time (see part 3.2). CNES studies present an original synchronization mechanism by sending an interaction from the fastest federate and have also showed the performance of time management services in CERTI for federates with short execution cycles (see section 3.3).

To resume, the synchronization phase can be done either by three different methods :

1. By consulting the hardware clock (Wall clock Time) on a mono-processor system; or using a distribution of hardware clock like RCIM system for distributed applications.
2. The federate which have the high speed cycle sends an interaction to all each others in order to rhythm the execution of all others federates involved in the federation.
3. The use of Time Management HLA mechanisms to ensure messages delivery in all federation and synchronize every federates steps. The time advance can be correlated to an hardware clock to ensure the respect of real time constraints.

This synchronization phase is essential in the distributed context where the different nodes do not have the same timing reference (each using its own local wall-clock time). For present work, we simply consider periodic federates case in which each one synchronizes by consulting the local wall clock time (common for all federate on a uni-processor system).

3.2 The `tick()` service

One of the most significant source of indeterminism in HLA simulation is the **`tick()`** service. This service, although it is not present in the interface specifications for the standard 1.3 (this service is present in IEEE 1516 standard under the name **`EvoqueCallback()`**), is usually still implemented in a RTI. It is a necessary service which allows the RTI to invoke callbacks for the federate using this service.

Both **`tick()`** versions present problems for real-time simulations. In the case of bounded **`tick()`** version (version that accepts two timing parameters **Tmin** and **Tmax**), we know that RTI will release the processor for federate (and therefore the application) for the worst at time **t = Tmax**. However, it can not be sure that this federate will receive the callback before the expected time **Tmax**. That is a matter for the reliability of application (fault tolerance). In other **`tick()`** version (the one without any timing arguments), the federate is sure that the RTI will release processor only when the callback will be expected, however it has no way to know when this callback will happen and therefore presents problem of temporal uncertainty.

Moreover, in CERTI case, **`tick()`** was not blocking. It immediately returned when the RTI could not launch any callback in return, or it returned after having launched a particular callback. Generally, the callback function, launched by the RTI, assigns a value for a flag to mark if the callback is arrived. This is done while the federate enters in a busy waiting loop. This loop generates, on each **`tick()`** call, exchanges of messages between the federate and its RTIA. It generates also useless context switches between these two processes. So the processor resource may be only used by these only two processes : this may seriously disrupt real-time federates.
To avoid such a lock of the processor, this service was re-implemented for CERTI in a blocking mode for CNES studies. In other words, this function returns

only after a callback function has been launched by the RTI. Structure of programming is syntactically the same, but semantically, things are very different because only a few messages are generated and only two context switches are involved. This makes the processor free to be used by many other processes as long as it is not possible to return from a **`tick()`** call.

3.3 Time Management use for real-time

Time management mechanisms provided by HLA are one of the main benefits of this simulation standard [13]. These services could benefit from real-time assurances. Indeed, these mechanisms allow a consistent global time throughout the simulation and could help to ensure respect of deadlines and to keep consistency between the different federates cycles during their executions. 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 principal operation required to implement time management services is determination of the *Lower Bound on Time-Stamp* (LBTS) of any subsequent message that may be later received for a federate. The LBTS value is crucial because any message with time-stamp less than LBTS can be delivered to the federate while still guaranteeing time-stamp order delivery.

ONERA/CNES experiments show that time management seems good for real-time. Indeed, best results are obtained by requiring time management services in both tests cases of study. These services generate some overhead. But, in fact, this overhead is compensated by the better synchronization that these services enforce between federates. This better synchronization between federates reduces latency in data exchanges, reduces the cycle duration and makes the global behavior more regular because jitters are also very reduced. In fact the time advancing algorithm enforces a very good synchronizing for federates that seems to be the best efficient approach. Each federate is a time stepped driven federate and its lookahead is equal to its time step. For ONERA/CNES experiments, the original Chandy and Misra algorithm [14] can be used because it is efficient (with this large positive lookahead) and it is not subject to time creep problem (the number of NULL messages is acceptable and the different time steps are neighbor).

For others kind of applications, other time management algorithms like Samadi's algorithm [15] or Mattern's algorithm [16], should be used to limit the number of NULL messages exchanged between all federates. However, in these cases, computation of LBTS cannot generally be guaranteed to complete within a bounded time because it depends on the participation of all other federates in the execution and transient messages cause an LBTS computation to be aborted and retried. Fujimoto and McLean have modified the LBTS computation in order to respect a bounded time computation for real-time executions [17]. The authors have then proposed an extension for time-stamp assignment in Time Management mechanisms called "*the offset-epoch method*" to increase the efficiency of the original time management algorithm by eliminating these transient messages and computing an LBTS adapted to this new method [18].

3.4 Real-time RTI vision

The RTI is the HLA underlying middleware for interconnecting the various federates to a global federation execution. So, it must provide predictability and ensure that simulation services work in accordance with the timing constraints. In literature, different approaches have been proposed to implement a real-time RTI. These different approaches include:

1. Multi-threaded synchronous process for RTI [19, 20, 21] ;
2. Global scheduling services in RTI [19, 20, 21] ;
3. Real-time Optimized RTI services like time Management from Fujimoto and McLean [19] (see also section 3.3) or Data Distribution Management for Boukerche works [21];
4. Quality of service communication with, for example, RSVP protocols [22] or specific protocols like VRTP [23] ;
5. Use a real-time operating system to allow preemptive priority scheduling.

These different techniques allow a better use of system resources, better scalability when the number of federated using the RTI grows and also a higher reactivity for RTI services.

In our case, a key benefit is to master the implementation of RTI used and thus able to incorporate changes in the source code to ensure temporal predictability of CERTI. The CERTI has, originally, no mechanism for taking into account quality of service and no tools to provide an end to end predictability. In this sense, it does not handle events differently according to a priority and it uses no predictability mechanism whatsoever at the network or the operating system. The initial results, providing some answers about using CERTI in the context of a real-time application, came from ONERA/CNES studies. These studies have shown that CERTI (in its original version) is able to assume multiple real-time federates with short period. The second step is give it some mechanism to ensure end to end predictability and determinism.

3.5 Scheduling theory

One commonly proposed way of constructing a hard real-time system is to build the system from a number of periodic tasks, each assigned static priorities, and dispatched at run-time according to the static priority preemptive scheduling algorithm. The main thrust of *scheduling theory* research with this approach has been to derive an analysis that can bound the behavior of the tasks at run-time.

Original work by Liu and Layland [24] provides *a priori* analysis to determine if a set of periodic tasks would be guaranteed to meet their deadlines. They propose a definition of a periodic task based on timing parameters:
A periodic task is a quadruplet ***<ri , Ci , Di , Pi>*** as,
-***ri*** is the time of initial activation of the task;
-***Ci*** is the worst case execution time;
-***Di*** is the deadline;
-***Pi*** is the period.

Furthermore, in this work each task is assigned a unique priority monotonically with task period, and hence the name rate monotonic scheduling. Current rate monotonic scheduling (RMS) theory assumes that the deadline of a task is equal to the task period. Deadline Monotonic Scheduling (DMS), proposed by Leung and Whitehead [25], is also a static priority scheduling approach (like RMS) where the priority of a task is assigned according to its deadline.

Characteristics of a periodic task such as its period , its deadline or its worst case computation time must be evaluated before the system run. The behavior of the real-time system must be predictable which means that with certain assumptions it should be possible to show at design time that all the timing constraints of the application will be met. Since these timing characteristics of periodic tasks are known, 100% guarantees can be given, at design time, that their timing constraints will be satisfied. The primary criterion in the static scheduling of periodic tasks is predictability, i.e., determining a feasible schedule wherein all tasks meet their timing requirements.

4. Towards a Real Time CERTI

4.1 Need of formal model for validation

To our knowledge, no related work from simulation community has linked any formal model from scheduling theory (uni-processor or distributed) with concepts of distributed simulations. Thus real-time simulations are validated by experiments rather than formal models and schedulability tests. We claim that a formal model compliant with schedulability techniques is essential to validate real-time simulations composed by periodic federates. We choose to first validate our approach on a single processor system by using well known schedulabilty techniques.

4.2 Basic assumptions

An HLA simulation consists of five phases (see Figure 5). We consider that it is not necessary to ensure temporal properties for all HLA services involved in each phase. Our interest is clearly for services implied in simulation loop (Object Management and Time Management). In this paper, we only take into account Object Management services.

1. Creation
⇒ Federation is created by one of the federates
⇒ Others federates join federation
2. Initialization
⇒ Each federate states his intentions of publication and subscription
⇒ Establishment of time management policy (if used)
⇒ Synchronization phase between federates (if used)
⇒ Registration of simulated objects
3. Simulation Loop
⇒ Time Advance (for each federate if used)
⇒ Receipt of updates to subscribed data
⇒ Local computations (for each federate)
⇒ Sent updates to data published
4. Termination
⇒ Remove registered objects
⇒ Disable time management policy (if used)
5. Suppression
⇒ All Federate leave the federation
⇒ Destruction of federation by creator federate

Figure 5: HLA simulation scheme

This paper presents the first results that have been proved for a single processor problem. However, to apply our method, we need to make some basic assumptions:

1. Federate-RTIA pair is taken like one and only real time task;
2. RTIG is the highest priority task, it only runs when it is needed;
3. Tasks therefore share the same timing reference (the CPU one), there is no need to synchronize the different cycles of funds (using time management mechanisms or sending interactions explained above in section 3.1);
4. Tasks communicate via a call to `updateAttributesValues()` service, we assume that the receiver federate is awaiting

`reflectAttributesValues()` callback in reception phase (i.e. the assumption that he calls `tick()` service in its blocking version at the right time);

5. Here we focus on static scheduling algorithms within priorities for each task of the system is calculated before the computation phase of the system (see section 3.5). Indeed, these algorithms are deployed in the case of embedded systems because they have a predictable behavior.

4.3 Periodic task with precedence constraint model

Under the above assumptions, federates simulate periodic processes (like different avionics systems components) and so therefore we can consider them as periodic tasks defined by Liu and Layland or Leung and Whitehead (explained in 3.5). Periodic tasks are time-driven and recur at regular intervals called the period. However in our case, these different tasks communicate by using HLA principles (through calls to the RTI services implies in simulation loop like `updateAttributeValues()` or `sendInteractions()`).These communications could be represented by periodic messages like in figure 6. In this figure, federate 1 runs periodically with a 10ms period and send a message to federate 2 which runs at the same period.

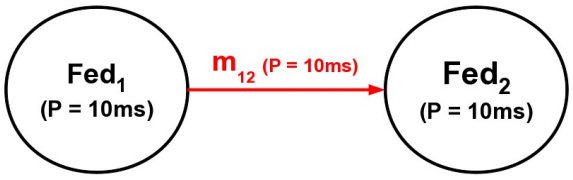

Figure 6: Illustration of a message exchange between two federates

When two tasks of the same period are related by a data-dependency, we can simply impose that the producer always executes before the consumer using Rate Monotonic Analysis (RMA). This corresponds to usual simple precedence constraints which is our interest in present paper. Indeed, these communications between periodic federates (periodic tasks in the model) can be seen as simple precedence constraints requiring that the task issuing the message runs before the receiving task (see Figure 7). These dependencies between tasks can also be solved using techniques from Chetto [26] combined with Deadline monotonic Analysis (DMA). The principle of these alternatives is to make independent tasks by changing temporal parameters from dependent tasks.

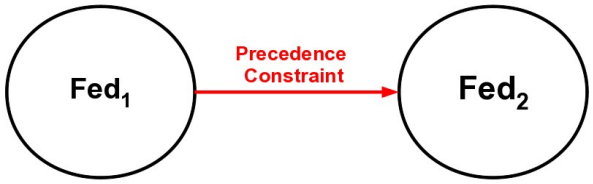

Figure 7: Illustration of precedence constraint between federates

When the message producer task and the message consumer task have different periods, there are several possible communication patterns. For instance, if the producer is 10 times faster than the consumer, the specification can impose that the consumer takes produced data by the second instance out of 10 successive instances. Such communication patterns correspond to more complex extended precedence constraints, which only relate a subset of instances for communicating tasks. To solve these kinds of precedence constraints, we need some others more recent models like new models developed at ONERA [27].

4.4 Evaluate WCET

The calculation of Worst Case Execution Time (WCET) is a key parameter for scheduling because it allows to determine the ***Ci*** parameter value for a task. In our case, a task (Federate-RTIA) consists of three phases: a phase of receiving the data, a calculation phase of the new data, a phase of transmission of this new data. We assume that the time for synchronization phase is null. The WCET of the task will be equal to the sum of the WCET for each phase :

$$Ci = WCET(Receive) + WCET(Compute) + WCET(Send)$$

Calculation of the WCET(Compute) should take into account specific calculations made by the federate and the context switching time of the process (if it would change its priority during the simulation loop). Calculation of values for WCET(Receive) and WCET(Send) must take into account the context switching time between federates and their RTIAs, the time to read and write on different communication sockets (TCP, UDP or Unix sockets) and time for RTIG computation (which is the task with the highest priority and that is only activated on request for federates communications).

5. Illustration

5.1 Original test case

To illustrate our approach we will take a test case from the collaborative work between ONERA and CNES. It

is a federation composed of 4 periodic federates which a real time system composed of 4 tasks (see figure 8):

1. $Fed_1 : <0, 5, 50 , 50> ;$
2. $Fed_2 : <0, 1 , 10 , 10> ;$
3. $Fed_3 : <0, 1 , 10 , 10> ;$
4. $Fed_4 : <0, 1 , 10 , 10> .$

We assume that every ***Ci*** is equal to ten percent of the task's period. That is a pessimistic evaluation which take into account all WCET computing phases (explain in section 4.4). We can use some more important computing time for each task, but in present work, it is just to give an illustration for our proposed methodology.

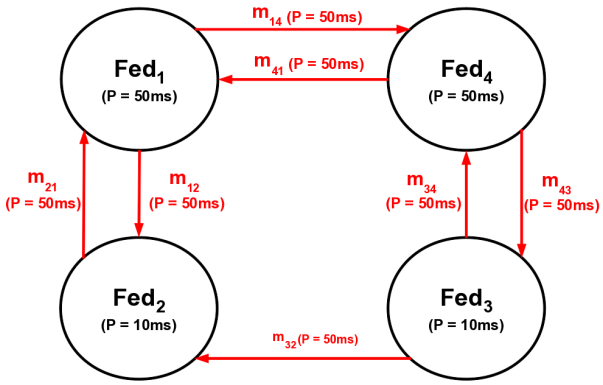

Figure 8: Data exchange in ONERA/CNES federation

In this example, two problems arise to apply techniques for solving simple precedence constraints between tasks:

1. Communicating tasks do not run at the same period. That is not compliant with simple precedence scheme explain in section 4.3.
2. There is a cyclical dependence problem in the graph between tasks Fed_1 and Fed_4 that are interdependent on the same period:

$$P(Fed_4) = P(Fed_1) = P(m_{14}) = P(m_{41})$$

5.2 Unfolding the graph task

To solve the first problem explained before we must unfold the graph of tasks. In this sense, we divide each previous task into a set of subtasks. The period of each subtask is equal to the hyper-period for the set of basic tasks (hyper-period is the *least common multiple* of of all tasks periods). We retain the principle of the critical moment in order to apply the Deadline Monotonic algorithm (the critical moment is when all tasks started at the same time, all ***ri*** are equal to 0). In this test case, hyper-period is 50 milliseconds and we obtain a set of twelve subtasks :

1. $Fed_1^1 : <0, 5, 50 , 50> ;$
2. $Fed_2^1 : <0, 1, 10 , 50> ;$
3. $Fed_2^2 : <0, 1, 20 , 50> ;$
4. $Fed_2^3 : <0, 1, 30 , 50> ;$
5. $Fed_2^4 : <0, 1, 40 , 50> ;$
6. $Fed_2^5 : <0, 1, 50 , 50> ;$
7. $Fed_3^1 : <0, 1, 10 , 50> ;$
8. $Fed_3^2 : <0, 1, 20 , 50> ;$
9. $Fed_3^3 : <0, 1, 30 , 50> ;$
10. $Fed_3^4 : <0, 1, 40 , 50> ;$
11. $Fed_3^5 : <0, 1, 50 , 50> ;$
12. $Fed_4^1 : <0, 5, 50 , 50> .$

To solve the second problem from the interdependence explain in section 5.1, we should spread on the tasks graph for two times hyper-period (2*50 milliseconds = 100 milliseconds). This technique transforms interdependence between tasks Fed_1 and Fed_4 in simple dependence between four subtasks. Indeed, Fed_1^1 is dependent with Fed_4^2 and Fed_1^2 is dependent with Fed_4^1. We obtain a set of 24 subtasks relies by simple precedence constraints (see Figure 9) :

1. $Fed_1^1 : <0, 5, 50 , 100> ;$
2. $Fed_1^2 : <0, 5, 100 , 100> ;$
3. $Fed_2^1 : <0, 1, 10 , 100> ;$
4. $Fed_2^2 : <0, 1, 20 , 100> ;$
5. $Fed_2^3 : <0, 1, 30 , 100> ;$
6. $Fed_2^4 : <0, 1, 40 , 100> ;$
7. $Fed_2^5 : <0, 1, 50 , 100> ;$
8. $Fed_2^6 : <0, 1, 60 , 100> ;$
9. $Fed_2^7 : <0, 1, 70 , 100> ;$
10. $Fed_2^8 : <0, 1, 80 , 100> ;$
11. $Fed_2^9 : <0, 1, 90 , 100> ;$
12. $Fed_2^{10} : <0, 1, 100 , 100> ;$
13. $Fed_3^1 : <0, 1, 10 , 100> ;$
14. $Fed_3^2 : <0, 1, 20 , 100> ;$
15. $Fed_3^3 : <0, 1, 30 , 100> ;$
16. $Fed_3^4 : <0, 1, 40 , 100> ;$
17. $Fed_3^5 : <0, 1, 50 , 100> ;$
18. $Fed_3^6 : <0, 1, 60 , 100> ;$
19. $Fed_3^7 : <0, 1, 70 , 100> ;$
20. $Fed_3^8 : <0, 1, 80 , 100> ;$
21. $Fed_3^9 : <0, 1, 90 , 100> ;$
22. $Fed_3^{10} : <0, 1, 100 , 100> ;$
23. $Fed_4^1 : <0, 5, 50 , 100> ;$
24. $Fed_4^2 : <0, 5, 100 , 100> .$

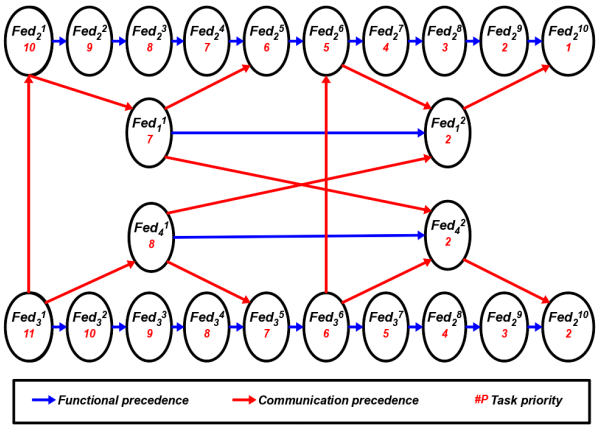

Figure 9: Simple precedence constraints graph for ONERA/CNES federation

This is easily verified using the CHEDDAR open source tool [28] for instance**.** Priorities for the set of subtasks are given in figure 9 (using deadline monotonic analysis combined with Chetto techniques for precedence).

5.3 Additional real-time mechanism for CERTI

The main problem from these techniques is to change priority of CERTI processes for each periodic step of simulation. This change must be apply not only on the federates processes but also on the RTIAs processes. To allow these priorities changes for each federate and its associated RTIA (it is divided into sub-tasks, each with its own priority), we have added to CERTI interface different methods. Modification of priority is also rely on the choice of real-time scheduling algorithms for POSIX/Linux. Two real-time algorithms, SCHED_FIFO and SCHED_RR, are intended for time-critical applications that need precise control over the way in which runnable processes are selected for execution. Only processes with superuser privileges can get a static priority higher than 0 (and lower than 99 for Linux system) and can therefore be scheduled under SCHED_FIFO or SCHED_RR. All scheduling is preemptive: if a process with a higher static priority gets ready to run, the current process will be preempted and returned into its waiting list. In this case, we use the SCHED_FIFO techniques which is compatible with deadline monotonic hypothesis.

Another interface added in CERTI allows to use affinity mechanism. CPU affinity is a scheduler property that *bonds* a process to a given set of CPUs on the system. The Linux scheduler will honor the given CPU affinity and the process will not run any other CPUs. However, we can be sure that on multi-processor platform, we only have one processor for the federation execution to respect our basic scheduling assumptions (uni-processor scheduling).

We also use the *mlockall* mechanism for each federate and its RTIA processes to disables memory paging into the address space of 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 and they are guaranteed to stay in RAM until the pages are unlocked again or until the process terminates or starts another program. Real-time applications require deterministic timing, and, like scheduling, memory paging is one major cause of unexpected program execution delays.

6. Future Trends

6.1 Model formal extension to distributed case

In a distributed real-time system, the timely availability of computational results is guaranteed only if the underlying network supports timely delivery of messages. That is, if the time at which a message is sent and the length of the message, it should be possible to predict the time at which the message will be delivered to the destination. Such a requirement can be satisfied only if the network ensures predictable communication delays. Thus, to ensure the safety of a real-time application implemented on a distributed real-time system, the tasks (as well as the messages) must be properly scheduled on each node (as well as on each communication channel). We should investigate a novel approach to take into account a formal model for tasks (execution units) and also for messages (communication units) from Saad-Bouzefrane's work [29]. All communicating tasks update their timing parameters (including the time of initial activation ***ri*** of the task) by taking into account messages timing parameters. We want to combine this approach with Audsley's algorithm [30] on each node for validation of the global distributed system (Audsley's algorithm is the only optimal algorithm for a set of periodic tasks with different initial activation time). Indeed, we hope this new approach help for validation of a distributed simulation using CERTI.

6.2 Add some deterministic mechanism for CERTI

CERTI is a software developed in C++. This language is usually used to implement some RTI (MAK RTI, DMSO RTI-NG, RTI KIT). Its benefits in term of performance make it a good language for simulation mechanism. However, this language have some gap for real-time like memory allocation which implements algorithm with non bounded time to compute in its

original version. We want to include some predictable allocation techniques and algorithms like the TLSF library [31], which source code is available under GPL license.

7. References

[1] J.A.Stankovic: "Misconceptions about real-time computing", IEEE Computer Journal, 1988.

[2] Object Management Group: "Minimum CORBA - Joint Revised", OMG Document orbos/98-08-04, 1998.

[3] DMSO: "High Level Architecture Specifications" Version 1.3. 1998.

[4] IEEE: "Standard for Modeling and Simulation (M&S) High Level Architecture (HLA). Std 1516. 2001.

[5] B. Bréholée, P. Siron: "CERTI: Evolutions of the ONERA RTI Prototype" Fall Simulation Interoperability Workshop. September 2002.

[6] P. Siron, E. Noulard, J.-Y. Rousselot: "CERTI" www.cert.fr/CERTI. 2008.

[7] Object Management Group: "Real-Time CORBA - Joint Revised Submission", OMG Document orbos/99-02-12, Febuary 1999.

[8] B.O.Gallmeister: "POSIX.4: programming for the real world", O'Reilly & Associates, Inc. 1995.

[9] R.Jansen, W.Huiskamp, J.Boomgaardt, M. Brassé: "Real-time Scheduling of HLA Simulator Components", 2004.

[10] R.M.Fujimoto: "Zero Lookahead And Repeatability In The High Level Architecture" In Proceedings of the 1997 Spring Simulation Interoperability Workshop, 1997.

[11] R.M.Fujimoto, T. McLean: "Repeatability in real-time distributed simulation executions" Proceedings of the fourteenth workshop on Parallel and distributed simulation, 2000.

[12] E.Noulard, B.D'Ausbourg, P.Siron: "Running Real Time Distributed Simulations under Linux and CERTI", European Simulation Interoperability Workshop, 2007.

[13] R.M.Fujimoto: "Time Management in the High Level Architecture" Simulation, 71, pp 388-400. December 1998.

[14] K.M.Chandy, J.Misra: "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs", Software Engineering, IEEE Transactions, 1979.

[15] F.Mattern: "Efficient algorithms for distributed snapshots and global virtual time approximation", Journal of Parallel and Distributed Computing, 1993.

[16] B.Samadi: "Distributed simulation algorithms andd performance analysis", Phd thesis, University of California, Los Angeles, 1985.

[17] T.McLean: "Hard Real-Time Simulations using HLA" Proceedings of the Simulation Interoperability Standards Organization (SISO) Simulation Interoperability Workshop, 2001.

[18] T.McLean, R.Fujimoto : "Predictable Time Management for Real-Time Distributed Simulation" Proceedings of the seventeenth workshop on Parallel and Distributed simulation, 2003.

[19] T.McLean, R.Fujimoto, B.Fitzgibbons: "Middleware for real-time distributed simulationsl", Concurrency and Computation: Practice and Experience, 2004.

[20] H.Zao, N.D.Georganas: "Architecture proposal for Real-Time RTI" Proceedings of the Simulation Interoperability Standards Organization (SISO) Simulation Interoperability Workshop, 2000.

[21] A.Boukerche, L.Kaiyuan: "A Novel Approach to Real-Time RTI Based Distributed Simulation System" Proceedings of the 38th annual Symposium on Simulation, 2005.

[22] H.Zao: "HLA Streaming and Real-Time Extensions" Phd thesis, School of Information Technology Engineering, University of Ottawa, 2001.

[23] D.Bruzman, M.Zyda, K.Watsen, M.Macedonia: "Virtual Reality Transfer Protocol Design rational" Proceedings of the sixth IEEE Workshop on Enabling Technologies, 1997.

[24] C.L. Liu, J.W.Layland: "Scheduling algorithms for multiprogramming in a hard real-time environnment", Journal of the Association for Computing Machinery, 1973.

[25] J. Y. T.Leung, J.Whitehead: "On the complexity of fixed-priority scheduling of periodic, real-time tasks", Journal of Performance Evaluation, 1982.

[26] H.Chetto, M.Silly, T.Bouchentouf: "Dynamic scheduling of real-time tasks under precedence constraints", Real-Time Systems Journal, 1990

[27] J.Forget, F.Boniol, E.Grolleau, D.Lesens, C.Pagetti : "Scheduling Dependent Periodic Tasks Without Synchronization Mechanisms", 16th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS, April 2010.

[28] F.Singhoff, J.Legrand, L.Nana, L.Marcé: "Cheddar : a flexible Real Time Scheduling framework", Ada Letters, 2004.

[29] S.Saad-Bouzefrane: "Etude temporelle des Applications Temps Réel Distribuées à Contraintes Strictes basée sur une Analyse d'Ordonnançabilité", Phd thesis, Université de Poitiers, 1998.

[30] N.C.Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start time", Technical Report, Real-Time Systems Research Group, Dept. of Computer Science, University of York, England, 1991.

[31] M.Masmano, I.Ripoll, A.Crespo, J. Rea: "TLSF: A New Dynamic Memory Allocator for Real-Time Systems", Real-Time Systems, Euromicro Conference on, 2004.

Author Biographies

PIERRE SIRON was graduated from a French High School for Engineers in Computer Science (ENSEEIHT) in 1980, and received his doctorate in 1984. He is currently a Research Engineer at ONERA and he works in parallel and distributed systems. He is leader of the CERTI Project. He is also Professor at the University of Toulouse, ISAE, and the head of the computer science program of the SUPAERO formation (French High School for Engineers in Aerospace Sciences).

MARTIN ADELANTADO was graduated from a French High School for Engineers in Computer Science (ENSEEIHT) in 1979, and received his doctorate in 1981. He is an ONERA (French Aeronautics and Space Research Center) Research Engineer and works at the Information Processing and Modelling Department (DTIM). His fields of interest include simulation, real-time systems and distributed systems.

JEAN-BAPTISTE CHAUDRON received the DEA in Artificial Intelligence (equivalent to fifth years of University studies) in Université Paul Sabatier, Toulouse, France. He pursues a PhD degree in computer science at ONERA/DTIM/SER, Toulouse, France. He is currently working on the extension of the High Level Architecture (HLA) towards the world of real-time.