Coincidence Problem in CPS Simulations: the R-ROSACE Case Study

Henrick Deschamps, Gerlando Cappello, Janette Cardoso, Pierre Siron

Coincidence Problem in CPS Simulations: the R-ROSACE Case Study

Henrick Deschamps and Gerlando Cappello
Modelling and Simulation department
Airbus Operation SAS, Toulouse, France
Email: {firstname.name}@airbus.com

Janette Cardoso and Pierre Siron
Complex systems engineering department
ISAE-SUPAERO, University of Toulouse, France
Email: {firstname.name}@isae.fr

***Abstract*—This paper presents ongoing work on the formalism of Cyber-Physical Systems (CPS) simulations.**

We focus on a distributed simulations architecture for CPS, where the running simulators exist in concurrent and sequential domains. This architecture of simulation allows the expression of structural and behavioral constraints on the simulation. We call scheduling of simulation the temporal organization of the simulators interconnection.

In this paper we address the problem of the interconnected simulations representativity. To do so, we highlight the similarities and differences between task scheduling and simulation scheduling, and then we discuss the constraints expressible over that simulation scheduling. Finally, we illustrate a constraint on simulation scheduling with an extension of the open source case study ROSACE, implemented with CERTI, a compliant High-Level Architecture (HLA) RunTime Infrastructure (RTI). HLA is an IEEE standard for distributed simulation.

***Index Terms*—Aeronautics, CPS, ROSACE, Modelling, Simulation, Scheduling, HLA, CERTI.**

I. Introduction

The aeronautical sector is subject to long and expensive development cycles, especially on avionic products, corresponding to a significant part of the development costs of a civil aircraft. One of the main objectives of the aeronautical industry is to identify and correct system design errors as soon as possible. Using simulation during the development and integration lifecycle of an avionics product is one answer to this need, as long as the simulation sufficiently reflects reality.

Aircraft are systems including controllers tightly interacting with the environment to stabilize the vehicles, which are defined as Cyber-Physical Systems (CPSs) [1]. Thus, aircraft simulations require the interaction of avionics simulations with environment simulations. Due to the complexity of the simulated systems and the simulated environments, as well as the need of incrementally improve systems one by one, simulations are more and more modular, composed of smaller simulations.

We consider that every modular component of the simulation is sufficiently representative. For that, we rely on the existing knowledges in model engineering, these skills can be different if the model represents a physical part or a cyber part. Also, we do not address the problem of parallelization, or of the distribution of large model simulations, our starting point is a set of components produced by experts in model engineering and distributed simulation engineering. With these components, certain abstracted constraints and degrees of freedom are expressed for the integration. Certain very hard constraints for strong coupling during the solving of a set of complex equations will be considered in the future.

With these hypotheses, our problematic is the composition of the existing simulation components that reflect at best the reality. This is a contribution to simulation integration engineering.

Our approach is to propose in section III a formalism supporting the description of the simulation components and the logical model scheduling to be executed. Then we propose in section IV another formalism for the targeted execution architecture. This formalism is generic; section V describes an implementation using HLA-CERTI, but any other implementation could be used. In section VI, we focus on some simulation constraints: Logical latencies and in particular the coincidence constraint. Section VII presents R-ROSACE, a representative aeronautics application that allows to validate our approach. The simulation results illustrate the impact of coincidence problem in the cyber part of the CPS. Conclusions are discussed in section VIII.

II. Related work

The simulation of Cyber-Physical Systems touch many research and engineering[1] domains and is needed at different steps in its design as represented in fig. 1.

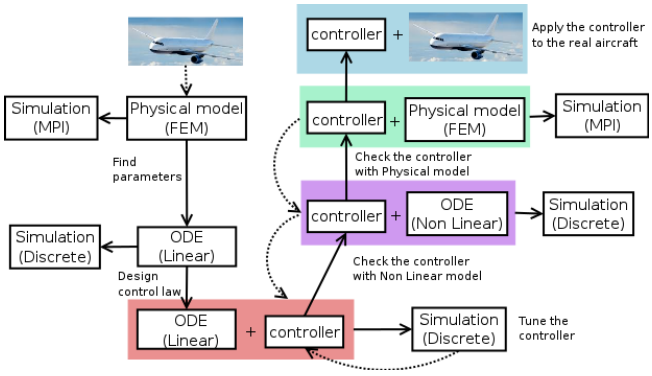


Figure 1: CPS design.

[1]For an interesting discussion about research models and engineering models, see [2].

The simulation focus can be more: - a) on the *physical side*: The lack of analytical solution for complex systems needs a simulation phase whose goal is to approximate the behavior of the complex physical phenomenon as faithfully as possible [3]; or - b) on the *cyber side*: it is necessary to simulate the control to be embedded in the CPS from the earlier stage until the simulation of the computer itself. The simulation of physical systems always involves *discretization* of some type of continuous (or hybrid) model described by differential equations. A good survey about modeling and simulation of CPS is presented in [3]. Concerning the modeling, the Ordinary Differential Equation (ODE) in fig. 1 may be replaced by Partial Differential Equation (PDE) or Differential Algebraic Equation (DAE). As for the simulation, differential equations can be solved by discretizing the time (using, e.g., Runge-Kutta algorithms) or discretizing the state values (using Quantized State System (QSS) algorithms). If the model is quite complex, then parallelization is an interesting solution for speed-up a simulation and/or to avoid model simplification. A key point then is how to decouple models with ODE and DAE, which are strongly sequential by nature. One approach consists in using Transmission Line Modeling (TLM) [4] [5].

Another aspect to point out concerning simulation is that more and more co-simulation is needed, providing interoperability, exchange and reusability of simulations. Two standards exist, with different characteristics: Functional Mock-up Interface (FMI) and High-Level Architecture (HLA). HLA [6] is a IEEE standard originally developed for distributed military simulations and it was not real-time. However, nowadays, HLA is also used in real-time simulation as in [7] [8] [9]. FMI is a tool independent standard to support both model exchange and co-simulation of dynamic models, as for today under the roof of the Modelica Association [10]. The standard allows distributed simulation but it is up to the user to provide the way to do so. Some work, as [11] [12] propose to combine both standards HLA and FMI.

III. Characterization of the model scheduling

A. Model scheduling and task scheduling

This section clarifies the concept of "model scheduling" in CPS simulation.

CPS are systems, often complex, composed of multiple systems and interacting with an environment. A simulation of CPS is a simulation of the cyber part, tightly interacting with the simulation of the physical part. A simulation of CPS comprises the assembly of the cyber part inserted in the simulation framework, and a discretization of the physical part, as illustrated in fig. 2.

These simulations are not necessarily monolithic. A simulation of a system composed of multiple systems can be divided into a simulation of the system components. For instance, a simulation of an aircraft can be divided into the simulation of the engines, the simulation of the structures, the simulation of the rudder, etc... This distribution is particularly adapted to systems where the components can evolve quickly and independently, and when "hardware in the loop" is needed.

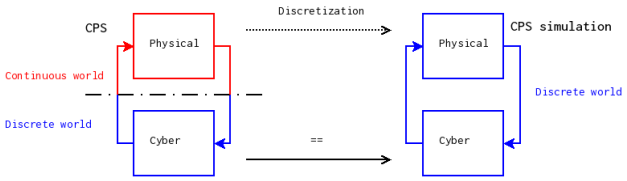


Figure 2: the heterogeneous assembly of components in CPS simulation.

Model scheduling is defined in [13] as the temporal organization of components periodical execution and the synchronization of their interactions. In model scheduling, we call dataflow the flow of data produced by components and consumed by others. A dataflow is either a direct data production and consumption between two close models, or a sequence of data produced and consumed by components between two distant components.

Model scheduling could be brought closer to task scheduling. The periodical execution of component is similar to the scheduling of periodical tasks, except that a component execution instance can be divided in three steps: the data consumption, the computation, and the data production. Furthermore, the notion of the deadline is not refined yet, but we consider it to be the next component iteration for the sake of simplicity. Fig. 3 illustrate the similarity between task and model scheduling.

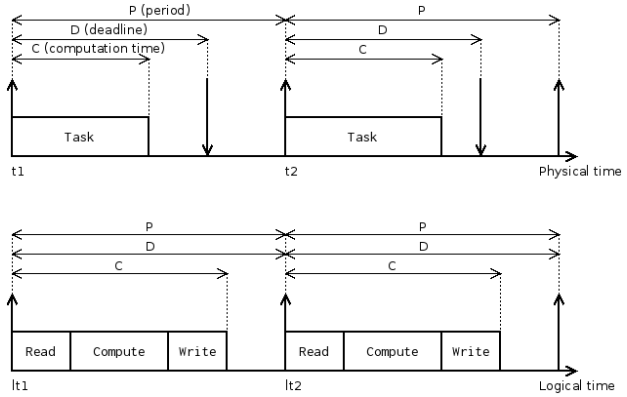


Figure 3: Similarity between task and model scheduling.

More specifically, a component is equivalent to a task, and components can be run sequentially or concurrently by logical processors. An example of execution architecture is given in section IV.

Nevertheless, model scheduling and task scheduling have major difference, described in the two following subsection.

B. Dataflow and precedence

In task scheduling, one of the most major constraints is the precedence constraint. This constraint is defined in [14] as a relation between two activities i and j, by $i \rightarrow j$, meaning j cannot start before i.

A priori, the communication between simulation components can be converted into precedence constraints. For

instance, if a component A produces data consumed by a component B, then A must be executed before B. But the problem is more complex with CPS scheduling.

In CPS systems, there are algebraic loops. With the previous example, A can produce a data for B, and B for A. These algebraic loops lead to two conflicting constraints. The solution is to find a precedence constraint that can be relaxed in an algebraic loop and to break the loop. For instance, if the data produced by B for A can be delayed, then the precedence constraint between B and A is removed.

Nevertheless, in complex systems such as aircraft, algebraic loops are common, and some systems are implied in multiple loops. The dataflows are much more similar to a mesh than a ring, as illustrated in fig. 4.

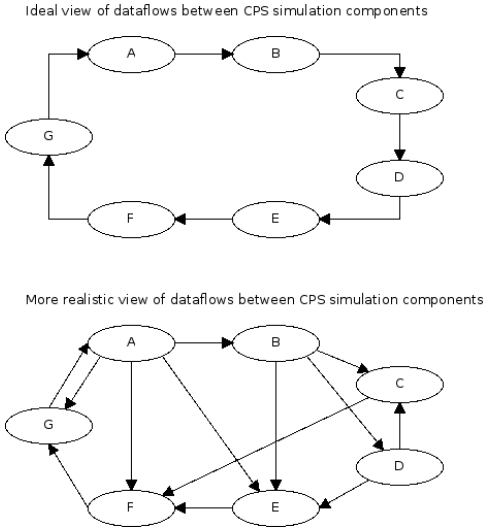


Figure 4: Ideal and more realistic views of dataflows in a CPS.

The relaxation of precedence constraints comes from the simulated systems requirements and environment modelers specialists. Systems, such as avionics, are designed to tolerate certain delays when exchanging data, and environment modelers might design simulation components to tolerate delays. More specifically, minimum and maximum latencies can be associated with data exchanged between two components in a simulation, as well as a long datapath between two components separated by multiple components. As long as those latencies are respected, precedences can be set, and the simulation will still be representative.

IV. An execution simulation architecture

We consider an execution architecture named Simulation Execution Architecture (SEA) and defined in [13]. In the SEA, there is a double level of scheduling, a global scheduler scheduling logical processors using local schedulers, illustrated in fig. 5. Logical processors, running in a concurrent domain, can execute periodic tasks. Tasks are executed sequentially on logical processors. It has to be noted that the global scheduler does not have information about the tasks scheduled by its logical processors, and cannot give an order between two tasks. There are two kinds of communications, intraprocessor and interprocessor communications. Independently of the nature of the communications (shared memory, variables, messages...), those two kinds of communications have different properties:

- Intraprocessor communication allows tasks in the same logical processor exchanging information directly.
- Interprocessor communication allows tasks in different logical processors exchanging information on logical processors synchronization points, under the direction of the global scheduler.

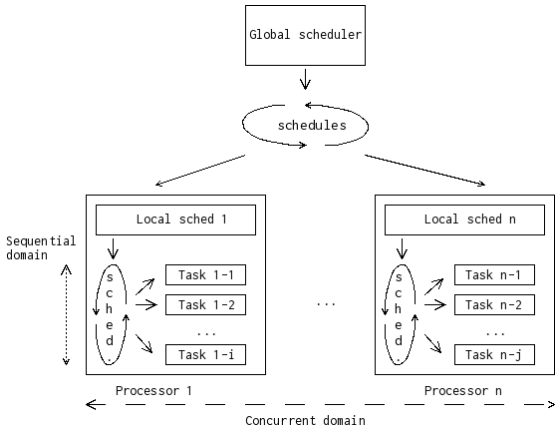


Figure 5: Illustration of the SEA double level of scheduling

In the SEA architecture, the components of a simulation are statically associated to tasks for the simulation execution. After the association, the periods of the tasks are set to the period of the components.

In [8], examples of simulation executions are given, illustrated with HLA. In HLA, a distributed simulation is composed of simulations, called federates. One of these execution models is named time stepped, where a federate, associated with a simulation component, is considered as a periodical task. In this execution model, data produced by simulation component can only be consumed in the future. This imply a cost in term of logical time delay for every dataflow. In this paper, the model of execution is close to this time stepped execution model, but a federate executes multiple simulation components. This execution model is illustrated in fig. 6, linked to simulation components allocation. This model execution optimizes the use of logical time and allows the definition of task sequence, where logical time delay can be null. This notion of null delays does not exist in real time scheduling or most of the distributed simulation.

V. Implementing the SEA with HLA

The SEA can be implemented with different distributed simulations. In this paper, we will focus on HLA/CERTI.

The High-Level Architecture (HLA) is a standard from the IEEE, for software architecture [6]. This standard defines methods and a framework to build global simulations comprised of smaller simulations, the federates. The HLA federates communicate through a RunTime Infrastructure (RTI) [7],

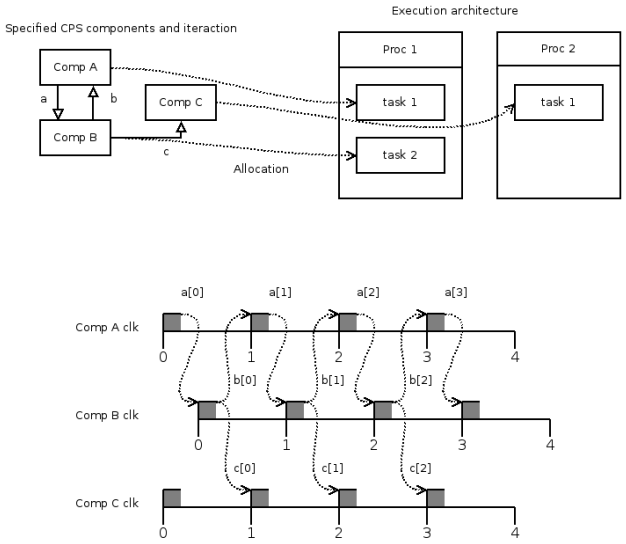


Figure 6: Illustration of different logical time latencies between components.

and using publication/subscription mechanisms to exchange data. In this paper, we consider the CERTI implementation [15] for the RTI.

Model instances in the same federate run sequentially. Model instances in different federates run concurrently.

- The models are components of simulation hardcoded or imported into a federate from a library.
- The logical processors and local schedulers are the federates.
- The global scheduler is composed of the RTI Gateway (RTIG) and RTI Ambassadors (RTIAs).
- The intraprocessor communication is shared memory.
- The interprocessor communication is network communication through RTIG and RTIA.

Fig. 7 illustrates the implementation of the execution architecture of simulation with HLA/CERTI, with a partition of 4 models *a*, *b*, *c* and *d* and three logical processors, considering the use of one computer for the two first logical processors, and one computer for the third one.

The scheduling mechanisms are divided into two levels: The CommonFederate, and the SpecializedFederate inheriting the CommonFederate.

The CommonFederate holds lists of subscribable and publishable objects, bound to a list of attributes, following recommendations from [16]. The CommonFederate advances its logical time depending on the minor frames through HLA services and executing library models depending on their frequencies. SpecializedFederates, inheriting from CommonFederate, initialise the minor frame and the previous list, depending on the ports in and out linked to extraprocessor communications. The intraprocessor communication is a simple shared memory, instantiated with SpecializedFederate attributes.

Fig. 8 depicts the structural bindings of models in SpecializedFederates. Listing 1 depicts how the CommonFederate

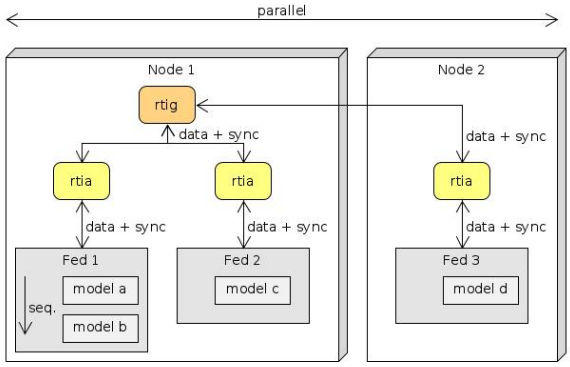


Figure 7: Illustration of the HLA/CERTI execution architecture.

is declared, and how it handles simulation phases. Listing 2 depicts how the SpecializedFederate overrides the simulation step, creating the sequence of models.

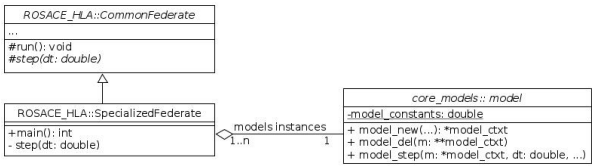


Figure 8: Structure of the binding of models from the library with HLA/CERTI federates.

Listing 1: abstract federate declaration

```
1  structure CommonFederate
2
3      -- Ordered set of models to run periodically.
4      ordered_set<model> models
5
6      -- Variables for intraprocessor communications.
7      map<string name, data value> internal_consumptions,
           internal_productions
8      -- Variables for extraprocessor communications.
9      map<string name, data value> external_consumptions,
           external_productions
10
11     Time t -- Federate current time
12     Time ts -- Federate timestep (GCD of models periods)
13
14     void run():
15         begin
16             ordered_set<void (*)()> phases = {
17                 creation_phase,
18                 initialization_phase,
19                 simulation_loop_phase,
20                 ending_phase,
21                 closing_phase
22             }
23             foreach phase in phases:
24                 phase()
25         end run;
26
27
28     -- Federation creation and joining
29     virtual void creation_phase()
30
```

```
31      -- Publications and subscription declarations,
32      -- binding of external_* with HLA attributes,
33      -- times, variables and models init and synchronization.
34      virtual void initialization_phase()
35
36      -- Loop simulation step until the end of the simulation.
37      virtual void simulation_loop_phase()
38
39      -- Deleting registered objects and leaving federation.
40      -- The last one destroy the federation.
41      virtual void ending_phase()
42
43      -- Function to be specialized by federates.
44      virtual void simulation_step() = 0
45
46      ...
47  end
```

Listing 2: simulation step

```
1   begin
2       -- Waiting for the other federates
3       time_advance_request(t)
4       while not time_advance_grant(t):
5           wait()
6
7       -- Federate inputs retrieving
8       foreach cons in external_consumptions:
9           retrieve_attribute_value(cons, t)
10
11      -- Populating variable for models
12      foreach cons in external_consumptions:
13          internal_consumptions[cons.name] = cons.value
14
15      -- Models sequential run
16      foreach model in models:
17          if t % model.period: -- if the period is coherent
18              model.run(internal_consumptions, &internal_productions)
19
20      -- Populating variable for federate communication
21      foreach prod in internal_productions:
22          external_productions[prod.name] = prod.value
23
24      -- Federate outputs update
25      foreach prod in external_productions:
26          update_attribute_value(prod, t + ts)
27
28      -- Advancing local time
29      t += ts
30
31  end
```

VI. Simulation constraints

CPS simulation representativity and reproducibility requirements can be translated into constraints, either coming from the simulated systems (frequencies, latencies, ...), or the simulation execution. In this section, we will give an insight of the currently identified expressible constraints on CPS simulation scheduling.

In the SEA, interprocessor and intraprocessor communications might have different costs on logical time latencies. Fig. 9 illustrates how different partitions can lead to different logical time latencies. In model C, if data from A and B are compared, then it can lead to a simulation that is not representative of the reality.

Let consider that the two models A and B are sending their logical times to C, that subtracts these logical times. For the first partition, we will have:
$\forall lt, C[lt+1] = A[lt] - B[lt] = lt - lt = 0$
But for the second partition, we will have:
$\forall lt, C[lt+1] = A[lt+1] - B[lt+1] = lt + 1 - lt = 1$. If this subtracter has been designed without considering different delays, the second execution is invalid.

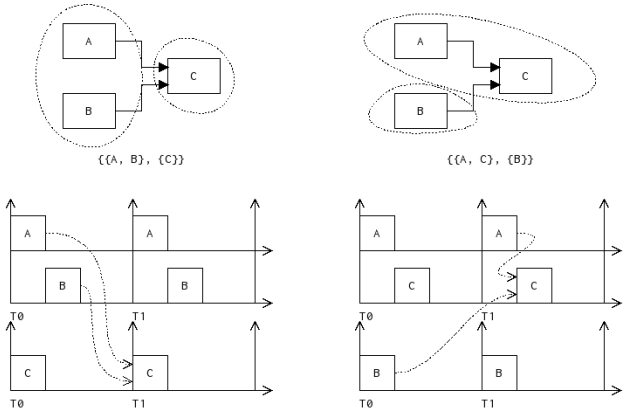


Figure 9: Impact of the partitioning of components on data-flows latencies.

We call the constraint of having same latencies on data-paths the coincidence constraints. This constraint is not limited to simple synchronization of transmission. More generally, when a model iteration produces data, processed by other models on different paths, another model might receive the final production on these paths. If the different latencies on different data-paths are representative of the real CPS, and the final component models a system or physical phenomenon tolerating delay, there is no problem, but most of the time this is not the case, and coincidence constraints have to be identified and respected when partitioning and mapping. Such a case is considered in the R-ROSACE example following.

VII. Example with R-ROSACE

A. Introduction to the case study

The ROSACE (Research Open-Source Avionics and Control Engineering) case study covers different steps from the conception to the implementation of a baseline flight controller. Originally, the ROSACE case study started with the flight controller developed in Matlab/SIMULINK, ending with a multi-periodic controller executing on a multi/many-core target [17]. The case study itself is a longitudinal flight controller, designed to be used as a benchmark and to illustrate the translating of Matlab/SIMULINK specifications to multi-threaded code executing on multi/many-cores.

A major challenge of designing ROSACE controller is the need of interactions between control and software engineers. Control engineering and computer science do not consider the same problems in design, as these two disciplines are technically and culturally separated. For instance, computer science does not consider physical system requirements, such as stability, while control engineering ignores important computing limitations, such as tasks schedulability and network

resources. This issue is particularly prevalent when designing CPS [18], endorsing our willingness to base our study on the ROSACE case study.

The ROSACE case study objective is to validate the real-time aspect of the controller implementation. The following properties are taken into account:

P1 Settling time
P2 Overshoot
P3 Rise time
P4 Steady-state error

An operational scenario is a set of events that includes the interaction of a system with its environment and its users. The ROSACE case study has multiple operational scenarios. The following multiple operational scenarios are taken into account in the original case study:

Case 1 The pilot set a new value to the Vertical Speed (V_z). $V_z : 0m.s^{-1} \rightarrow 2.5m.s^{-1}$

Case 2 The pilot set a new value to the True Airspeed (V_a). $V_a : 230m.s^{-1} \rightarrow 235m.s^{-1}$

Case 3 The pilot wait for t = 50s then set a new Altitude (h). $h : 10000m \rightarrow 11000m$, with $V_z = -2.5m.s^{-1}$

Case 4 The pilot regularly set a new Altitude. $h : 10000m \rightarrow 10500m \rightarrow 11000m \rightarrow 11500m \rightarrow 8000m$, with $V_z = -2.5m.s^{-1}$

All of them are while in cruise flight, at equilibrium.

Case 1 also have the following requirements:

P1 Settling time $V_z \leq 10s$
P2 Overshoot $V_z \leq 10\%$
P3 Rise time $V_z \leq 6s$
P4 Steady-state error $V_z \leq 5\%$

B. Adding redundancy

R-ROSACE is an extension of the open source ROSACE case study, adding redundant controllers [17] [19]. The added redundancy allows us to create multiple datapaths with different latencies.

The case study models and report is available in [20].

R-ROSACE has been implemented with multiple frameworks, including HLA/CERTI, following the architecture of simulation introduced in this paper, the redundancy of controllers is illustrated in fig. 10.

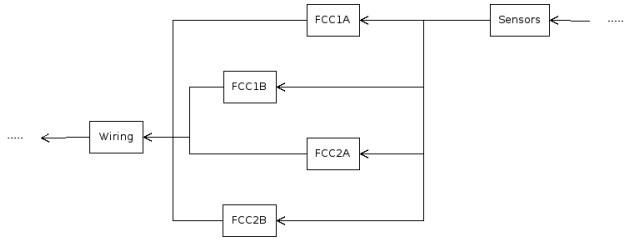


Figure 10: R-ROSACE redundant controllers components, with sensors and wiring as interfaces.

The Flight Control Computers (FCC) correspond to the ROSACE controllers. Three controllers exist in the original case study:

- Vertical Speed controller.
- True Airspeed controller.
- Altitude controller.

We group them in a single component, called FCC, and add a monitoring logic, to use this FCC in command or monitor mode, depending on its usage.

When using multiple FCCs, it must be determined which one will provide commands to the actuators. From the Airbus experience, we know that we have sufficiently information to determine the command, but we have to introduce another component, a wiring model containing switches. This component will simulate the logic of selecting the commands using the relays [21].

In R-ROSACE, we will consider a couple of FCCs, each one is composed of a command, and a monitor. We will also add the wiring model between this couple of FCCs, and the actuators.

Figure 11 illustrates the placement of the components in the redundant ROSACE case study. Moreover, the nature of each component is highlighted (continuous or discrete).

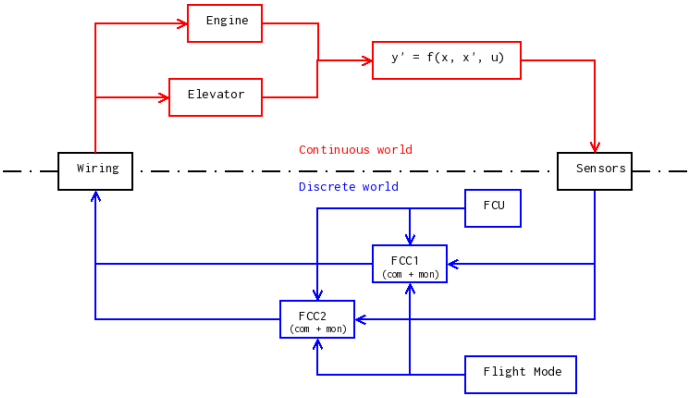


Figure 11: The R-ROSACE case study components view

C. Simulation with different partitions and mappings

All the scenarios were implemented and simulated, however, in the following, we will only present and analyze case 1.

Different partitions of R-ROSACE has been considered, especially the two boundary ones:

- All the components of the simulation are in a single logical processor, following the order of the dataflows (latencies are minimized).
- Every component has its logical processor (latencies are maximized).

Those two boundary partitions show us that there is no problem of logical latencies on R-ROSACE. They are illustrated in figs. 12a, 12b, 13a and 13b, and their predicted executions are illustrated in figs. 14a and 14b

We also consider partitions that should break the coincidence requirements of the added redundancy. To do so, we isolated the FCCs in a dedicated cluster and manipulated their order during the mapping phase. Figs. 12c, 13c, 13d and 13e

illustrate the partitions and mappings, and the predicted executions of the FCCs in their cluster are illustrated in figs. 15. For predicted execution in fig. 15b, the requierment is broken because FCC1A produce a value at t0, consumed by FCC1B at $t0 + period(FCC)$, but the sensors values used by FCC1A and FCC1B came from different sensors model iterations. The data from FCC1A and sensors are not coincident for FCC1B. For predicted execution in fig. 15c, the problem is the same, but with FCC1 and FCC2.

D. Simulation results

Simulation observations are presented for the vertical speed in fig. 16.

P1, **P2**, **P3** and **P4** requirements from the R-ROSACE case study (i.e., the avionics world, not the simulation world) are verified for centralized and distributed simulations (figs. 16a and 16b), as well as isolated FCC when they are mapped in order, or partial order (figs. 16c and 16d).

We predicted that the execution of mapped FCCs in reverse order, fig. 15c, should not be valid, and the observation of the simulation results in fig. 16e confirm this prediction expressed at simulation scheduling-level.

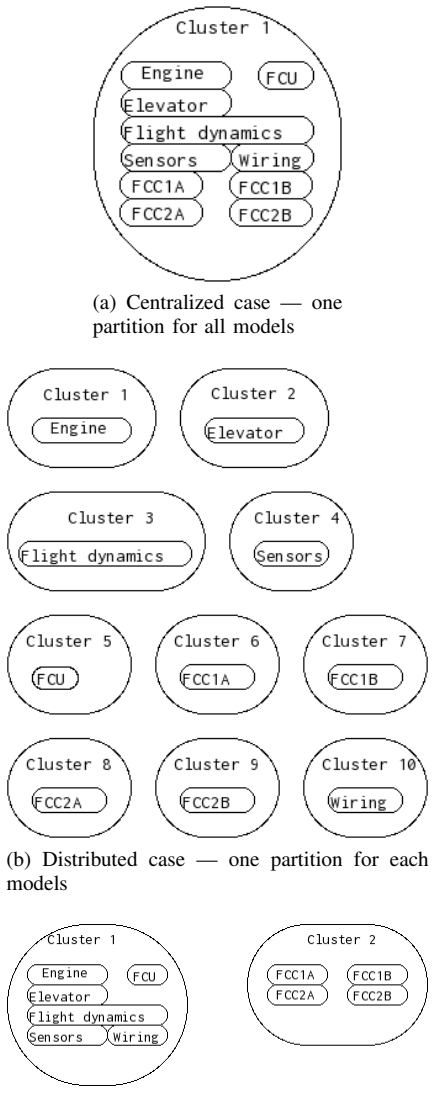
However, we also predicted that fig. 16d should not be valid, as it does not respect the coincidence constraint, but the case study requirements are still met. This particular test case illustrates that simulated system can back simulation failure. The point is that the FCC1 production is invalid, but FCC2 is, and the redundancy mechanism hides the FCC1 problem. Adding others requirements and observing other metrics could have, a posteriori, help identify the invalidity of the simulation, but in the scope of CPSs simulation, this is not sufficient as it covers the problem a posteriori and it must and is cover a priori with the coincidence constraint.

VIII. Conclusion

In [13], we defined a formalism to analyze the scheduling of CPSs simulations. With this formalism, certain temporal requirements can be verified. Nevertheless, a complete set of constraints could not be directly expressed in the formalism.

In this paper, we detailed the implementation of a case study, R-ROSACE, available on [19]. We presented an execution of the case study with HLA/CERTI. We identified in this case study a constraint, expressible on the CPS scheduling during the partition and mapping process. This constraint, the coincidence constraint, is not HLA/CERTI specific, and address every SEA implementation. Other implementation presented in [13] had the same problem. This paper also illustrates how to identify and express new constraints on CPSs simulations schedulings formalized with SLA and SEA.

Synchronization of dataflow is already treated in the literature, nevertheless, the identification and treatment in the scope of CPS simulation is particular as this problem occurs during simulation components integration (i.e., this is totally related to the simulation execution), and the simulation can be invalid, without visible effects on the results.

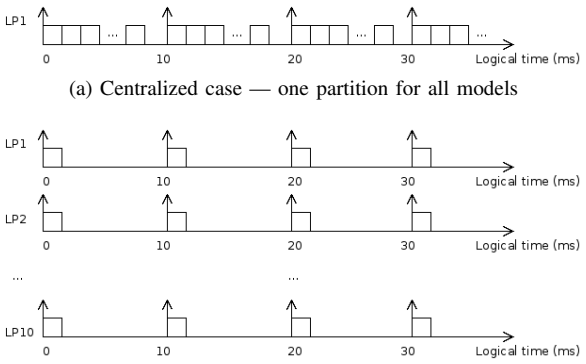


(a) Centralized case — one partition for all models

(b) Distributed case — one partition for each models

(c) FCC standalone case — one partition for FCCs couples

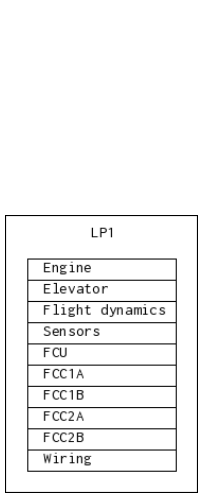
Figure 12: Considered partitions for this paper results presentation



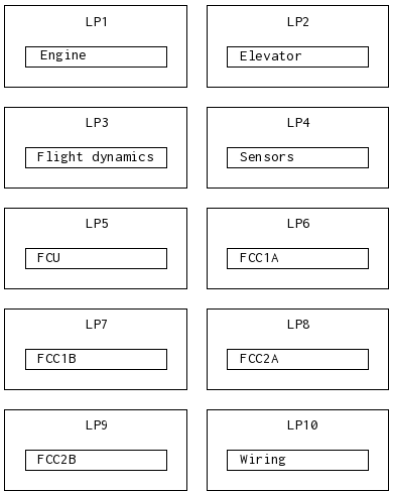
(a) Centralized case — one partition for all models

(b) Distributed case — one partition for each models

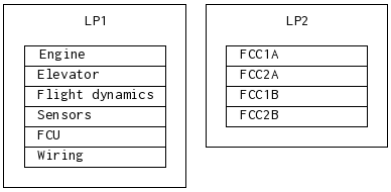
Figure 14: Predicted executions for this paper results presentation, for simple partitions/mappings



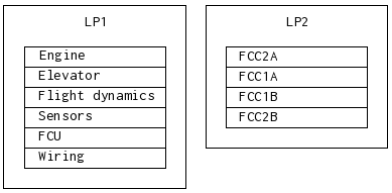
(a) Centralized case — one partition for all models



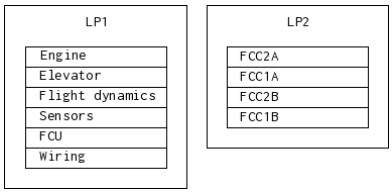
(b) Distributed case — one partition for each models



(c) FCC standalone case — one partition for FCCs couples, com executed before mon

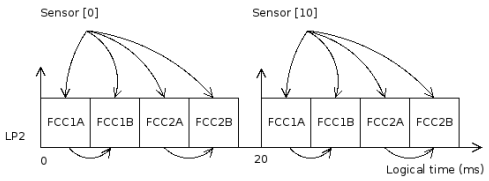


(d) FCC standalone case — one partition for FCCs couples, one couple com before mon, and one mon before com

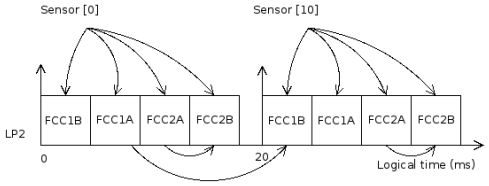


(e) FCC standalone case — one partition for FCCs couples, mon before com

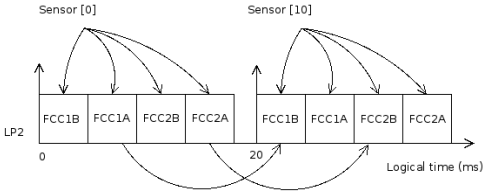
Figure 13: Considered mappings for this paper results presentation



(a) FCC standalone case — one partition for FCCs couples, com executed before mon



(b) FCC standalone case — one partition for FCCs couples, one couple com before mon, and one mon before com



(c) FCC standalone case — one partition for FCCs couples, mon before com

Figure 15: Predicted executions in LP2 for this paper results presentation, for FCC standalone cases

In future work, we will identify more constraints, with the aim of expressing them in a formalism. Furthermore, we will improve our method of representing the SEA, to take into account concepts that were left behind for now, such as the preemption and overrun. This formalism should be applied in the academic and industrial context to prove that the scheduling of a distributed simulation, is valid by design.

The application of the verification of constraint such as the coincidence constraint will be done with allocation algorithm, automatically searching for the best partition and mapping of an SLA on an SEA, and generating a scheduling of simulation.

ACKNOWLEDGEMENT

The work described in this paper is supported through an Industrial Agreement for Research Training — CIFRE — financed by the National Association for Research in Technology (ANRT). This work is also financed and supervised by Airbus, and supervised by the ISAE-SUPAERO, University of Toulouse.

REFERENCES

[1] C. Landauer, "Flight Systems are Cyber-Physical Systems," Nov. 2012.

[2] E. A. Lee, *Plato and the Nerd: The Creative Partnership of Humans and Technology*. MIT Press, 2017.

[3] A. B. K.-E. Feki, "Distributed real-time simulation of numerical models : application to power-train," Ph.D. dissertation, Universite de Grenoble, 2014.

[4] M. Sjölund, R. Braun, P. Fritzson, and P. Krus, "Towards efficient distributed simulation in modelica using transmission line modeling," in *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools :*, 2010.

[5] A. B. Khaled, M. B. Gaiid, N. Pernet, and D. Simon, "Fast multi-core co-simulation of cyber-physical systems: Application to internal combustion engines," *Simulation Modelling Practice and Theory, Elsevier*, pp. 43–48, 2014.

[6] Institute of Electrical and Electronics Engineers and IEEE-SA Standards Board, *IEEE standard for modeling and simulation (M & S) high level architecture (HLA): object model template (OMT) specification*. New York: Institute of Electrical and Electronics Engineers, 2010, oCLC: 682577410.

[7] C. Gervais, J.-B. Chaudron, P. Siron, R. Leconte, and D. Saussié, "Real-time distributed aircraft simulation through HLA," in *Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*. IEEE Computer Society, 2012, pp. 251–254.

[8] J.-B. Chaudron, *Architecture de simulation distribuée temps-réel*. Toulouse, ISAE, Jan. 2012.

[9] J.-B. Chaudron, D. Saussié, P. Siron, and M. Adelantado, "How to solve ODEs in real-time HLA distributed simulation," in *SISO (Simulation Interoperability Standards Organization)*, ORLANDO, United States, Sep. 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01399161

[10] "Functional mock-up interface (fmi)." [Online]. Available: http://fmi-standard.org

[11] H. Neema, J. Gohl, Z. Lattmann, J. Sztipanovits, G. Karsai, S. Neema, T. Bapty, J. Batteh, and H. Tummescheit, "Model-based integration platform for fmi co-simulation and heterogeneous simulations of cyber-physical systems," 2013.

[12] A. Garro and A. Falcone, "On the integration of hla and fmi for supporting interoperability and reusability in distributed simulation," in *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, ser. DEVS '15. San Diego, CA, USA: Society for Computer Simulation International, 2015, pp. 9–16. [Online]. Available: http://dl.acm.org/citation.cfm?id=2872965.2872967

[13] H. Deschamps, P. Siron, J. Cardoso, and G. Cappello, "Toward a formalism to study the scheduling of Cyber-Physical systems simulations," in *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT) (DS-RT'17)*, Rome, Italy, Oct. 2017.

[14] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[15] B. Bréholée and P. Siron, "Certi: Evolutions of the onera rti prototype," in *Fall Simulation Interoperability Workshop*, 2002.

[16] F. Kuhl, J. Dahmann, and R. Weatherly, *Creating computer simulation systems: an introduction to the high level architecture*. Upper Saddle River, NJ: Prentice Hall PTR, 2000.

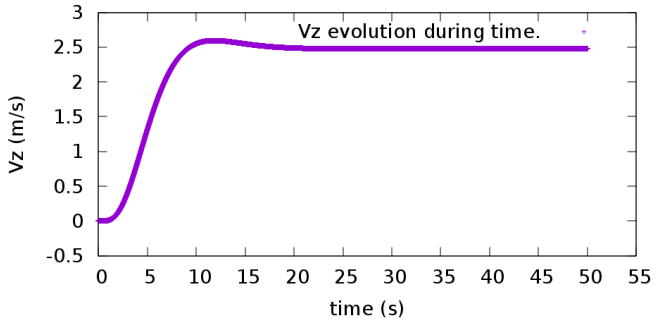
[17] C. Pagetti, D. Saussié, R. Gratia, E. Noulard, and P. Siron, "The ROSACE case study: from Simulink specification to multi/many-core execution," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2014, pp. 309–318.

[18] D. Henriksson and H. Elmqvist, "Cyber-Physical Systems Modeling and Simulation with Modelica," 2011.

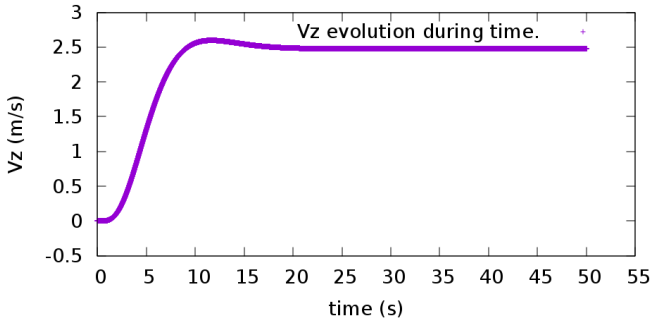
[19] "SchedMCore | Easy MultiCore Scheduling Analysis and Simulation." [Online]. Available: http://sites.onera.fr/schedmcore/

[20] "R-ROSACE case study." [Online]. Available: https://svn.onera.fr/schedmcore/branches/ROSACE_CaseStudy/redundant/

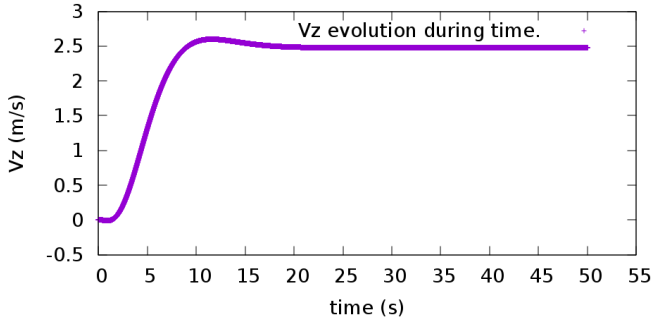
[21] R. Bernard, J.-J. Aubert, P. Bieber, C. Merlini, and S. Metge, "Experiments in model based safety analysis: Flight controls," *IFAC Proceedings Volumes*, vol. 40, no. 6, pp. 43–48, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667015310934



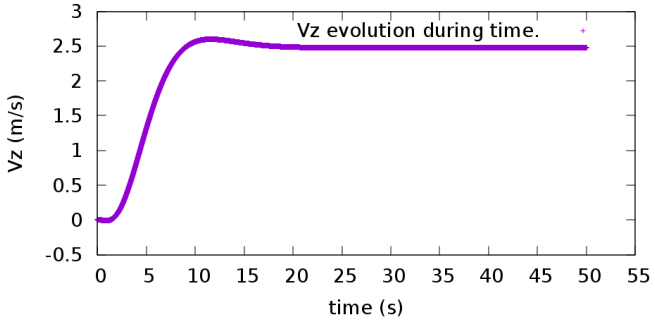
(a) Vertical speed evolution, centralized case — one partition for all models, direct order



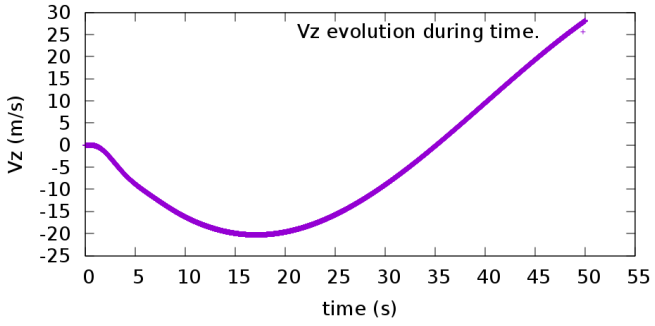
(b) Vertical speed evolution, distributed case — one partition for each model



(c) FCC standalone case — one partition for FCCs couples, com executed before mon



(d) FCC standalone case — one partition for FCCs couples, one couple com before mon, and one mon before com



(e) FCC standalone case — one partition for FCCs couples, mon before com

Figure 16: Vertical speed evolution in R-ROSACE, scenario 1: V_{z_c} set to $2.5m.s^{-1}$