Implementation of a Cyber-Physical Systems simulation components allocation tool

Henrick Deschamps and Gerlando Cappello
Airbus Operation SAS
Modelling and Simulation department
316 Route de Bayonne, 31300 Toulouse, France
Email: {firstname.name}@airbus.com

Janette Cardoso and Pierre Siron
ISAE-SUPAERO, University of Toulouse
10 avenue Édouard Belin, 31055 Toulouse, France
Email: {firstname.name}@isae-supaero.fr

KEYWORDS
Aeronautics, CPS, Modelling, Simulation, Allocation, Scheduling, HLA, CERTI.

ABSTRACT

This paper presents ongoing work on the formalism of Cyber-Physical Systems (CPS) simulations. In this paper, we consider distributed CPS simulations, for which there are strong constraints on the interaction of simulation components. In our previous work, we suggested a method to estimate a simulation scheduling on a given architecture and to verify constraints *a priori*. Trying to integrate these components manually is very time-consuming, and can lead to mistakes. We introduce a tool to make an automatic allocation and scheduling of CPS simulation. We base this on the real-time scheduling literature and adopt a heuristic-based approach, adapted to our framework. We then present a complete case study, including simulations and the physical architecture of the simulations, and we illustrate the generation of the scheduling with different heuristics.

Introduction

A Cyber-Physical System (CPS) is a feedback system comprised of communicating real-time systems and humans or environment in the loop.
Aircraft, Airbus and ISAE-SUPAERO activity area, is a type of CPS, where pilots, avionics, aircraft surfaces and the aircraft environment are tightly interacting through control loops to stabilize the vehicle.
CPS design work-flow can integrate simulation phases; this is particularly true in the aeronautical sector, different steps of the CPS design are illustrated in fig. 1. Development cycles are long and expensive on avionics product. They can be shortened using simulation during development and integration life-cycle.
Due to the complexity of the simulated systems and the simulated environment, as well as the need to incrementally improve systems, simulations are more and more modular. In our method, we consider that every modular component of the simulation is sufficiently representative. Furthermore, in our cases study, we only considered CPS with the physical part discretized, with

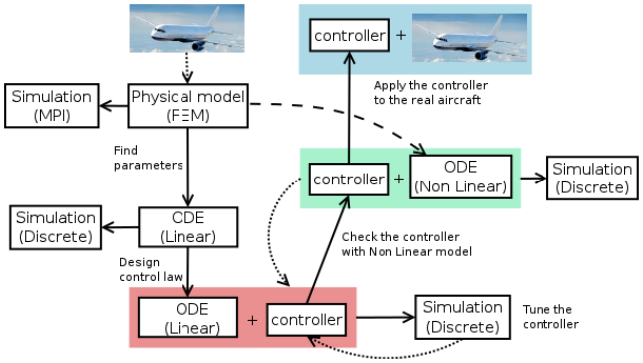

Figure 1: Steps of a CPS design and implementation

linear and non-linear ODEs, as depicted in fig. 2. For that, we rely on the existing skills of 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, specific abstracted constraints and degrees of freedom are expressed for the integration.

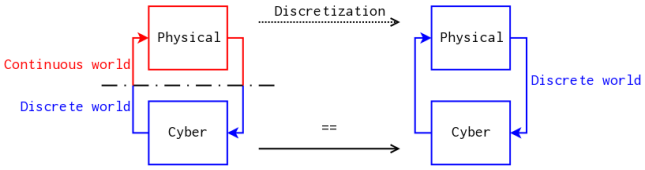

Figure 2: CPS discretization considered in our study

With these hypotheses, our problem is the composition of the existing simulation components that respect the abstracted constraints. In (Deschamps et al. 2017) and (Deschamps et al. 2018), a formalism for expressing the CPS simulation has been introduced. With small examples, temporal constraints can be verified on a CPS simulation, manually. Nevertheless, in an industrial context, a simulation of CPS can be composed of a substantial number of components and constraints. Our problem is the specification and implementation of tools that can automate the verification of constraints and the

generation of scheduling from the description of a CPS simulation.
In the following, after the related work, an overview of the CPS scheduling formalism is presented. Then we present our framework and will focus on the *Allocation function*, with an illustration using a concrete case-study. Finally, we will give concluding remarks and our future work.

Related work

Latterly, progress has been made in the study of accurate CPS simulation using modular blocks.
Numerous studies on CPS simulation in which the emphasis is on fidelity or verification uses the Functional Mock-up Interface (FMI) standard (Blockwitz et al. 2012). The FMI standard defines an implementation interface, the Functional Mock-up Unit (FMU), allowing the integration of heterogeneous models. This standard is therefore ideally suited for CPS simulation. An instance of FMU, also called a slave, can have a specific solver and be executed standalone or can wait for an execution directed by a master. The master algorithm also synchronizes FMUs, following specific synchronization points, and manages communication between FMUs.
In (Sadvandi et al. 2018), the authors present a CPS simulation platform capable of executing simulation loops with models, software, or hardware in the loop. These platform objectives are to meet the same needs as those presented in the introduction, namely the reduction of development costs through early validation through simulation. In this study, the constraints inherent to the CPS are highlighted, particularly at the level of the temporal behavior of the simulation integration, to validate a whole control system. FMI was chosen to meet the interaction needs of simulation components; however, the configuration of execution and time cycle modes is left to simulation designers. The scheduling of a simulation depends mostly on synchronization, as well as solvers. This is not part of the FMI standard, and the handling of fidelity problems due to interactions is left to simulation designers.
(Saidi et al. 2016) highlights the problem of correct data exchange between models, due for example to dependencies between models, using the FMI standard. In this study, the evolution of time and the way data exchange occurs is highlighted, and the focus is on the effectiveness of synchronizations. The authors propose solutions and tools to reduce execution times, based on acyclic graphs, and by proposing allocation heuristics, respecting specific constraints, in particular, interdependence between models while trying to accelerate simulations. This approach is similar to the SynDEx approach (Lavarenne et al. 1991). Our study also deals with the heuristic allocation proposal, but we do not use acyclic graphs. However, our targeted use cases for testing are frequently in real time. Simulation acceleration is not relevant.
In the position paper (Zheng and Julien 2015), the authors propose to check a CPS by observing its behavior during execution. The paper raises the problem of the current lack of control over the fidelity of the temporal behavior for a CPS simulation. It is reminded in this paper that the setting of synchronizations with FMI is left to simulation integrators, which is complicated and error-prone. The authors also focus on Modelica and assert that Modelica is not applicable to the heterogeneous models that constitute a CPS. However, (Henriksson and Elmqvist 2011) contains an example of CPS with heterogeneous models simulated with Modelica, but in this last study, the integration with tools allowing the execution time analysis is left to the future work.
In this paper, the components and architecture used are not linked to FMI, to avoid unnecessary dependencies. Nevertheless, a full description of the components is available in (Deschamps et al. 2017), and the "C-function" aspect, as well as data port, are very similar to FMI. The objective being to ensure the accuracy of the temporal behavior of a CPS simulation, as well as the expression of temporal requirements, and porting solution to FMI will not be considered before the full proof of concept.

The Simulation Distributed Architecture Model

This study is part of work on CPS simulation scheduling. The following briefly presents the method developed in (Deschamps et al. 2017) and (Deschamps et al. 2018). The method relies on the expression in a first formalism of the simulation logical architecture, sLa, and in a second formalism of the simulation execution architecture, sEa. The sLa allocation on the sEa generates the scheduling. Implementation details are added.

The Simulation Logical Architecture, sLa

The sLa allows the expression of structural and behavioral constraints of the simulation, abstracting its execution. This formalism is primarily inspired by the DEVS formalism (Zeigler et al. 2000), using components, couplings, states and internal/external functions.
The components have information about input and output ports, their refreshing frequency, the simulated system or physical phenomena states, the initial states. Internal or transition function, periodically called, update component states, and output functions update component outputs.
Channels are used to connect one component output port to another component input port.
Requirements that can be described in the sLa, for now, are:

- Latency requirements – The most common in aeronautical sector. Due to real-time data exchange

delay between avionics, data-path loops in logical time can be broken in a simulation.

- Coincidence requirements – Some simulated systems might need to receive data from other components at the same logical time to be representative.

- Affinity requirements – From simulation integrators expertise, some simulation components might operate better if in the same logical processor.

- Precedence requirements – From simulation integrators expertise, some simulation components might require to be scheduled in a specific order.

An illustration of sLa components and channels is presented in figure 3.

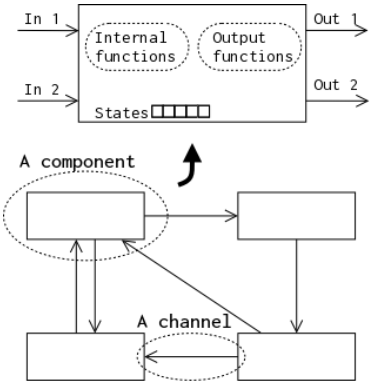

Figure 3: sLa components and channels

The Simulation Execution Architecture, sEa

The sEa is a formalism allowing the estimation of simulation execution parameters. This is a simple Architecture Description Language (ADL), in the form of a generic off-line partitioned scheduler. An illustration of the generic sEa is available in figure 4.
The sEa can represent multiple simulators, with very different execution and communication methods, as long as the following are respected:

- A global scheduler schedules logical processors, in the concurrent domain.

- Each logical processor has a similar local scheduler that schedules its tasks, in the sequential domain.

- Communication between tasks in a same logical processor and tasks in different logical processors might be different (various media, synchronizations, or latencies for instance).

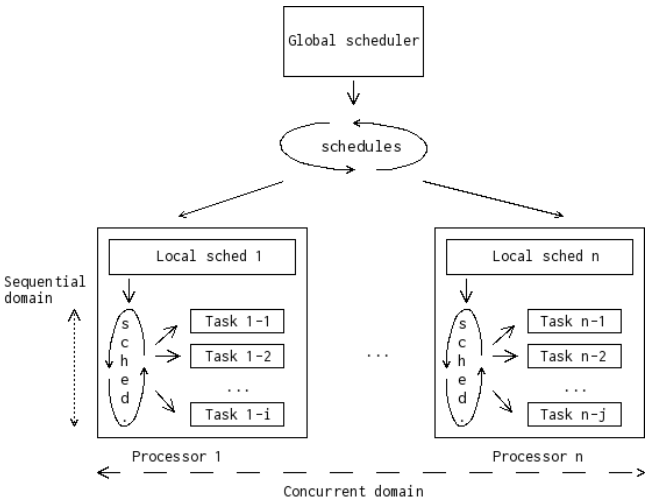

Figure 4: An sEa, with its double level of scheduling

The allocation function

The allocation function partitions the sLa components and maps them to the sEa tasks on logical processors. Knowing the sEa behavior, the resulting scheduling can be determined, and sLa requirements verified.
The allocation method is illustrated in figure 5.
The partition and mapping of components are inspired from (David et al. 1992). Partitioning consists in grouping and splitting the components into different sets. Mapping consists in choosing an order for all the sets of the chosen partition.

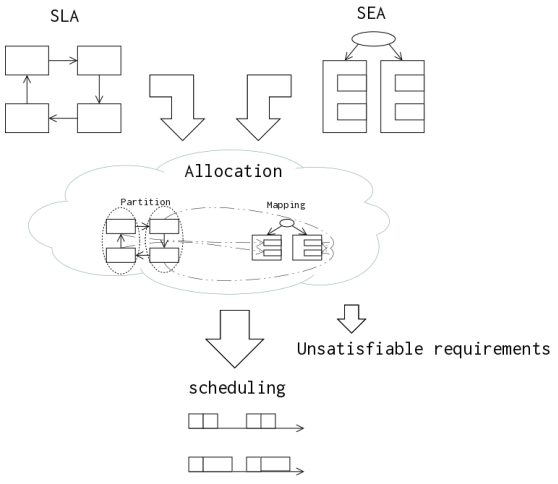

Figure 5: Determination of scheduling based on allocation of sLa on sEa

Implementation of the allocation tool

The allocation tool implements the previous formalisms and allocation. In this study, we will focus on the allocation of sLa on sEa use case. Other use cases are,

for instance, validation of existing scheduling, or verification of sLa or sEa syntax.
The allocation tool is represented in figure 6.

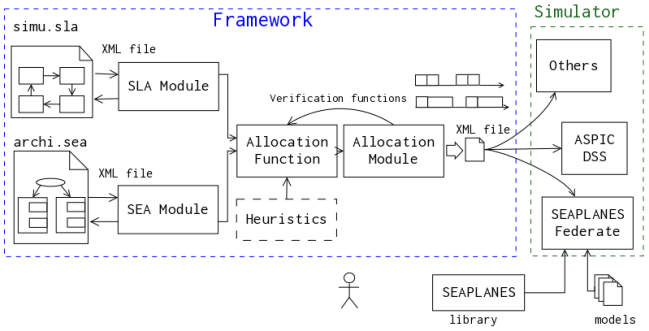

Figure 6: Allocation tool modules

The allocation tool modules

In the allocation use case, the **sLa module** and the **sEa module** read sLa and sEa respectively as XML files, and produce objects that the **allocation function** can use.
The **allocation module** contains the classes that allow the representation of scheduling, and the writing of this scheduling in an XML file. This module also contains the functions to verify the sLa requirements depending on the sEa implementation.
The problem of allocating tasks offline on a partitioned scheduler is known to be equivalent to the bin packing problem, which is NP-hard (Dhall and Liu 1978). The **allocation function** uses a **heuristic** to partition and map sLa components on sEa tasks. This **allocation function** uses the **allocation module** to create the allocation object and uses allocation object methods to verify the sLa requirements. **Heuristics** implementations are independent of the allocation function. As for now, four heuristics are implemented. The four heuristics are variations of the most known heuristics used to solve the bin packing problem. The difference with the classical heuristics being that a logical processor can schedule a component if the utilization allows it (regarding the logical processor's components' time budgets and periods), but also if the requirements are valid. If there is at least one logical processor left that is not full, but no allocation without breaking requirements, then a new allocation search is executed, considering the deletion of requirements, from the least to the most important ones. Considering an ordered set of components, and an ordered set of logical processors:

- First-fit – Each component is allocated to the first logical processor in the set. If this logical processor cannot schedule it, then the component is allocated to the next one, and so on. Logical processor set can be manipulated as a list.
- Next-fit – Same as First-fit, but the search of logical processor starts at the one following the last allocated. Logical processor set can be manipulated as a circular buffer.
- Best-fit – Search for the logical processor starts from the least, up to the most utilized one. Logical processor set can be manipulated as a binary heap, indexed by utilization.
- Worst-fit – Search for the logical processor starts from the most, down to the least utilized one. Logical processor set can be manipulated as a binary heap, indexed by utilization, in reverse order.

Compatible simulators

As long as a simulator can be described with the off-line partitioned scheduler, this simulator is compatible with the allocation results. Different simulators are already compatible with the allocation results.
ASPIC (*Atelier de Simulation Pour l'Intégration et la Conception*, in English, simulation framework for integration and design) and DSS (Distributed Simulation Scheduler) at Airbus. sEaPLANES (sEa Partition-based Logical processor Allocator Node with Extensible inline Scheduler) at ISAE-SUPAERO.
DSS is a framework for scheduling AP2633 models: Airbus simulation model containing entry points, state machine, and variables needed for scheduling. This framework schedules logical processors in logical time, and regularly synchronize data between them. Logical processors in logical time can be synchronized with wall-clock time. The primary component of a DSS simulation is its configuration file. This file contains the AP2633 models used, with their location and execution frequency. The allocation produced by the allocation module can be converted in DSS configuration file. A more detailed description of DSS can be found in (Deschamps et al. 2017).
ASPIC is a real-time simulation framework. In this framework, logical processors are scheduled according to the scheduling policy, in real-time. Depending on the scheduling policy, logical processors can be preempted. A description of ASPIC can be found in (Casteres and Ramaherirariny 2009).
sEaPLANES is a C++ framework of simulation built at the ISAE-SUPAERO in the scope of our study, based on CERTI (Bréholée and Siron 2002). CERTI is an Open source implementation of the general purpose architecture for distributed simulation HLA (High-Level Architecture) (Institute of Electrical and Electronics Engineers and IEEE-SA Standards Board 2010), developed and supported by the ONERA and the ISAE-SUPAERO. The HLA standard defines methods and a framework to build global simulation comprised of smaller simulation, the federates.

sEAPLANES is inspired from (Gervais et al. 2012), every logical processor of sEAPLANES can periodically execute tasks, exchange the data between tasks through a RunTime Infrastructure, and advance time through HLA time management services. sEAPLANES is also inspired by Airbus best practices on distributed simulation framework.
Each logical processor schedules its tasks depending on their periods, and manages intraprocessor and extraprocessor communications. Intraprocessor communication, between two tasks in a single processor, is done with shared memory. Extraprocessor communication, between two tasks in two different logical processors, is managed using the HLA publication-/subscription-based communication mechanisms. Models associated with logical processors can be manually developed, automatically generated, from instance from Matlab, or retargeted from real target or older simulations. The block diagram in fig. 8 illustrates the association of simulation models in tasks on logical processors. Fig. 7 illustrates the implementation of sEAPLANES with HLA/CERTI, with three sEAPLANES logical processors on two CPUs, for scheduling four models. Fig. 9 illustrates an allocation with associated flows of an sLA on sEAPLANES. sEAPLANES were also used in interaction with Matlab HLA toolbox-based simulation, and with Ptolemy-HLA, embedding C-code generated from Matlab (Cardoso and Siron 2018).

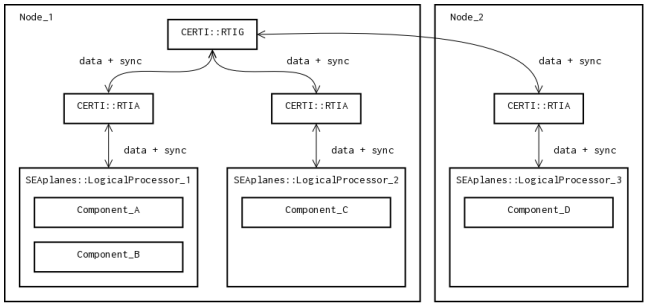

Figure 7: Example of HLA/CERTI, sEAPLANES, and R-ROSACE integration

R-ROSACE on sEAPLANES case study

This case study illustrates the allocation of an aircraft simulation, R-ROSACE, on a physical architecture of simulation, sEAPLANES.

R-ROSACE

The ROSACE (Research Open-Source Avionics and Control Engineering) case study (Pagetti et al. 2014) is a longitudinal flight controller of a medium-size aircraft. It covers different steps from the conception to the implementation of such a controller. ROSACE was chosen since it is an excellent example of a CPS, where a

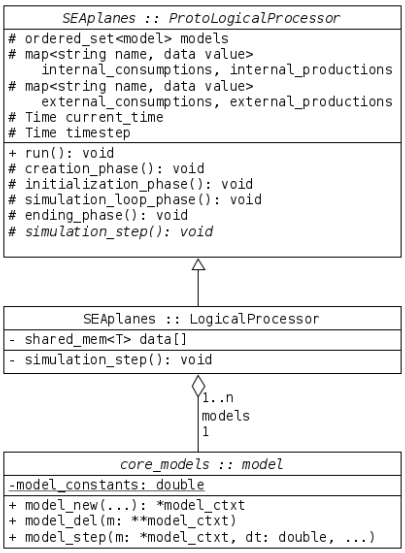

Figure 8: RROSACE sEAPLANES implementation

significant challenge is the need of interactions between the engineers responsible for the aerodynamic characteristics (physical part) and the control law (cyber part). The design of the simulation adds extra interaction with software engineers because of the need to tackle physical system requirements, such as stability, and computers science requirements, such as tasks schedulability and network resources.

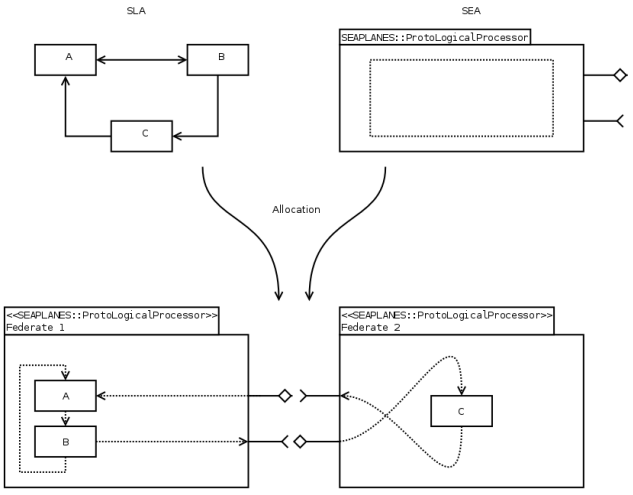

Figure 9: sLA components allocation on sEAPLANES

R-ROSACE is an extension of the open source ROSACE case study, adding redundant controllers. The component breakdown of R-ROSACE, presented in fig. 10, allows the generation of the R-ROSACE sLA. R-ROSACE has been implemented with multiple frameworks, following the architecture of simulation described in sec. "The Simulation Distributed Architecture Model"; the redundancy of controllers is illustrated in

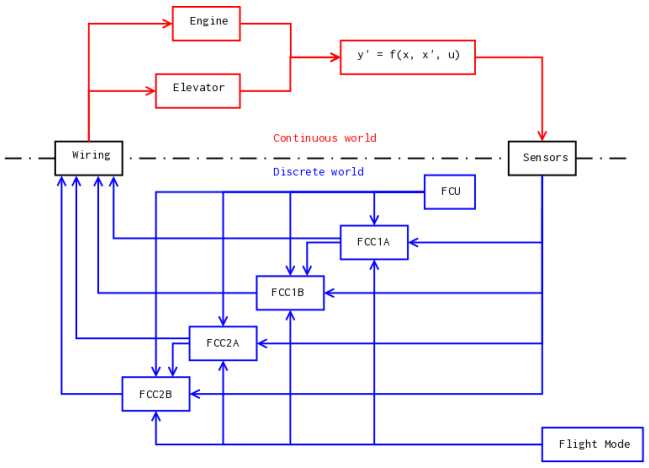

Figure 10: RROSACE components breakdown

the components breakdown figure, with FCC for Flight Control Computer. In R-ROSACE, there are 15 components, 57 channels, and 6 requirements. The requirements are due to the redundancy between FCCs. There is coincidence between each FCCs couple and the wiring, as well as between data received by FCCs couple, and FCC monitoring. An excerpt of this sLA, with a component, a channel and a requirement, is shown in lst. 1. In this excerpt, the following is written:

- Component `engine` – the engine is simulated with a period of 50 ms and an estimated time budget of 1 ms. The simulated engine component has an input port the `delta_x_c`, a change of thrust command, and an output port, `T`, a simulated thrust.
- Channel `engine T` to `flight_dynamic T` – the engine to flight dynamic thrust channel.
- Coincidence requirement between `fcc_1a`, `fcc_1b`, and `wiring` – a coincidence requirement for the wiring to receive the delta thrust computed by FCC 1A and the validation of this command by FCC 1B at the same logical time.

R-ROSACE implementations are tested through multiple operational scenarios. An operational scenario is a set of events that includes the interaction of a system with its environment and its users. A full description of R-ROSACE, operational scenarios, and Simulink models, are available at `https://svn.onera.fr/schedmcore/branches/ROSACE_CaseStudy/redundant/`. The coincidence requirements in R-ROSACE are described in (Deschamps et al. 2018).

R-ROSACE allocation on sEaPLANES

The full sEaPLANES sEa, with an unlimited number of logical processors, and without real-time constraints, is shown in lst. 2.

Listing 1: R-ROSACE sLa excerpt

```
<?xml version="1.0" ?>
<sla name="rrosace" xmlns="">
  <components>
    <component name="engine" period="50ms"
    time_budget="1ms">
      <ports_in>
        <port label="delta_x_c"/>
      </ports_in>
      <ports_out>
        <port label="T"/>
      </ports_out>
    </component>
        <!-- ... -->
  </components>
  <channels>
    <channel>
      <from component="engine" port="T"/>
      <to component="flight_dynamics" port="T"/
    >
    </channel>
        <!-- ... -->
  </channels>
  <requirements>
    <requirement weight="100">
      <coincidence>
        <path>
          <ord index="0">
            <channel>
              <from component="fcc_1a" port="
    delta_x_c"/>
              <to component="wiring" port="
    delta_x_c_1"/>
            </channel>
          </ord>
        </path>
        <path>
          <ord index="0">
            <channel>
              <from component="fcc_1a" port="
    delta_x_c"/>
              <to component="fcc_1b" port="
    delta_x_c_com"/>
            </channel>
          </ord>
          <ord index="1">
            <channel>
              <from component="fcc_1b" port="
    relay_delta_x_c"/>
              <to component="wiring" port="
    relay_delta_x_c_1"/>
            </channel>
          </ord>
        </path>
      </coincidence>
    </requirement>
        <!-- ... -->
  </requirements>
</sla>
```

The allocation tool was run with multiple configurations. Configuration consisted of different heuristics, various inputs order, and additional constraints. In all the cases tested, an allocation has been found, and few cases did not satisfy all constraints. Nevertheless, the allocation is not guaranteed, and with a different case study, we could have had configurations that do not

allow allocations. The resulting allocations have been tested with the real simulator, with tracing and data logging for verification. All the resulting scheduling estimation from allocation were similar to real schedulings on real targets.

Listing 2: sEAPLANES sEA

```
<?xml version="1.0" ?>
<sea name="seaplanes" xmlns="">
    <logical_processors multiperiodic="True"
    real-time="False">
        <intraprocessor_communication/>
        <interprocessor_communication option="
    p2p" sync="synchronous"/>
    </logical_processors>
</sea>
```

When choosing different heuristics, with the inputs provided in the default order and no additional constraints, the results are different allocations, with constraints satisfied in all the cases. First-fit and Worst-fit heuristic are similar; all the components are pushed on the first logical processor. The resulting allocation file is provided in lst. 3, with one component per logical processor, scheduled respecting their orders (`ord`), and their periods. It should be noted that the two different allocations have slightly different simulation results, all acceptable, and different performances. Simulation with first-fit/worst-fit heuristics uses 0,1 s to run one simulation second, while simulation with first-fit/worst-fit heuristics uses 0,4 s to run one simulation second. This is due to the use of the time-consuming synchronization mechanism that is increased with the second allocation.
When modifying the input orders, especially between the FCCs and wiring, or when adding constraints, such as an affinity constraint, the resulting allocation might not verify all the constraints.
Two examples can be considered:

- With the input sequence [FCC1A, wiring, FCC1B] and first-fit heuristic – After allocating FCC1A and wiring in this order on a same logical processor, which means a logical latency of 0 s between the two components, FCC1B cannot be allocated with a logical latency of 0 s from FCC1A, and a logical latency of 0 s to wiring. FCC1B is allocated in the same logical processor, and the logical latency to wiring is 50 ms. Trying to allocate FCC1B to another logical processor leads to 200 ms of logical latency from FCC1A, and 200ms to wiring.
- With the input sequence [FCC1A, FCC1B, FCC2A, FCC2B, wiring], next-fit heuristic, and adding an affinity constraint between FCC1A, FCC1B – FCC1A and FCC1B will be allocated in the same logical processor, while FCC2A and FCC2B in two different one. When trying to allocate wiring, two of the coincidence constraints cannot be satisfied, irrespective of the logical processor chosen.

Listing 3: Allocation with first-fit and worst-fit

```
<?xml version="1.0" ?>
<allocation name="alloc" xmlns="">
  <logical_processor>
    <task name="engine" ord="0" period="50ms"/>
    <task name="elevator" ord="1" period="50ms"/>
    <task name="flight_dynamics" ord="2" period="
    50ms"/>
    <task name="h_filter" ord="3" period="100ms"/>
    <task name="az_filter" ord="4" period="100ms"/
    >
    <task name="Vz_filter" ord="5" period="100ms"/
    >
    <task name="q_filter" ord="6" period="100ms"/>
    <task name="Va_filter" ord="7" period="100ms"/
    >
    <task name="fcu" ord="8" period="200ms"/>
    <task name="flight_mode" ord="9" period="200ms
    "/>
    <task name="fcc_1a" ord="10" period="200ms"/>
    <task name="fcc_1b" ord="11" period="200ms"/>
    <task name="fcc_2a" ord="12" period="200ms"/>
    <task name="fcc_2b" ord="13" period="200ms"/>
    <task name="wiring" ord="14" period="50ms"/>
  </logical_processor>
</allocation>
```

Conclusion and perspectives

In this paper, we presented an allocation tool implementing the method we developed in previous work. We developed an extensible allocation tool, that can use multiple methods for allocating sLA components on sEA tasks. We developed basic heuristics to test the allocation tool, and we displayed them with the R-ROSACE on sEAPLANES case study. Although this case study is straightforward regarding the target at Airbus, we proved on this small case study that we can automatically generate allocation and scheduling, respecting requirements, from the description of the components and execution architecture.

Nevertheless, one of the first criticisms we can make on our tool is that the implemented heuristics are too limited. Indeed, depending on the components input order, the result can vary enormously. The problem is that a component placed during the execution of one of these heuristics cannot be moved, even if a move could significantly improve the results.

Moreover, the result depends greatly on the validity of the inputs. Adding constraints that do not exist due to overzealousness can have a negative impact on the result. Conversely, not identifying a constraint can lead to unrepresentative scheduling, without our tool being able to indicate it fully.

Finally, the currently developed tool is modular and extensible, with different scheduler, communication, synchronization, and heuristics. An inconvenience is that adding a sEA with exotic local schedulers or communications is currently expensive in our tool. Many methods need to be overridden. For instance, the delays estimation on data paths. We are aware that our tool is today only a proof of concept, but such complexity is not to be neglected in the context of industrialization.

While the problems of input validities and the cost of adding new sEAs are implementation or integration challenges, for which we do not have any improvement proposals for the moment, we do have a way to improve the heuristics. The next step in our work is to implement a heuristic for which the order of the input components has no impact. We have already identified a metaheuristic candidate, simulated annealing adapted to CPS simulation scheduling, by considering simulation constraints, and generating state neighborhood from partitioning and mapping.
Finally, the step of creating sEAPLANES federates from allocation files is currently quite fast, but manual. We also plan to set up code generators from the allocation files.

Acknowledgment

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

Blockwitz T.; Otter M.; Akesson J.; Arnold M.; Clauss C.; Elmqvist H.; Friedrich M.; Junghanns A.; Mauss J.; Neumerkel D.; Olsson H.; and Viel A., 2012. *Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models*. 173–184. doi:10.3384/ecp12076173.

Bréholée B. and Siron P., 2002. *Certi: Evolutions of the onera rti prototype*. In *Fall Simulation Interoperability Workshop*.

Cardoso J. and Siron P., 2018. *Ptolemy-HLA: A Cyber-Physical System Distributed Simulation Framework. M Lohstroh et al (Eds): Lee Festschrift, Springer, LNCS 10760*, 1—-21. doi:10.1007/978-3-319-95246-8_8.

Casteres J. and Ramaherirariny T., 2009. *Aircraft integration real-time simulator modeling with AADL for architecture tradeoffs*. In *Automation Test in Europe Conference Exhibition 2009 Design*. 346–351. doi:10.1109/DATE.2009.5090686.

David V.; Fraboul C.; Rousselot J.Y.; and Siron P., 1992. *Partitioning and mapping communication graphs on a modular reconfigurable parallel architecture. Parallel Processing: CONPAR 92—VAPP V*, 43–48.

Deschamps H.; Cappello G.; Cardoso J.; and Siron P., 2017. *Toward a formalism to study the scheduling of cyber-physical systems simulations*. In *Proceedings of the 2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications*. IEEE Computer Society, Rome, Italy.

Deschamps H.; Cappello G.; Cardoso J.; and Siron P., 2018. *Coincidence Problem in CPS Simulations: the R-ROSACE Case Study*. In *Proceedings of the 2018 9th European Congress Embedded Real Time Software and Systems*. Toulouse, France.

Dhall S.K. and Liu C.L., 1978. *On a Real-Time Scheduling Problem. Operations Research*, 26, no. 1,, 127–140.

Gervais C.; Chaudron J.B.; Siron P.; Leconte R.; and Saussié D., 2012. *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, 251–254.

Henriksson D. and Elmqvist H., 2011. *Cyber-Physical Systems Modeling and Simulation with Modelica*.

Institute of Electrical and Electronics Engineers and IEEE-SA Standards Board, 2010. *IEEE standard for modeling and simulation (M & S) high level architecture (HLA): object model template (OMT) specification*. Institute of Electrical and Electronics Engineers, New York. ISBN 978-0-7381-6249-2. OCLC: 682577410.

Lavarenne C.; Seghrouchni O.; Sorel Y.; and Sorine M., 1991. *The SynDEx software environment for real-time distributed systems design and implementation*. In *European Control Conference*. vol. 2, 1684–1689.

Pagetti C.; Saussié D.; Gratia R.; Noulard E.; and Siron P., 2014. *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, 309–318.

Sadvandi S.; Corbier F.; and Mevel E., 2018. *Real time and interactive co-execution platform for the validation of embedded systems*. In *Proceedings of the 2018 9th European Congress Embedded Real Time Software and Systems*. Toulouse, France.

Saidi S.E.; Pernet N.; Sorel Y.; and Khaled A.B., 2016. *Acceleration of FMU Co-Simulation On Multi-core Architectures*. 106–112. doi:10.3384/ecp16124106.

Zeigler B.P.; Praehofer H.; and Kim T.G., 2000. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic Press, San Diego, 2nd ed ed. ISBN 978-0-12-778455-7.

Zheng X. and Julien C., 2015. *Verification and Validation in Cyber Physical Systems: Research Challenges and a Way Forward*. IEEE. ISBN 978-1-4673-7088-2, 15–18. doi:10.1109/SEsCPS.2015.11.