THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)

Présentée et soutenue par :
Henrick DESCHAMPS

le lundi 15 juillet 2019

Titre :

Scheduling of a Cyber-Physical System Simulation

Ordonnancement d'une Simulation de Système Cyber-Physique

École doctorale et discipline ou spécialité :

ED MITT : Informatique et Télécommunications

Unité de recherche :

Équipe d'accueil ISAE-ONERA MOIS

Directeur(s) de Thèse :

M. Pierre SIRON (directeur de thèse)
Mme Janette CARDOSO (co-directrice de thèse)

Jury :

M. Thierry SORIANO Professeur SUPMECA - Rapporteur, Président
M. Pierre SIRON Professeur ISAE-SUPAERO - Directeur de thèse
Mme Janette CARDOSO Professeure ISAE-SUPAERO - Co-directrice de thèse
Mme Claudia FRYDMAN Professeure Université Aix-Marseille - Rapporteure
M. Hans VANGHELUWE Professeur University of Antwerp

This page was intentionally left blank.

To my familly, you've always been there for me.

To my friends, you bring me immense happiness, I couldn't live without you.

To my loved cat, Jacobien, you're the sweetest fur ball on earth.

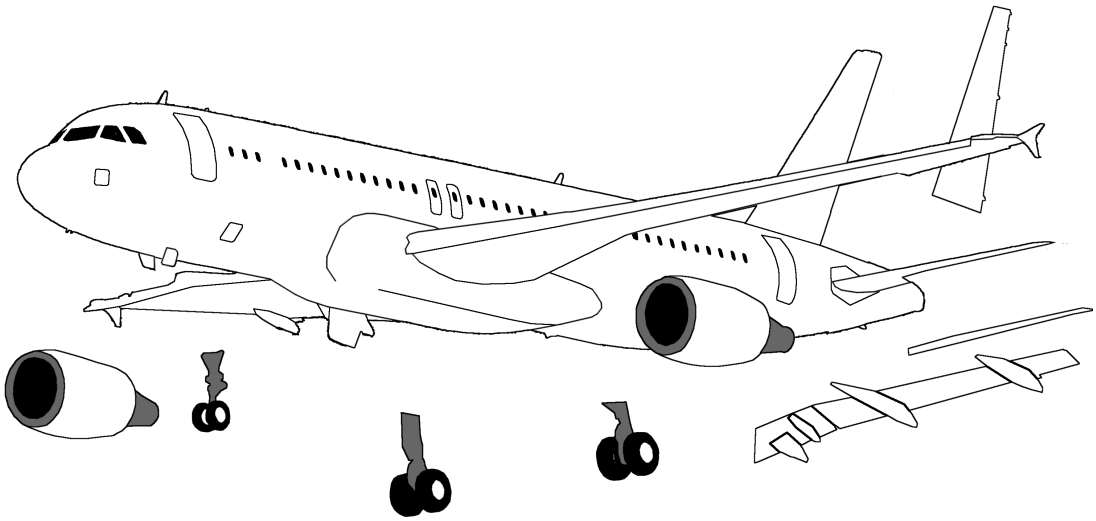
This page was intentionally left blank.

“Hardware met Software on the road to Changtse. Software said: “You are Yin and I am Yang. If we travel together, we will become famous and earn vast sums of money.” And so they set forth together, thinking to conquer the world.

Presently, they met Firmware, who was dressed in tattered rags and hobbled along propped on a thorny stick. Firmware said to them: “The Tao lies beyond Yin and Yang. It is silent and still as a pool of water. It does not seek fame; therefore, nobody knows its presence. It does not seek fortune, for it is complete within itself. It exists beyond space and time.”

Software and Hardware, ashamed, returned to their homes. ”

– The Tao of Programming, 8.4, translated from ancient chinese by Geoffrey James, 1986



Acknowledgments

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.

The people concerned by my acknowledgements are all French-speaking, and only French-speaking for some of them. Therefore I will be continuing in French.

Je souhaiterais exprimer ma gratitude à tous ceux qui m'ont soutenu pendant ce travail.

Tout d'abord merci à ma famille de m'avoir soutenu depuis le début, je pense que c'est devenu dur pour vous de comprendre mes problèmes et enjeux pendant mes études, mais vous avez quand même fait votre possible pour être là, et moi j'en suis là grâce à vous.

Un grand remerciement à Airbus et aux gens qui m'ont entouré là-bas, en particulier Gerlando Cappello, pour m'avoir encadré pendant trois ans, mais aussi tous les membres des équipes R & T et multisystèmes du département simulation, ainsi que tout le département Simulation. Vous ne travaillez pas que sur des modèles, vous êtes des modèles pour moi, vous m'avez beaucoup appris et pas que techniquement. Merci à Bruno et Richard de m'avoir montré comment bien analyser une situation et organiser une solution, vous observer m'a fait grandir. Merci aux experts Bernard, Fernand, Thomas et Patrice. Chacune de vos paroles fait germer des idées. Merci à tous mes anciens collègues, à Virginie, Olivier-Laurent, Sylvain, Frédéric, François, Jacques, Min, Rémi, Alain et tous ceux que je n'ai pas cités ici.

Un grand remerciement aussi à l'ISAE-SUPAERO, à mes encadrants Janette Cardoso et Pierre Siron qui ont participé à lancer ce sujet et qui m'ont épaulé jusqu'à ce que les résultats soient concrétisés. Un grand merci à Jean-Baptiste Chaudron pour toute l'inspiration qu'il a pu me donner, et plus généralement un grand merci aux DISC, Alain, Rob, Odile, et à tous les professeurs. Merci aux stagiaires et doctorants qui ont partagé mon quotidien la moitié du temps.

Merci à tous les acteurs académiques et industriels qui ont permis à ce travail de voir le jour. Merci aux membres du jury, aux relecteurs de mes papiers, en particulier au *shepperd* d'ERTS2, et merci aux membres de la communauté scientifique qui ont assisté a mes présentations, et parfois posé des questions. Vos remarques et questions à tous m'ont offert les retours nécessaires pour que mes travaux divergent le moins possible.

Enfin merci à mes amis. Merci à toi, Jonathan, j'ai été peu disponible pendant cette thèse pour faire la fête, mais je sais que tu m'excuseras et qu'on se rattrapera. Merci aux vieux de l'INSA, vos mariages, anniversaires, vos soirées, j'ai fait mon possible pour venir le plus souvent, et chaque fois ça a été un énorme plaisir. Vous êtes mon phare ici, quand je pense perdre ma route je vous revois, et je sais que je suis sur le bon cap. Merci à ceux qui ont été proches de moi pendant cette thèse. Antoine, je ne comprendrais jamais le plaisir que tu as à courir, mais j'imagine que tu ne comprendras jamais pourquoi moi c'est l'Haskell. Eyal, tu es vraiment cool pour quelqu'un qui fait des maths. J'espère que tu comprendras que c'est un compliment et pas une insulte. Fred, tu es libre alors fais-le, fuis la France, ce pays est trop petit pour les gens qui ont de grandes idées. John, sous ta carapace se cache un petit cœur tout mou. Je n'ai jamais vu personne autant s'inquiéter pour un petit oiseau, bientôt tu rejoindras peut-être le camp de ceux qui ne mangent pas de viande, qui sait ? Et bien sûr au beau bureau, Lucas, tu n'as pas de chance tu commences à peine, mais tu as l'air bien robuste j'ai aucune crainte pour toi. Bastien, avion, plop, hihi, le stade, précis, chihuahua, Toulouse, nage, ratafia, canard, Ramonville-Saint-Agne. C'est la seule phrase que j'ai trouvée qui rassemble ce que tu montres de toi, mais j'en ai une autre pour ce que tu caches un peu : présent, bienveillant, remarquable, doué, vivant, honnête et important. Doriane, ça fait un petit moment qu'on partage le même bureau et je te faisais confiance, je n'ai pas recompté mes affaires en partant. Mais arrivé chez moi je me suis aperçu que beaucoup de joie avait disparu. Tu es sûre que tu ne m'en as pas volé un peu quand je partais pour la laisser dans le bureau ? Bon, j'arrête la blague nulle ici avant que ça parte trop loin et que tout le DISC se mette à chercher où peut être caché la joie dans le bureau et voici ce que je pense vraiment : M. Siron a dit que tu étais une influenceuse dans le bureau, moi je pense qu'il a sous-estimé ton rôle. Quand tu partais en vacances ou en conférence, c'était pas juste une chaise vide de plus dans le bureau, c'était vraiment son âme qui partait, Bastien et Lucas pourront le confirmer. Moi je pars sans regret, mais le cœur serré. Quand Bastien est parti, ça a fait un vide, quand je suis parti, j'imagine que ça a dû vous faire bizarre un peu, et bientôt, ça sera votre tour. Le beau bureau, c'était une belle aventure, mais le monde change tout autour de nous, nos chemins se séparent et je ne sais pas si un jour j'aurai un aussi bel équipage pour naviguer avec moi. Aujourd'hui je ne peux que vous dire merci d'avoir été là, merci d'avoir été vous.

Et aux doctorants qui vont arriver ici : Courage, c'est dur, surtout vers la fin, mais quand c'est fini et qu'on se retourne pour voir ce qu'on a accompli. On est fier, on se dit que ça valait le coup toutes ces petites contributions qu'on a faites tous les jours, on regrette des choses que l'on a perdues, sacrifiées, mais on sait qu'on a gagné quelque chose. Ce quelque chose, ce n'est pas le titre de docteur. Ça, c'est que du papier. Le vrai cadeau de ce travail c'est un talisman magique, unique, personnel qu'on a au fond de son cœur et que jamais personne ne pourra atteindre : la confiance. On a moins peur dans la vie, on sait ce qu'on est capable de faire et au fond on sent que s'il le faut, on peut le refaire et on le refera. Les obstacles ne sont plus des limites, mais des étapes, les erreurs ne sont plus des regrets, mais des leçons, les critiques ne sont plus des flèches, mais des lanternes, les ombres ne sont plus des doutes, mais des défis, les douleurs ne sont plus des souffrances, mais de l'essence, et ce qui se dresse sur la route n'est plus une contrainte, mais un objectif.

Toulouse, Août 2019

This page was intentionally left blank.

Abstract

The work carried out in this Ph.D. thesis is part of a broader effort to automate industrial simulation systems. In the aeronautics industry, and more especially within Airbus, the historical application of simulation is pilot training. There are also more recent uses in the design of systems, as well as in the integration of these systems. These latter applications require a very high degree of representativeness, where historically the most important factor has been the pilot's feeling.

Systems are now divided into several subsystems that are designed, implemented and validated independently, in order to maintain their control despite the increase in their complexity, and the reduction in time-to-market. Airbus already has expertise in the simulation of these subsystems, as well as their integration into a simulation. This expertise is empirical; simulation specialists use the previous integrations schedulings and adapt it to a new integration. This is a process that can sometimes be time-consuming and can introduce errors.

The current trends in the industry are towards flexible production methods, integration of logistics tools for tracking, use of simulation tools in production, as well as resources optimization. Products are increasingly iterations of older, improved products, and tests and simulations are increasingly integrated into their life cycles.

Working empirically in an industry that requires flexibility is a constraint, and nowadays it is essential to facilitate the modification of simulations. The problem is, therefore, to set up methods and tools allowing *a priori* to generate representative simulation schedules.

In order to solve this problem, we have developed a method to describe the elements of a simulation, as well as how this simulation can be executed, and functions to generate schedules. Subsequently, we implemented a tool to automate the scheduling search, based on heuristics. Finally, we tested and verified our method and tools in academic and industrial case studies.

Keywords: Scheduling, Simulation, Cyber-Physical System, Real-Time, Allocation, Heuristics.

This page was intentionally left blank.

Résumé

Les travaux menés dans cette thèse de doctorat s'inscrivent dans le cadre d'un effort plus large d'automatisation des systèmes de simulation industriels. Dans l'industrie aéronautique, et plus particulièrement au sein d'Airbus, l'application historique de la simulation est la formation des pilotes. Il existe aussi des utilisations plus récentes dans la conception de systèmes, ainsi que dans l'intégration de ces systèmes. Ces dernières utilisations exigent un très haut degré de représentativité, là où historiquement le plus important était le ressenti du pilote.

Les systèmes sont aujourd'hui divisés en plusieurs sous-systèmes qui sont conçus, implémentés et validés indépendamment, afin de maintenir leur contrôle malgré l'augmentation de leurs complexités et la réduction des temps de mise sur le marché. Airbus maîtrise déjà la simulation de ces sous-systèmes, ainsi que leurs intégrations en simulation. Cette maîtrise est empirique, les spécialistes de la simulation reprennent l'ordonnancement d'intégrations précédentes, et l'adaptent à une nouvelle intégration. C'est un processus qui peut parfois être chronophage, et qui peut introduire des erreurs.

Les tendances actuelles de l'industrie sont à la flexibilité des moyens de production, à l'intégration d'outils logistiques permettant le suivi, à l'utilisation d'outils de simulation en production, et à l'optimisation des ressources. Les produits sont de plus en plus souvent des itérations d'anciens produits améliorés, et les tests et simulations intégrés à leurs cycles de vie.

Travailler de manière empirique dans une industrie qui nécessite de la flexibilité est une contrainte, et il est aujourd'hui important de facilement modifier des simulations. La problématique est donc de mettre en place des méthodes et outils permettant *a priori* de générer des ordonnancements de simulations représentatifs.

Afin de répondre à ce problème, nous avons mis en place une méthode permettant de décrire les composants d'une simulation, la manière dont cette simulation pourra être exécutée, ainsi que des fonctions permettant de générer des ordonnancements. Par la suite, nous avons implémenté un outil afin d'automatiser la recherche d'ordonnancement, en se basant sur des heuristiques. Enfin nous avons testé et vérifié notre méthode et outils sur des cas d'études académiques et industriels.

Mots-Clefs : Ordonnancement, Simulation, Système Cyber-Physique, Temps-Réel, Allocation, Heuristique.

This page was intentionally left blank.

List of Publications

International Conferences

[DCCS17b] Henrick Deschamps, Gerlando Cappello, Janette Cardoso, and Pierre Siron. 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*, Rome, Italy, October 2017. IEEE Computer Society

[DTCS17] Henrick Deschamps, Bastien Tauran, Janette Cardoso, and Pierre Siron. Distributing Cyber-Physical Systems Simulation: The Satellite Constellation Case. In *Proceedings of the 2017 5th International Federated and Fractionated Satellite Systems Workshop*, Toulouse, France, November 2017

[DCCS18a] Henrick Deschamps, Gerlando Cappello, Janette Cardoso, and Pierre Siron. 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, February 2018

[DCCS18b] Henrick Deschamps, Gerlando Cappello, Janette Cardoso, and Pierre Siron. Implementation of a Cyber-Physical Systems simulation components allocation tool. In *Proceedings of the 2018 32nd European Simulation and Modelling Conference*, pages 112–119, 2018

Electronic Articles

[DCCS17a] Henrick Deschamps, Gerlando Cappello, Janette Cardoso, and Pierre Siron. R-ROSACE: adding redundancy to the ROSACE case study. March 2017

Oral Presentation

[Des18b] Henrick Deschamps. Presentation of the Coincidence Problem in CPS Simulations: the R-ROSACE Case Study, April 2018. url: "`http://projects.laas.fr/IFSE/FAC/program/`"

Poster Presentation

[Des16b] Henrick Deschamps. Scheduling of a cyber-physical system simulation, July 2016. url: "https://cps2016.sciencesconf.org/resource/page/id/10"

Software

[Des16a] Henrick Deschamps. Redundant ROSACE, 2016. url: "https://svn.onera.fr/schedmcore/branches/ROSACE_CaseStudy/redundant/"

[Des17] Henrick Deschamps. SEAplanes, 2017. url: "https://openforge.isae.fr/svn/ordo-simu-dist/code/seaplanes"

[Des18a] Henrick Deschamps. Allocation tool, 2018. url: "https://openforge.isae.fr/svn/ordo-simu-dist/code/allocation_tool"

Contents

Acknowledgments **vi**

Abstract **ix**

Résumé **xi**

List of Publications **xiii**

Contents **xv**

List of Figures **xxi**

List of Tables **xxiii**

Listings **xxv**

List of Abbreviations **xxvii**

Glossary **xxxiii**

I Introduction **1**

1 General Introduction **5**
1.1 Context 6
1.1.1 The French National Association for Research and Technology 6
1.1.2 The industrial actor — Airbus 7
1.1.3 The academic actors 9
1.2 Motivation 10
1.2.1 Industrial use of simulation 10
1.2.2 Current process for simulation integration 13
1.2.3 The new requirements of simulation integration automation 13
1.3 Approach 14

1.3.1 Analyzing industrial processes . . . 14
1.3.2 Expression of formalisms, method, and implementation of tools . . . 15
1.3.3 Implementing simple use case and tools . . . 15
1.3.4 Applying to industrial use case with industrial tools . . . 15
1.4 Document organization . . . 15

2 Background **18**
2.1 Cyber-physical systems . . . 18
2.1.1 Systems . . . 19
2.1.2 Complex systems . . . 22
2.1.3 Time in complex systems . . . 23
2.2 Modelling & Simulation . . . 24
2.2.1 M&S concepts . . . 24
2.2.2 Application to the CPS . . . 27
2.3 Real-time and scheduling . . . 30
2.3.1 RT concepts . . . 30
2.3.2 Temporal aspect of RT . . . 33
2.3.3 Computability and solutions generation . . . 37

II State of the art **38**

3 Distributed simulation principles **43**
3.1 The birth, use and potential of Distributed Simulation . . . 43
3.1.1 The foundation of the domain . . . 43
3.1.2 The current use in industry . . . 46
3.1.3 The future challenges of Distributed Simulation . . . 48
3.2 Benefits and drawbacks of simulation distribution . . . 48
3.2.1 Execution time . . . 49
3.2.2 Model composability and Interoperability . . . 49
3.2.3 Privacy and data integrity . . . 49
3.2.4 Hybrid simulation . . . 50
3.3 The specific characteristics of distributed simulation . . . 50
3.3.1 Clock synchronisation . . . 50
3.3.2 Time management . . . 51

III Cyber-physical system simulation scheduling formalism **53**

4 Problem statement **57**
4.1 Cyber-physical system simulation scheduling . . . 57
4.1.1 Simulation frameworks, simulation middlewares, and their executions . . . 57
4.1.2 Analysis of the main frameworks used in this thesis . . . 59
4.2 Simulation scheduling and real-time scheduling . . . 71
4.2.1 Similarities between simulation scheduling and RT . . . 71

4.2.2 Conditions for an eligible scheduling . . . 76
4.2.3 Scheduling tolerability . . . 77

5 Characterization of the model scheduling 78
5.1 Strategy for formalizing the scheduling of Cyber-Physical System . . . 79
5.1.1 Elements and categories in CPS simulation schedulings . . . 79
5.1.2 Hierarchical modelling . . . 79
5.1.3 Architecture Description Language examples . . . 80
5.1.4 The categorization and allocation strategy . . . 80
5.2 Simulation Logical Architecture description . . . 80
5.2.1 Characterising cyber components . . . 81
5.2.2 Characterising physical components . . . 82
5.2.3 Extension to generic atomic model . . . 83
5.3 Simulation Execution Architecture description . . . 86
5.3.1 Schedulers and communications expressions . . . 86
5.3.2 The sEA ADL . . . 87
5.3.3 sEA representation of frameworks . . . 88
5.4 Allocation of logical architecture on execution architecture . . . 89
5.4.1 sLA partitioning . . . 89
5.4.2 Partitions mapping . . . 90
5.5 sLA requirements and verification . . . 91
5.5.1 The precedence constraint . . . 92
5.5.2 The latency constraint . . . 92
5.5.3 The coincidence constraint . . . 92
5.5.4 The affinity constraint . . . 94

IV Solutions design, implementation, and applications 96

6 Allocation tool and heuristics 101
6.1 From methods to tools . . . 101
6.1.1 sLA and sEA computer-readability adaptation . . . 101
6.1.2 Existing MBSE adaptation approach . . . 104
6.1.3 Full language definition approach . . . 104
6.1.4 Hierarchical structure language use approach . . . 106
6.2 Implementation of the allocation use case . . . 106
6.2.1 Front-end modules . . . 107
6.2.2 The allocation tool modules . . . 114
6.2.3 Back-end adapters . . . 115
6.3 Allocation heuristics . . . 116
6.3.1 Greedy heuristics . . . 117
6.3.2 Simulated Annealing . . . 118

7 The Redundant ROSACE case study **121**
7.1 The original case study . . . 122
7.1.1 ROSACE introduction . . . 122
7.1.2 Original operational scenarios . . . 122
7.2 The RROSACE case study . . . 123
7.2.1 Modifying ROSACE to RROSACE . . . 123
7.2.2 From controllers to redundant Flight Control Computers . . . 123
7.3 Implementation of the case study . . . 127
7.3.1 RROSACE models library . . . 127
7.3.2 Simple loop . . . 130
7.3.3 Testing strategy . . . 131
7.3.4 Results . . . 139
7.4 Method and tool validation . . . 142
7.4.1 Academic validation of concepts . . . 142
7.4.2 Feedback of industrial experience . . . 145

V Conclusion **146**

8 Summary of contributions **151**
8.1 CPS simulation scheduling . . . 151
8.1.1 Problem definition . . . 151
8.1.2 CPS scheduling concepts . . . 152
8.2 RROSACE and SEAplanes . . . 152
8.2.1 RROSACE case study . . . 152
8.2.2 SEAplanes framework . . . 152
8.3 SLA, SEA and allocation method . . . 153
8.3.1 Formalizing CPS scheduling . . . 153
8.3.2 Manipulating CPS scheduling . . . 153
8.4 Allocation tool . . . 153
8.4.1 Allocation tool implementation . . . 153
8.4.2 Scheduling optimization . . . 154
8.4.3 Industrial application . . . 154

9 Perspectives **155**
9.1 Academic perspectives . . . 155
9.1.1 Domain extension . . . 155
9.1.2 Performance evaluation . . . 156
9.1.3 Mastering scheduling . . . 156
9.1.4 SEAplanes improvements . . . 156
9.2 Industrial perspectives . . . 157
9.2.1 Refinement of the allocation tool . . . 157
9.2.2 Tool for guided deployment . . . 157
9.2.3 Parallel catalyst and cost optimizations . . . 157

VI Résumé en Français de la thèse **158**

10 Introduction **163**
10.1 Contexte et motivation . . . 163
10.1.1 Usage industriel de la simulation . . . 164
10.1.2 Processus actuel d'intégration de la simulation . . . 165
10.1.3 Nouvelles exigences de l'automatisation de l'intégration . . . 165
10.2 Approche . . . 165
10.2.1 Analyse des procédés industriels . . . 166
10.2.2 Expression des formalismes, méthode et mise en œuvre des outils . . . 166
10.2.3 Mise en œuvre d'un cas d'utilisation simple et d'outils . . . 167
10.2.4 Application au cas d'utilisation industriel avec des outils industriels . . . 167

11 Formalisation des simulations de systèmes cyber-physiques **168**
11.1 Identification du problème . . . 168
11.1.1 Ordonnancement de simulation de CPS . . . 168
11.1.2 Ordonnancement de simulation et ordonnancement de systèmes temps-réels . . . 171
11.2 Caractérisation de l'ordonnancement de modèles . . . 173
11.2.1 Le Simulation Logical Architecture . . . 173
11.2.2 Le Simulation Execution Architecture . . . 174
11.2.3 Allocation de l'architecture logique sur l'architecture d'exécution . . . 176
11.2.4 Validation des exigences . . . 177

12 Conception, implémentation et application de la solution **179**
12.1 L'outil d'allocation et ses heuristiques . . . 179
12.1.1 Des méthodes aux outils . . . 179
12.1.2 Mise en œuvre du cas d'utilisation de l'allocation . . . 181
12.1.3 Heuristiques d'allocation . . . 183
12.2 Le cas d'étude RROSACE . . . 184
12.2.1 Le cas d'étude original . . . 184
12.2.2 Le cas d'étude RROSACE . . . 184
12.2.3 Mise en œuvre de l'étude de cas . . . 185

13 Conclusion **188**
13.1 Résumé des contributions . . . 188
13.1.1 Ordonnancement de simulation de CPS . . . 188
13.1.2 RROSACE et SEAplanes . . . 189
13.1.3 Le SLA, le SEA et la méthode d'allocation . . . 190
13.1.4 L'outil d'allocation . . . 190
13.1.5 Application industrielle . . . 191
13.2 Perspectives . . . 191
13.2.1 Perspectives académiques . . . 192
13.2.2 Perspectives industrielles . . . 193

VII Appendices **195**

A RRROSACE resources **199**
A.1 Complete RROSACE SLA 199
A.2 RROSACE allocations on SEAplanes 210
A.2.1 Greedy heuristics 210
A.2.2 Specific allocations 213

VIII References **217**

References **218**

List of Figures

1.1 The ANRT logo . . . 6
1.2 The CIFRE logo . . . 7
1.3 The Airbus group major divisions, with marketing segments and major products 7
1.4 Structure of the main actors within Airbus . . . 8
1.5 The academic actors logo . . . 9
1.6 Cockpit of the simulator for an Airbus A350 XWB aircraft . . . 11
1.7 Pilot training with “tonneau antoinette” at levavasseaur’s workshop in 1910 . . . 12
1.8 V cycle steps and possible integration of simulation. . . . 13
1.9 Ph.D. thesis work approach tasks dependencies . . . 17

2.1 Cyber-Physical System simplistic illustration . . . 19
2.2 Illustration of a system . . . 20
2.3 Control of a system through a control-loop . . . 20
2.4 Experimentation on a system . . . 21
2.5 Simulation, as an experimentation on a modelled system . . . 25
2.6 Wallclock, simulation and physical times relations illustration . . . 28
2.7 Full flight simulator of an A350XWB at Airbus . . . 29
2.8 PRISE flight simulation training device, at ISAE-SUPAERO . . . 29
2.9 Task definition . . . 32
2.10 Cyclic scheduling of tasks, with minor and major frames . . . 33
2.11 Latency and jitter illustration . . . 34
2.12 Ilustrations of scheduling general characteristics. . . . 36

4.1 High level view of simulation scheduling in frameworks and middlewares . . . 58
4.2 DSS high level view of simulation scheduling levels . . . 60
4.3 DSS architecture . . . 61
4.4 illustration of DSS short and long cycles and synchronization for one LC . . . 62
4.5 Illustration of DSS short and long cycles and synchronization for two Local Controllers . . . 62
4.6 ASPIC high level view of simulation scheduling . . . 63

4.7 CERTI-based simulation architecture . . . 65
4.8 sEAplanes high level view of simulation scheduling . . . 66
4.9 sEAplanes-based simulation architecture . . . 67
4.10 Two-level scheduling . . . 72
4.11 Relation between task and model scheduling . . . 74
4.12 Ideal and more realistic views of dataflows in a CPS. . . . 75
4.13 Preemption in model scheduling. . . . 75
4.14 Overrun in task scheduling. . . . 76

5.1 Illustation of the CPS simulation scheduling workflow . . . 81
5.2 Simplified view of a component. . . . 85
5.3 The sEA with its double level of scheduling. . . . 87
5.4 Example of partitions from a single set of components . . . 90
5.5 Example of a precedence requirement due to system breakdown . . . 92
5.6 Example of a latency requirement due to asynchronism . . . 93
5.7 Impact of the partitioning of components on data-flow latencies. . . . 93
5.8 Example of an affinity requirement due to systems interactions . . . 95

6.1 Compiler system . . . 102
6.2 Compiler processing . . . 102
6.3 Allocation system . . . 103
6.4 Three-stage compiler structure, adapted for multi-language, multi-platform compilation . . . 103
6.5 Allocation processing . . . 105
6.6 Allocation processing simplified with XML parsers . . . 106
6.7 Allocation tool modules interaction in allocation use case . . . 107
6.8 Example of a concrete implementable precedence requirement . . . 110
6.9 Example of a concrete implementable latency requirement . . . 111
6.10 Example of a concrete implementable coincidence requirement . . . 111
6.11 Example of a concrete implementable precedence requirement . . . 112
6.12 Example of an allocation . . . 114
6.13 sEAplanes integration of models . . . 115
6.14 sEAplanes allocation implementation . . . 116
6.15 Illustration of the neighborhood connections between different model allocations 120

7.1 Design of a new FCC, from controllers of original case study . . . 124
7.2 Instantiation of an FCC COM . . . 125
7.3 Instantiation of an FCC MON . . . 125
7.4 The RROSACE case study components view . . . 126
7.5 RROSACE simple loop sequence diagram. . . . 131
7.6 Centralized simulation of RROSACE . . . 135
7.7 Comparing of discrete RROSACE results with original ROSACE . . . 140
7.8 From discrete to cyber-physical models . . . 141

List of Tables

1.1 Synthezis of simulation types 12

2.1 Simulation and emulation difference in simulation of computer software 25

4.1 Table of links between the simulation frameworks of this work, and the partitioned scheduling 72
4.2 Table of equivalences between multiprocessor real-time scheduling and simulation scheduling 76

5.1 DSS, ASPIC and SEAplanes implementations comparison 89

6.1 List type implementation for $*$-fit heurisitics 117
6.2 parameters values 119

7.1 Mapping of RROSACE models instances with AP2633 models for DSS implementation of RROSACE 139

This page was intentionally left blank.

Listings

6.1 Example of a NED file . . . 105
6.2 Basic structure of an SLA file . . . 108
6.3 Example of a SLA components . . . 109
6.4 Example of an SLA channels . . . 109
6.5 A precedence in an SLA . . . 110
6.6 A latency constraint in an SLA . . . 111
6.7 A coincidence constraint in an SLA . . . 112
6.8 An affinity constraint in an SLA . . . 113
6.9 The SEAplanes SEA . . . 113
6.10 Example of an intermediate scheduling allocation . . . 114
7.1 elevator.h . . . 128
7.2 elevator.c . . . 129
7.3 rrosace.fed . . . 133
7.4 ElevatorLP.h . . . 135
7.5 ElevatorLP.cpp . . . 137
7.6 rrosace_first_fit.is . . . 142
7.7 rrosace_best_fit.is . . . 143
7.8 rrosace_best_fit_with_affinity.is . . . 143
7.9 RROSACE allocation problem . . . 144
7.10 rrosace_best_fit_2_lps.is . . . 144
A.1 rrosace.sla . . . 199
A.2 rrosace_first_fit.is . . . 210
A.3 rrosace_next_fit.is . . . 211
A.4 rrosace_best_fit.is . . . 212
A.5 rrosace_worst_fit.is . . . 212
A.6 rrosace_with_affinity_req.sla . . . 213
A.7 rrosace_best_fit_with_affinity.is . . . 213
A.8 RROSACE allocation problem . . . 214
A.9 rrosace_best_fit_2_lps.is . . . 215

This page was intentionally left blank.

List of Abbreviations

Acronyms

AADL Architecture Analysis & Design Language

ABS Agent-based Simulation

A/C Aircraft

ADIRS Air Data Inertial Reference Unit

ADL Architecture Description Language

ALSP Aggregate Level Simulation Protocol

API Application Programming Interface

ASPIC Simulation Framework for Integration and Design (Atelier de Simulation Pour l'Intégration et la Conception)

ATA Air Transport Association of America

CC Community Controller

CCSL Clock Constraint Specification Language

CM Central Manager

CMSD Core Manufacturing Simulation Data

CORBA Common Object Request Broker Architecture

CPS Cyber-Physical System

DAE Differential Algebraic Equation

DES Discrete-event Simulation

DEVS Discrete Event System Specification

DIS Distributed Interactive Simulation

DOF Degree Of Freedom

Dist-Sim Distributed Simulation

DSS Distributed Simulation Scheduler

Dist-Sys Distributed System

FADEC Full Authority Digital Engine Control

FAL Final Assembly Line

FCC Flight Control Computer

FCS Flight Control Systems

FCU Flight Control Unit

FFS Full Flight Simulator

FIFO First In First Out

FMI Functional Mock-up Interface

FOM Federate Object Model

FTD Flight Simulation Training Device

GPU Graphical Processing Unit

GVT Global Virtual Time

HitL Human-in-the-loop

HLA High Level Architecture

HMI Human Machine Interface

HPC High Performance Computing

HiL Hardware-in-the-loop

IaaS Infrastructure as a Service

ICT Information and Communication Technologies

I/O Input / Output

IoT Internet of Things

IR Intermediate Representation

IRMs Interoperability Reference Models

LC Local Controller

LLVM Low Level Virtual Machine

LP Logical processor

MARTE Modeling and Analysis of Real-Time and Embedded systems

MBSE Model-Based Systems Engineering

MiL Model-in-the-loop

MKP Multiple Knapsack Problem

M&S Modelling & Simulation

MOM Management Object Model

MSaaS Modelling Simulation as a Service

NED NEtwork Description

NP Nondeterministic Polynomial time

ODE Ordinary Differential Equation

OMG Object Management Group

OMT Object Model Template

OOP Object-Oriented Programming

OS Operating System

PaaS Platform as a Service

PADS Parallel and Distributed Simulation

PDAE Partial Differential Algebraic Equation

PDE Partial Differential Equation

PDES Parallel Discrete Event Simulation

Ph.D. Philosophiæ Doctor

PoC Proof of Concept

PRISE Platform for Research in Embedded Systems Engineering (Plate-forme pour la Recherche en Ingénierie des Systèmes Embarqués)

P2P Peer-2-Peer

RMS Rate-Monotonic Scheduler

ROSACE Research Open-Source Avionics and Control Engineering

RROSACE Redundant ROSACE

RT Real-time

RTI Run Time Infrastructure

RTIA RTI Ambassador

RTIG RTI Gateway

SaaS Software as a Service

SEA Simulation Execution Architecture

SEAplanes SEA proto-LP Allocation Nodes with Extensible inline Scheduler

SiL Software-in-the-loop

SA Simulated Annealing

SLA Simulation Logical Architecture

SOA Service Oriented Architecture

SOM Simulation Object Models

SysML Systems Modelling Language

TCP Transmission Control Protocol

TEM Timestamped Event Message

UDP User Datagram Protocol

UML Unified Modelling Language

WCET Worst Case Execution Time

WCTT Worst Case Transmission Time

XML EXtensible Markup Language

Abbreviations

a_z Body vertical acceleration

δ_e Elevator deflection

δ_{e_c} Elevator deflection command

e.g. $exempli\ gratia$ (for example)

h Inertial altitude

$i.e.$ $id\ est$ (that is)

m Meter

$m.s^{-1}$ Meters per second

ms Milliseconds

s Second

q Pitch rate

t Time

v_a True air speed

v_z Inertial vertical speed

Institutions

ANRT National Association for Research and Technology (Association Nationale de la Recherche et de la Technologie)

CIFRE Industrial Agreement for Research Training (Conventions Industrielles de Formation par la REcherche)

DARPA Defense Advanced Research Projects Agency

DISC Department of Complex Systems Engineering (Département d'Ingénierie des Systèmes Complexes)

DMSO Defense Modeling and Simulation Office

EADS European Aeronautic Defence and Space company

EDMITT Doctoral School of Mathematics Computer Science Telecommunications of Toulouse (École Doctorale Mathématiques Informatique Télécommunications de Toulouse)

i4MS Innovation for Manufacturing SMEs

IEEE Institute of Electrical and Electronics Engineers

ISAE Higher Institute of Aeronautics and Space (Institut Supérieur de l'Aéronautique et de l'Espace)

NATO North Atlantic Treaty Organization

NREN National Research and Education Network

SAC Standards Activity Committee

SUPAERO National SUPerior school of AERONautics and space (école nationale SUPérieure de l'AÉRONautique et de l'espace)

This page was intentionally left blank.

Glossary

System

actuator An object that transforms energy into a physical phenomenon that provides work, according to command, to control an environment. The usable energies, and physical phenomena can be of an extremely varied nature, which will be omitted in this work in order to focus on the principle of control.

competition Competition is the ability of a system to execute cooperative and competitive instructions.

complex system A complex system is a system composed of a high number of elements, in strong interaction, which makes the overall behavior of the system difficult to predict by observing its elements.

controller A controller is a system that allows to control other systems, placed in a control loop.

cyber-physical systems "A *cyber-physical system* consists of a collection of computing devices communicating with one another and interacting with the physical world via sensors and actuators in a feedback loop." [Alu15]

delay A duration of time.

distributed system A distributed system is a set of autonomous and interconnected computing units. The calculation units have the ability to coordinate their work and share their resources in a transparent way for an external actor, who will only see one system. A distributed system has major characteristics according to the scientific literature:

- Resource sharing.
- Openness.
- Competition.
- Scalability.
- Fault tolerance.
- Transparency.
- Heterogeneity.

embedded system An embedded system is a system to which a specific task is defined in a given environment, with limited resources, especially in terms of time. Embedded systems are often real-time systems.

environment From a system's point of view, the environment is the set of elements outside the system that can influence its behavior.

event An instantaneous phenomenon that can modify the state of a system.

fault tolerance The ability of a system to resist failure. Transparency and redundancy in a distributed system make it much more fault-tolerant than a stand-alone system.

heterogeneity Heterogeneity is related to openness, the ability of a distributed system to use different types of hardware, software and data components.

jitter Jitter is the variation in latency over time.

latency Latency is the time difference between cause and effect in an observed system. There are several types of latencies in engineering and in particular, all following the same concept, but applied differently. For example, network latency is the difference between receiving and sending a packet. In simulation, latency is the time difference between the initial input and the distinction of the output by the simulator user.

logical clock A *logical clock* is a system that can timestamp events with a logical time [Lam78]. Leslie Lamport defines a *logical clock* "C_i for each process P_i to be a function which assigns a number $C_i\langle a\rangle$ to any event a in that process."

logical time The *logical time* is an abstraction of time used to timestamp events [Lam78].

network A network, in the meaning of a computer network, is a set of equipment that exchanges information in a codified way. In the context of a cyber-physical system integrating a high number of embedded systems dedicated to specific tasks, the notion of network is essential.

openness Openness characterizes the interface quality of the components of a system. A good openness also facilitates scalability.

resource sharing Resource sharing is the means of accessing hardware, software and data by multiple parties. This notion is linked to competition.

scalability Scalability refers to the ability to change size while still satisfying performance requirements, often used to address the transition to a very large scale.

sensor A sensor is a system capable of transforming the state of a measurable physical quantity into another physical quantity, most often more easily measurable, such as an electrical voltage that can be digitized.

system A system is a set of elements, determined by a boundary separating it from its environment. The arrangement of the elements of a system has at some point formed its state, and the interaction of the system with its environment is done through interfaces that can be inputs and outputs. A system can also have functions, which will be abstract in our work, and may require resources, which will not be considered.

transparency Transparency is the ability of a system to be seen as a single entity by an external actor. Transparency remains the most cross-cutting concept. Resource sharing must be transparent, the location of a system, its movement and replication must be transparent, the scalability must be transparent.

Modelling & simulation

agent-based model A class of computer simulations where entities are autonomous agents interacting with each others.

AP2633 model The AP2633 model is defined by Airbus in order to have a process to guarantee the interoperability of simulation execution. An AP2633 model offers an entry point, via a function, and a state machine using predifined global variables. The states of the state machine, with the entry point, allows the framework to schedule the AP2633 models. The states being:

- **LOAD** data loading, file opening, models configuration and connections creation ;
- **INIT** internal initializing ;
- **REINIT** force the model to a point ;
- **RUN** simulation loop ;
- **HOLD** simulation stop on a timestep ;
- **UNLOAD** output writing, files and connections closing ;

AP2633 models consume and produce data, these data are global variables, with a name used to exchange by homonymy, or by aliases depending the configuration.

application programming interface Interface designed to operate a software component, such as a framework for instance.

attribute A property of a given object.

causal order An order relationship allowing causality to be expressed, when there is a direct precedence between two events.

causality Causality is the relationship between cause and effect. In science, the principle of causality defines cause and effect as two phenomena, and stipulates that cause must precede effect. This is a necessary assumption for any coherent mathematical modeling, which has never before been invalidated by experience.

chronological order An order relationship that allows the existence of one event to be expressed without causality before another.

digital twin A digital twin is a digital replica of a system. Such a replica makes it possible to apply best practices from the IT world to the industrial world, such as version management. In addition, they allow to analyze and correct errors arriving on real hardware

in simulation, by learning from a data flow of real systems. Digital twins are still rarely used today because they require the implementation of methods and tools to update the model according to the system.

discrete event simulation A technique used in the study of systems. It consists of a software model in which the change in the state of a system over time is a series of discrete events. Each event occurs at a specific time and changes the status of the system.

experiment "An experiment is the process of extracting data from a system by exerting it through its inputs" [CK06].

fidelity Functional similarity between a model and the system it represents.

flight simulation training device Similator similar to a full-flight simulator, but sometimes simplified, for flight and navigation procedures, including or not the reproduction of a cockpit and visual.

framework A set of libraries for software development. In the context of a computer simulation, the framework will define the execution, synchronization and communication between the components of the computer simulation.

full flight simulator Physical simulator allowing the simulation of the movements and acceleration of an aircraft and all of its systems. It is composed of a platform capable of representing up to six degrees of freedom, and often imitates a cockpit.

hardware-in-the-loop A technique used to include embedded devices in a simulation.

human-in-the-loop A technique used to include human participants in a simulation.

middleware Middleware is a software that creates a network between different applications. This exchange network can take different forms, one of the most well-known being the use of messages. Middleware is responsible for enabling computer applications to interact, cooperate and transmit information to each other.

model A simulated element copying the behaviour of a real element. A model (M) for a system (S) and an experiment (E) is anything to which E can be applied in order to answer questions about S [CK06].

model-in-the-loop A technique used to include a model in a simulation, the considered models often refers to very high level abstactions of system behavior.

physical time The time in the physical system [Fuj00].

reproducibility This is said of an experiment, when the observation made under specific conditions with given inputs always gives the same outputs. The reproducibility of a measurement cannot guarantee its accuracy or precision if a reliable reference is missing.

simulation An imitation of the operation of a real-world process or system [Ban10]. Simulation are experimentations carried out on models from which information are to be extracted.

simulation time A modelling of the physical time, used by the simulation [Fuj00]. Richard M. Fujimoto defines *Simulation Time* as "a totally ordered set of values where each value represents an instant of time in the physical system being modeled. [...] For any two values of simulation time T_1 representing physical time P_1, and T_2 representing P_2, if $T_1 < T_2$, then P_1 occurs before P_2, and $(T_2 - T_1)$ is equal to $(P_2 - P_1) \times K$ for some constant K. If $T_1 < T_2$, then T_1 is said to occur *before* T_2, and if $T_1 > T_2$, then T_1 is said to occur *after* T_2"

software-in-the-loop A technique used to include retargeted embedded software in a simulation.

state A set of data containing the information necessary to define a system.

wallclock time A hardware clocke linked to time elapsed during simulation execution [Fuj00].

Real-Time & scheduling

asynchronous Asynchronism is opposed to synchronism. It characterizes actions that do not occur at the same time. In the field of computer science, two phenomena are called asynchronous if they are not synchronized. It should be noted that there is a third concept specific to signals, which is found in telecommunications, beyond synchronism and asynchronism. Two signals are said to be plesiochronous if duplication or deletions are made to compensate for clock shifts.

complexity Complexity theory is the theory of theoretical computing, which formally studies the amount of resources, especially time, but also memory space, that an algorithm needs to solve a problem. Problems can be grouped into complexity classes.
We speak of a deterministic or non-deterministic problem, depending on whether or not it is necessary to use a deterministic turing machine to solve them.
Although it can be shown that the deterministic problems that can be solved in polynomial time P are included in non-deterministic polynomial time NP, the opposite has not yet been demonstrated or refuted.
This is an open problem that we will not be addressed in this work. Most specialists speculate that NP-complete problems are not solvable in polynomial time, and we will admit this conjecture. It is possible to determine the complexity of a problem before knowing its solution.

concurrency In computer science, concurrency is an emulation of parallelization using different execution stacks, often called tasks, which can be threads or processes. The operating system uses a scheduler to allocate processor access to threads or processes. Access to the processor is the source of concurrency.

constraint A constraint expresses a restriction on variables. In the context of scheduling, variables can be:

- Temporal — whether in terms of time allocation, precedence or coherence.
- Related to resources — on their uses or availability.

scheduling Cyclic scheduling is a specific scheduling class in which a task scheduling is repeated until the objective of scheduling is achieved.
These schedules are part of predictive schedules, and in this work, we will use the notions of minor frames and major frames, respectively the low and high level frames of the mutli-level structure of time frames of multi-periodic scheduling problems [HS92].

global scheduling In global scheduling, all scheduling resources are accessible to task jobs and a global scheduler can be seen as a single job queue, making it convenient for implementation.
Global scheduling requires a load balancing strategy, a positive point being that there are optimal schedulers with this approach.
Compared to other approaches, global scheduling reduces response time and allows better use of resources.
On the other hand, the global scheduler has limitations such as job or task migration costs, synchronization problems, and the impossibility to predict the execution of the load balancer.

heuristic Finding the optimal solution to a NP-complete problem is extremely difficult. According to complexity theory, there is no exact solution. More generally, empirically finding solutions to small instances of problems is possible, but working with large instances requires setting up methods to approach solutions, often in the form of algorithms.
These algorithms are called heuristics. The heurisitics make it possible to obtain an acceptable solution to the problem under consideration, but it is impossible to guarantee the optimality of this solution [RNP10].

job A job is an instance of a task. Referring to job scheduling rather than task scheduling means that several machines can schedule the same task, and it is these instances that are to be scheduled.

network scheduling Sub-case of task scheduling, applied to the network. Generally, schedulers are always non-preemptive when shaping traffic, and the objectives correspond to an improvement or guarantee of quality of service, such as bandwidth, latency or jitter. Defining a Worst Case Transmission Time (WCTT) often involves limiting latency and jitter. Mastering WCTT is extremely important in a distributed system.

npcomplete NP is a class in the complexity theory meaning Nondeterministic Polynomial time. A problem is NP if it can be verified in polynomial time in relation to the size of the input *i.e.* it can be quickly verified that a candidate solution is indeed the solution to the problem. The NP problem is NP-Complete if all the problems of the NP class are reduced

to it via a polynomial reduction; *i.e.* the problem is at least as difficult as all the other problems of the NP class, so it is not possible to quickly find the best solution[Wol06].

partitioned scheduling In partitioned scheduling, each task is assigned to a processor on which its jobs will be executed exclusively.
This is a scheduling class used in particular in critical industries, with solutions such as AUTOSAR or ARINC 653.
This use is due to the isolation between the cores, and the important study of scheduling frameworks, which often allows to pre-calculate the execution of scheduling, or to obtain a deterministic scheduling.
On the other hand, resources are often underutilized, and finding an optimal solution is equivalent to the problem of bin-packing, known to be NP-hard.

precedence The precedence constraint is a time constraint, it is the most common constraint expressed in scheduling. This constraint can be expressed in the form of a priority relationship, which is noted for two tasks, τ_A and τ_B, with τ_A having to be executed before τ_B:

$$\tau_A \prec \tau_B$$

More formally, we will note, with t_S the reference date associated with the execution of a job of the task τ_S during a cycle (*i.e.* its temporal location), and C_s its duration:

$$t_B - t_A = C_A$$

More generally, when we talk about the temporal location of a task in relation to another, with $d_{A,B}$ a positive temporal constraint, $t_B - t_A \geq d_{A,B}$:

- $d_{A,B} = C_A$ — This is the case of the previously stated precedence.
- $0 \leq d_{A,B} < C_A$ — The constraint is weak, and only the starting of τ_B after τ_A is important.
- $d_{A,B} > C_A$ — The constraint is strong and imposes an additional delay.

rate-monotonic scheduling Rate-monotonic scheduling is a static real-time online scheduling, optimal for a system of periodic, synchronous, independent and on-demand tasks with a pre-emptive scheduler [LL73].
In Rate-monotonic scheduling, tasks with the shortest period are executed first, and lower priority tasks still being executed when a higher priority task is reactivated are paused, i. e. pre-empted, for the time required to execute the prioritised tasks.
A Rate-monotonic scheduling is permissible as long as its load is below 100%, which means that it has sufficient resources to meet all deadlines, i.e. to perform all periodic tasks, without any need to be woken up while it is pre-empted or running.

real-time Real-time systems are systems for which the essential characteristic is that their executions are subject to time constraints, *i.e.* there is a time limit for the end of the execution, called a deadline, beyond which the results of the execution are no longer

valid [CGG+14]. However, soft-real time is distinguished from hard real-time by a form of overrun tolerance, decided by the designers. Generally, real-time systems are embedded and reactive, and time constraints range from micro-seconds to hours. In our case, real-time simulations must respect millisecond constraints, just like the physical system they stimulate.

resource A resource is a mean to be used to perform a task, and available in limited quantity (*e.g.* a CPU). This quantity is called its capacity and is written A_k for a resource k.
A resource may be consumable, in which case its use reduces its number, or renewable if it does not. A renewable resource can be cumulative if several tasks can use it at the same time, otherwise it is disjunctive. A disjunctive resource can be single or multiple. Typically, a single resource corresponds to a problem on a machine, and multiple resources to flow shop, open shop and job shop.

- The flow shop, in which n tasks must pass over all m machines in a single path, such that a machine can only process one task at a time.
- The job shop similar to the flow shop but with multiple paths.
- The open shop, similar to job shop, but the order of the machines is free.

scheduling Scheduling is a solution to a specific problem of sequencing, which consists in deciding on an order to process tasks or jobs in a set. The nature of these tasks can be extremely variable. Scheduling can refer to task scheduling, network scheduling or simulation scheduling depending on the context and nature of the tasks to be scheduled [Pin16].

sequencing Refers to a set of ordered tasks where only the order counts, regardless of the execution dates. In a sequencing process, the notion of tasks is simplified.

synchronous Synchronous etymologically means "at the same time". Two phenomena are said to be synchronous if they are sharing a common time. This implies that these two phenomena must originate at the same time and be carried out at the same speed. In practice this is impossible, especially in computer science. Systems are experiencing a phenomenon of clock drift. Two systems having the same time at a given moment and running at the same speed actually have a small delta difference, which shifts their time. We designate synchronous two systems that are regularly synchronized, *i.e.* their clocks are reset to a common time regularly. The problem of clock synchronization is still open today, especially in distributed systems, because there is no network that guarantees fixed latency (*i.e.* no jitter). It is also impossible to guarantee that two synchronized systems share the same absolute time. However, there are protocols with very good results, such as PTP (precision Time Protocol).

task A task τ_i in scheduling theory is an elementary entity with a start date t_i or end date c_i, a runtime p_i=$c_i - t_i$, which is often found in the computer literature under the name Worst Case Execution Time (WCET), and which consumes resources k with an intensity a_{i_k}.

Depending on the scheduling problem, a task may or may not be split up, this is called pre-emptive or non-preemptive problems. Two tasks with no consistency constraints are called independent tasks. For the notation of a periodic task with a period T_i, the following quadruplet is used:

$$\tau_i = \langle r_i, C_i, D_i, T_i \rangle$$

Where:

- τ_i is the task i.
- r_i is the date of activation of the task.
- C_i is the duration of the task execution.
- D_i is the critical deadline to be respected (deadline from the alarm clock).
- T_i is the period of the task.

tasks scheduling Task scheduling is about calculating task execution dates. To do this, a time organization is applied to a set of tasks using resources while taking into account constraints. A scheduling aims to satisfy one or more objectives, it is on the basis of these criteria that the quality of a scheduling can be judged. Scheduling can have certain characteristics:

- Admissible: A scheduling is considered admissible if it respects all the constraints.
- Semi-active: A scheduling in which a task cannot be advanced without changing the sequence on the resource.
- Active: A scheduling in which no task can be started earlier without delaying the start of another task.
- Without delay: Scheduling in which the execution of a task must not be delayed. Scheduling without delay is active.

Scheduling policies can also be distinguished in class:

- Dynamic / Static: In a static scheduling, the weights ($i.e.$ priorities) of the tasks are defined before the execution of the sequence. In the dynamic, these can vary.
- Idle or not: A scheduling in which dead times can be inserted instead of triggering available tasks, unlike a scheduling without delay.
- Preemptive or not: A task scheduling can be preempted.
- Online / Offline: Scheduling is offline if it is predetermined in advance. Otherwise it is online.
- Centralized / Distributed: Online scheduling is distributed if scheduling decisions are made by a local algorithm on each node. It is centralized when the same decision is made by a single node, whether or not the system is distributed. This node is called the privileged node.
- Optimal or not.

To check the schedulability, different methods exist:

- By simulation.
- Model checking (exhaustive exploration of the system's state space).
- The analytical approach.

Finally, the methods for solving scheduling problems are also divided into categories. The decision is correct if it guarantees the optimality of the solutions found, otherwise it is heuristic, when we observe that it is correct. We should also note that for a scheduling in which the order counts but not the time locations of the tasks, the term sequencing is preferred.
In a distributed system, additional difficulties arise. There is no offline centralized scheduling that is not pre-emptive and whose complexity is not at least polynomial if not NP-complete. Preemptive, it is at least NP-difficult [BRH90, HLV96], see the computational complexity.
In addition, in a distributed system, it is difficult to maintain a consistent view of the state of the system, especially when there are time constraints. This includes synchronization problems and clock drifts. Migrating a task can be expensive, which implies a real problem for the choice of assignment. Network scheduling has a very high impact, and constraints of reliability and availability of systems and networks appear.

Distributed simulation

deadlock In distributed simulation, simulation nodes usually have message queues to process, in order to send their own messages.

A situation in which all simulation nodes are waiting for a potential message from another node, blocking any progress in time from that node until the date of potential receipt of the message, thus blocking itself and the other nodes is called a deadlock.

distributed simulation Distributed simulation is a field of computer science and simulation that addresses the execution of a simulation in a parallel or distributed manner on a computer architecture, usually composed of multiple computers connected by a network.

A distinction is made between parallel simulation and distributed simulation by the type of computer resources used.

- A parallel simulation will be on the scale of a computer, or even a room containing strongly linked computers.
- A distributed simulation will be on the scale of geographically distributed computers.

lookahead The notion of lookahead was created to avoid causal constraints.

The lookahead means that a simulation node that has sent a message will not send one until at least its next lookahead.

For example, when a model X, with a lookahead of L_X, has emitted to another model Y a stamped message at the time T_s, Y infers that it will not receive any messages from X before $T_s + L_X$, and can therefore resume execution accordingly.

The lookahead, although introduced as an artificial blocking mechanism, can be justified by the behaviour of the modelled system. The following phenomena have been listed by R. M. Fujimoto as possible jusification for a lookahead:

- Limitation on communication — Following an event, when a system enters into an internal process before communicating again with the outside world.
- Physical limitation — Depends on the speed at which a system can respond to an event.
- Tolerance and inaccuracies — Depends on the ability of a system to return a correct response despite an error. For example, when the system entries are filtered.
- Non-preemptive behavior — When a system enters a phase that cannot be interrupted by an external phenomenon.
- Precomputing simulation activities — When a system is known to depend only on internal steps for a period of time.

null message The null message is a specific message that does not correspond to an activity in a simulation model, and was designed to avoid deadlocks.

It allows a simulation node X to tell the other nodes with a $T_{s,null}$ message that it will not send any stamped messages before $T_{s,null}$.

The null message can have a strong impact on the network load, which can be reduced when used with the lookahead.

This page was intentionally left blank.

Part I

Introduction

"A system is a set of things — people, cells, molecules, or whatever — interconnected in such a way that they produce their own pattern of behavior over time... The system, to a large extent, causes its own behavior. "

– Donella Meadows in Thinking in systems: A Primer, 2008.

"A model is a physical, mathematical, or logical representation of a system entity, phenomenon, or process. A simulation is the implementation of a model over time. A simulation brings a model to life and shows how a particular object or phenomenon will behave. It is useful for testing, analysis or training where real-world systems or concepts can be represented by a model. "

– in Systems Engineering Fundamentals, 2001.

"Models can easily become so complex that they are impenetrable, unexaminable, and virtually unalterable. "

– Donella Meadows in The unavoidable a priori, 1980.

"We're going to make it happen. As God is my bloody witness, I'm hell-bent on making it work. "

– Elon Musk for Wired, 2008.

Table of Contents

1 General Introduction **5**

1.1 Context 6

1.2 Motivation 10

1.3 Approach 14

1.4 Document organization 15

2 Background **18**

2.1 Cyber-physical systems 18

2.2 Modelling & Simulation 24

2.3 Real-time and scheduling 30

Chapter 1

General Introduction

Contents

1.1 Context 6
1.1.1 The French National Association for Research and Technology 6
1.1.2 The industrial actor — Airbus 7
1.1.3 The academic actors 9
1.2 Motivation 10
1.2.1 Industrial use of simulation 10
1.2.2 Current process for simulation integration 13
1.2.3 The new requirements of simulation integration automation 13
1.3 Approach 14
1.3.1 Analyzing industrial processes 14
1.3.2 Expression of formalisms, method, and implementation of tools . . . 15
1.3.3 Implementing simple use case and tools 15
1.3.4 Applying to industrial use case with industrial tools 15
1.4 Document organization 15

In this chapter, the actors involved, the motivation for this work, the approach to the problem, and the organization of this document will be described.

1.1 Context

The work described in this thesis is a joint project between the Department of Complex Systems Engineering (Département d'Ingénierie des Systèmes Complexes, DISC) of the Higher Institute of Aeronautics and Space (Institut Supérieur de l'Aéronautique et de l'Espace, ISAE)-national SUPerior school of AERONautics and space (école nationale SUPérieure de l'AÉRONautique et de l'espace, SUPAERO) and the EYYS simulation department of Airbus Group, supervised by a Industrial Agreement for Research Training (Conventions Industrielles de Formation par la REcherche, CIFRE), and partly financed by the National Association for Research and Technology (Association Nationale de la Recherche et de la Technologie, ANRT).

This is a three-year Philosophiæ Doctor (Ph.D.) thesis, initiated in March 2016, in the field of critical computing applied to aeronautical simulation.

1.1.1 The French National Association for Research and Technology

The ANRT is an association, born by ministerial decree on 16 October 1953, resulting from the will pushed by the public authorities to merge committees and older associations, with the aim of promoting technical research, helping its members collectively in their R&D activities, representing them before public authorities, French and international organizations [noab].

Figure 1.1 – The ANRT logo

It currently gathers around 350 research and development actors in France, including Airbus Operations and the ISAE, and its stated objective is to improve the French research and innovation system, in particular by enhancing the relationship between public and private research institutes.

The ANRT's three main actions are the CIFRE Conventions, the FutuRIS foresight platform and the improvement of the partnership research practices of Service Europe.

The ANRT provides financial support for the work produced during this doctoral thesis and presented in this manuscript, by a CIFRE Convention.

1.1.1.1 The Industrial Agreement for Research Training

The CIFREs are financial mechanisms fully financed by the French Ministry of Higher Education, Research and Innovation [noa15]. Since the creation of this system 30 years ago, the ANRT has been responsible for allocating the CIFREs.

Figure 1.2 – The CIFRE logo

The CIFRE involves three actors, a private company, a public laboratory and a doctoral student. It provides funding for the hiring of a doctoral student in the company, in collaboration with the public research laboratory. The doctoral student then devotes his time, shared between the laboratory and the company, to his research work.

The ANRT evaluates the applications and attributes the funds in such a way that it responds to the company's desire to develop, while training a future doctor, who must be able to prove real professional research experience and be able to enhance his methodological and scientific achievements.

1.1.2 The industrial actor — Airbus

Airbus is a European industrial group active in the fields of space, civil and military aeronautics [noaa].

Airbus Industrie was founded in the late 1960s in partnership with several European manufacturers, but it was in 2000 that the group also bringing together defence, space and helicopters was created under the name European Aeronautic Defence and Space company (EADS), before being renamed Airbus group in 2013, and then Airbus in 2017.

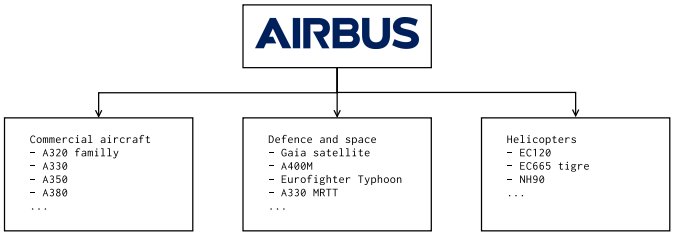


Figure 1.3 – The Airbus group major divisions, with marketing segments and major products

Today, Airbus is divided into three major subsidiaries, as illustrated in Figure 1.3:

- Commercial aircraft:
  producing passenger aircraft, the group's main sector;
- Defense and Space:
  focused on the production of satellites and military aircraft;
- Helicopters:
  placed on the civil and military helicopter markets.

Although European, Airbus has a global presence, with more than 170 sites worldwide and a workforce of about 130,000 people.

In 2017, Airbus had an order book of €690 billion and a sales revenue of €64 billion.

This work, supported by Airbus, took place within Airbus commercial aircraft, in the simulation department.

1.1.2.1 Airbus Commercial Aircraft

Airbus Commercial Aircraft is the aircraft manufacturer of the Airbus group [noac].

It is a leading civil aviation company, competing with Boeing, and having generated a record sales revenue of 52.5 billion euros in 2017.

Airbus Commercial Aircraft is responsible for the design of its products, passenger aircraft, the manufacture of aircraft components, as well as assembly in Final Assembly Lines (FALs), and offers services to its customers, the airlines.

In terms of its products, Airbus Commercial Aircraft is positioned on civil aircraft, the most famous being the A320 family, the A330, A350 and A380. Airbus Commercial Aircraft also offers private civil aircraft and civil cargo aircraft.

Since 2017, Airbus Commercial Aircraft has also been positioning itself with Pop.Up in the segment of autonomous flying vehicles.

1.1.2.2 EYYS, the Airbus Commercial Aircraft simulation department

The Airbus Commercial Aircraft actors involved in this thesis belong to avionics engineering, and more precisely to the simulation department as shown in the Figure 1.4.

Airbus simulation department is responsible for simulation during all phases of aircraft production, from design to pilot training, through product increments.

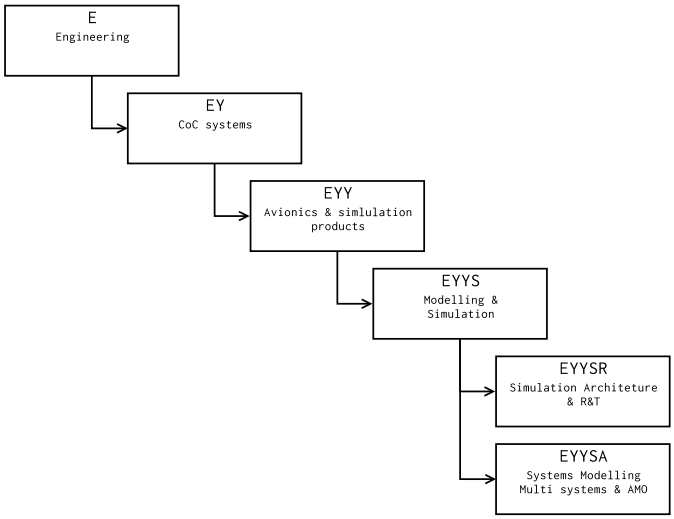


Figure 1.4 – Structure of the main actors within Airbus

To be more precise, this thesis began in the research service of the simulation department, EYYSR, before moving to the EYYSA multisystem service in 2018.

During this work, all the services of the simulation department, as well as some other services such as those for engines, were addressed, in particular, those of simulation integrators, to identify current methods and future needs.

A very positive aspect of working at Airbus is the internal transparency of processes.

1.1.3 The academic actors

The University of Toulouse [DEL] is conducting this doctoral thesis, through the DISC laboratory of the ISAE-SUPAERO school, and through the Doctoral School of Mathematics Computer Science Telecommunications of Toulouse (École Doctorale Mathématiques Informatique Télécommunications de Toulouse, EDMITT) doctoral school.

The ISAE-SUPAERO is an engineering school affiliated to the Ministry of Defence, with training programmes related to aeronautics, in particular, an engineering programme, international, advanced and research masters programmes, continuing programmes, and in our case, doctoral programmes [noae].

The ISAE-SUPAERO is in collaboration with doctoral schools for the supervision of its doctoral students, and within the framework of this thesis, with the EDMITT.

The purpose of the EDMITT is to help its doctoral students, through follow-up and training adapted to their professional project, to integrate effectively into the fundamental or finalized research professions [noad].

(a) ISAE-SUPAERO (b) University of Toulouse (c) EDMITT

Figure 1.5 – The academic actors logo

1.1.3.1 The Department of Complex Systems Engineering

The disc has been developing, within the isae-supaero, mathematical and computer skills for aeronautical and space engineering. This laboratory focuses on the models, methods and tools necessary to control the behaviour and performance of complex systems, the multi-physical or multi-scale nature of the systems studied, their dynamic behaviour, their distributed and communicating structure.

The disc itself is divided into 4 research groups:

- Applied mathematics;
- Communication networks;
- Engineering for critical systems;

- Decision-making systems.

The work of this thesis was carried out within the engineering group for critical systems, in collaboration with other groups, in particular, a publication with communication networks [DTCS17]. The main research themes of the engineering group for critical systems are system engineering with process and model, and the simulation of cyber-physical systems.

1.2 Motivation

The work carried out on this Ph.D. thesis is part of a larger industrial effort to automate industrial simulation systems.

The digitalization of the means of production, the convergence of the virtual world, digital design and management, with real-world products and goods is pushing companies to invest in new methods and tools in order to be the companies at the forefront of tomorrow's innovation [Kag15].

The current trend is the flexibility of production resources, the integration of logistics tools to monitor huge information flows, and the increasing use of simulation tools, while optimizing energy and raw materials used.

Airbus has a clear desire to be at the cutting edge of tomorrow's industrial world, and the ISAE-SUPAERO trains its students in future working methods. As Airbus' modernization also involves modernizing its current processes, it has decided to work with the ISAE-SUPAERO, one of the best schools in the field of aeronautics in Europe, to develop a plan to seek a local improvement path for a very specific problem related to the simulation of its passenger aircraft, in order to be ready to meet the future needs of its customers, both internal for aircraft programs and external for airlines and industrial collaborations.

1.2.1 Industrial use of simulation

Simulation has many industrial applications nowadays. In aeronautics, and more specifically within Airbus, three major uses of simulation are distinguishable.

1.2.1.1 Simulation for pilot training

The earliest and most visible application of the simulation is for pilot training.

In comparison to the simulations used within engineering, these simulations focus on features maximising the pilot's experience. Thus some areas may be leaners while others may go further in providing a greater sense of immersion [WLC07, KJ98]

The simulators for pilot training are called flight simulators, and are today computer-based with optional mechanical interface looking like a real aircraft, such as an A350 cockpit in Figure 1.6. Although considered modern, traces of flight simulators can historically be found as early as 1910 in the early days of aviation with Léon Levavasseur's "tonneau antoinette", Figure 1.7. This first flight simulator was only used to train future pilots, underlining the need identified very early on to use the simulation for training.

Figure 1.6 – Cockpit of the simulator for an Airbus A350 XWB aircraft

There is also an emerging trend in the industry of simulation for training in maintenance processes.

1.2.1.2 Simulation for systems design

A more recent use of simulation is for system design support.

These simulations require a very high degree of representativeness, and their complexities mean that they are only very rarely executed in real time [ASY82].

This use is becoming increasingly popular in the industry as it allows, in parallel with system development or in advance, to refine requirements by studying the impact of parameter variations.

Simulation options are now available in many design tools.

1.2.1.3 Simulation for integration tests

This is the most recent use of simulation, not widely democratized.

Certain industries in which Airbus is an example produce increasingly complex systems such as aircraft. In order to maintain control over these systems, they are divided into many

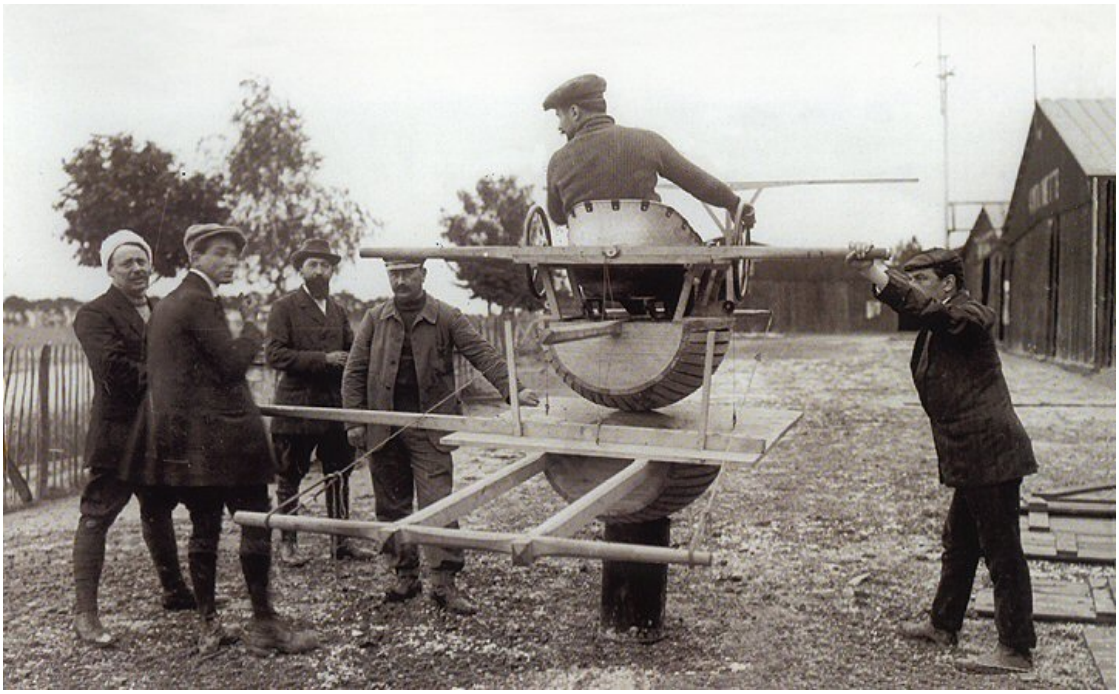


Figure 1.7 – Pilot training with “tonneau antoinette” at levavasseaur’s workshop in 1910

subsystems, which are designed, implemented, tested and validated independently. Figure 1.8 illustrates different possible integration of simulation during V cycle Cyber-Physical System (CPS) product development [SKK+12].

However, when these systems are integrated, problems may arise. These are integration problems, and can have many origins, for instance poor communication of interfaces, or forgotten time aspects.

In order to identify these problems as early as possible, and to increment existing subsystems without introducing regression, simulation can be used.

The next step in the industry of tomorrow is the creation of digital twins, which provides a digital representation of the product throughout its life cycle. Although this is beyond our scope, we can see our current work as a step towards digital twins.

	Representativeness	Real-time execution	Constrained time execution	Covered in this work
Pilot training	Perceivable and accurate	Yes, but soft	Possible	Yes
System design	Essential	Rarely	Possible	No
Integration test	Low upto high	Yes	Possible	Yes

Table 1.1 – Synthezis of simulation types

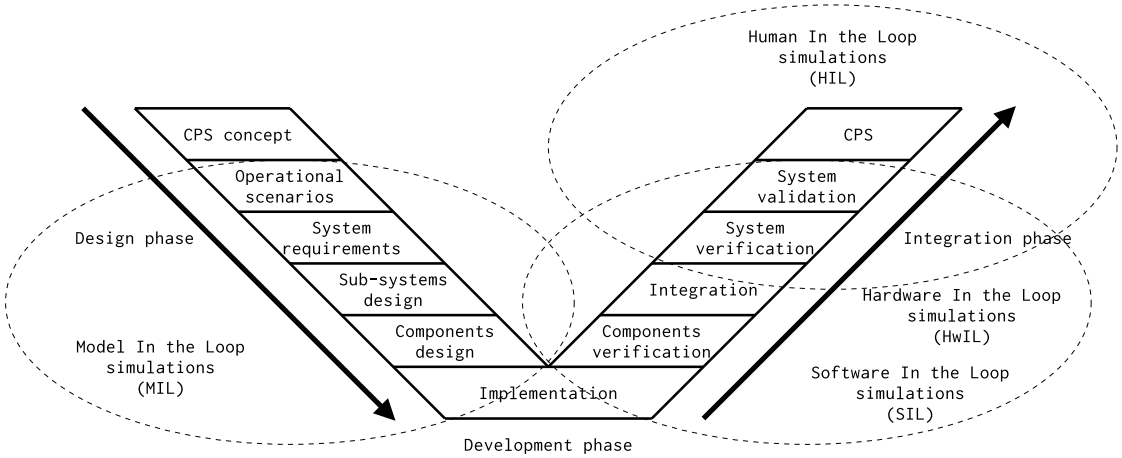


Figure 1.8 – V cycle steps and possible integration of simulation.

Table 1.1 summarizes these simulation types.

1.2.2 Current process for simulation integration

The simulation components can be models, devices, or software code. The models and the integrated software are considered as valid and representative for the integration, only the validation of the integration is to be considered at this stage.

Today, the representativeness of this integration is obtained empirically, the first integration has been compared to real flights, and all integrations are dependent on a previous integration, with sometimes strong modifications and comparisons with real flights.

It is an effective and pragmatic way to manage component integration, but too rigid, and impossible to integrate into methods and tools that often vary the components in use.

1.2.3 The new requirements of simulation integration automation

The recent growth of computing and computer storage capacities is catalyzing the integration of software into industrial processes. This integration addresses cost optimization needs, by accelerating deployments and limiting the use of other resources.

In our society, the emergence of the 4.0 industry is a perfect illustration of this trend. Airbus is one of the key players in the evolution of working conditions. The Airbus commercial aircraft simulation department, EYYS, wanted to participate in this modernization effort, and proposed to set up an alternative to the current industrial means of simulation production.

Based on previous work, the formalization of distributed simulation appeared to be one of the most serious ways to modernize simulation for training and integration.

The DISC of the ISAE-SUPAERO has published work on simulation distribution in the past, with a promising approach of a formalism of distributed simulations. This work basis is a

fertile platform to propose new methods, and new tools, in order to control the temporal execution of a distributed simulation.

1.3 Approach

Historically, Airbus has made considerable investments in modeling and generation for critical systems, particularly for onboard computers, so it has been a natural choice for Airbus to develop an approach and tools to manage the integration of simulation components.

The approach is based on the analysis of the empirical reasons that have led to the integration of existing simulations, the analysis of the industrial tooling needs of simulation actors, and the analysis of existing simulation tools, both at Airbus and at ISAE.

These analyses made it possible to take decisions on the direction of research work, formalisms, methods and tools, and to choose the major Ph.D. thesis work tasks:

- Academic state of the art.
- Airbus simulation analysis.
- Method definition and verification.
- Airbus case study definition.
- Tools design, implementation, validation, and optimization.

Nonetheless, the verification and validation steps required use of case studies, and we couldn't have waited until we had the industrial case study, nor could we have effectively implemented our method with a case study that was too complex. Therefore, we decided to set up in parallel of our work the creation of a simple case study, using an open-source simulation of CPS of a few components, and a simulation framework allowing the distribution of the results.

Figure 1.9 illustrate theses tasks, and their inter-dependencies.

1.3.1 Analyzing industrial processes

The first step of our work consisted in analyzing Airbus' current methods and tools, particularly scheduling generation, and the existing simulation frameworks.

On the current simulation scheduling, the observation is that they are obtained empirically, they satisfy the simulation needs to validate the tests, but there is a slight dissatisfaction in relation with the modernization of the tools, especially in the addition, the deletion and the modification of models, that imply a manual treatment of the modification of the scheduling, which is not consistent with a desire for total digitalization of the processes and methods.

It was essential to have an understanding of the role of simulation integrators, the first actors in contact with simulation scheduling, in order to clearly identify how to express simulation scheduling, and to understand the modification mechanisms involved in new scheduling generation.

It was also necessary to understand how the models are divided and packaged to determine which information is easily accessible to generate a scheduling, and to integrate these models into a simulation framework.

Finally, it has been necessary to interact with stakeholders such as simulation experts, simulation suppliers, and simulation customers to understand their needs from a simulation scheduling perspective in order to steer our project solutions.

1.3.2 Expression of formalisms, method, and implementation of tools

We have made a compilation of the state of the art of existing formalisms related to real-time distributed simulation. We have identified several valuable concepts in a number of formalisms, but none that are directly applicable to our particular problem, without a significant amount of work in the modeling of simulation execution architecture.

We have therefore decided to implement a formalism of our own which is inspired by scientific literature.

1.3.3 Implementing simple use case and tools

To demonstrate our concepts, to provide partial results and to evaluate our tools, we have implemented a simple case study and a simulation framework on the laboratory side. The main objective is to invest in tools very early on, and to be able to do small iterations on top of them, in order to validate our work, and if necessary, to gradually improve it.

1.3.4 Applying to industrial use case with industrial tools

The final part of the project consisted in applying our tool to a real industrial case study at Airbus. The main purpose is to validate the scalability of our work in a complex case study.

This step has also allowed us to identify the unstated elements in the design documents and the user guides of the simulation frameworks, and has also led to the development of the industrial tools prototype.

1.4 Document organization

Our work will be divided as follows.

Part I is an introduction to the field and to the needs. The industrial and academic context, as well as the needs that justified the implementation of our work, will be detailed in chapter 1, while the fundamentals of the fields of science and engineering that we will use are detailed in chapter 2.

Part II is our state of the art, we will focus mainly on distributed simulation in chapter 3.

Part III corresponds to our scientific research work. In chapter 4, we try to clearly identify the problem we are addressing, and the way to formulate it, while we provide our solutions in chapter 5.

In Part IV, we focus on applying the solutions proposed above to concrete problems. We first propose in chapter 6 an implementation of our method as a tool, then use it on an academic case study in chapter 7, adding our feed-back from its use in the industrial field.

Finally, Part V is our conclusion. We start by summarizing all our contributions in chapter 8, then explain their limitations, and the additional work that could be provided in chapter 9.

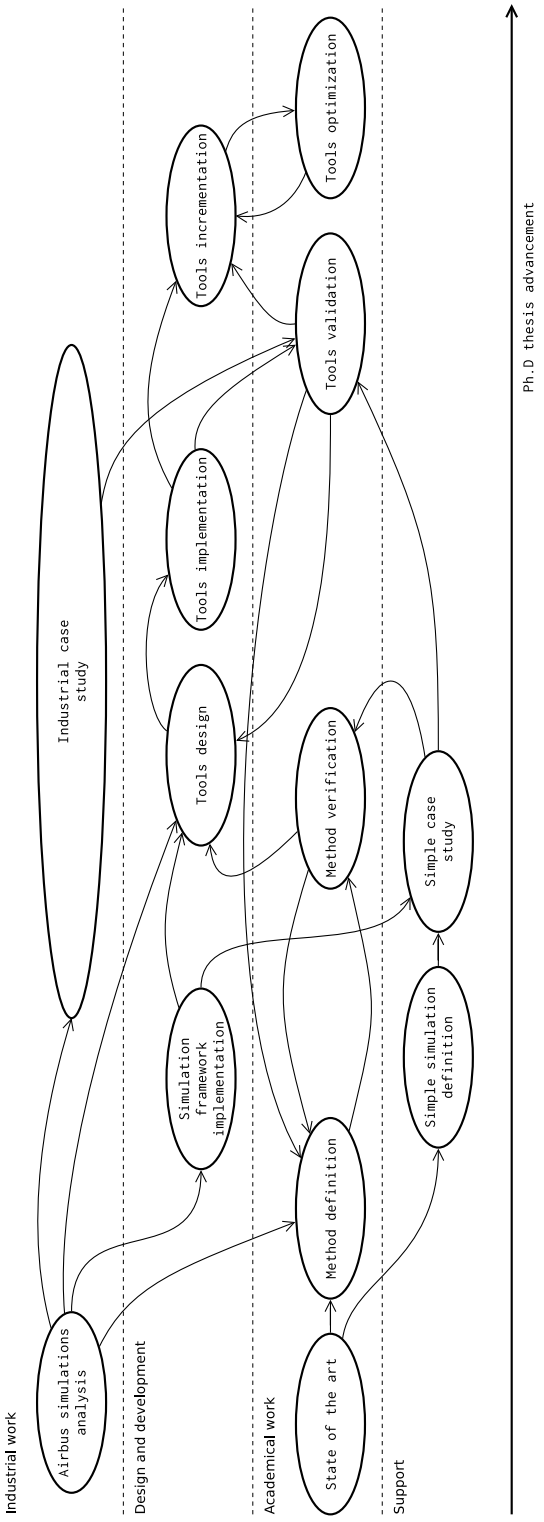


Figure 1.9 – Ph.D. thesis work approach tasks dependencies

Chapter 2

Background

Contents

2.1 Cyber-physical systems 18
2.1.1 Systems 19
2.1.2 Complex systems 22
2.1.3 Time in complex systems 23
2.2 Modelling & Simulation 24
2.2.1 M&S concepts 24
2.2.2 Application to the CPS 27
2.3 Real-time and scheduling 30
2.3.1 RT concepts 30
2.3.2 Temporal aspect of RT 33
2.3.3 Computability and solutions generation 37

This Ph.D. thesis is at the crossroads of several disciplines, in particular systems engineering, real-time software and software engineering, and computer simulation. In this chapter, the most important elements of these areas for further work will be presented and illustrated.

In parallel, we will highlight the most important concepts for understanding our work, thereby allowing us to set a limit on our working perimeter.

2.1 Cyber-physical systems

The work of this Ph.D. thesis focused on the simulation of CPS. CPS are complex systems integrating physical elements and systems.

Cyber-Physical System

"A *cyber-physical system* consists of a collection of computing devices communicating with one another and interacting with the physical world via sensors and actuators in a feedback loop." [Alu15]

Figure 2.1 illustrates this definition, a loop is formed between the physics and the system of a CPS, passing through the actuators and sensors.

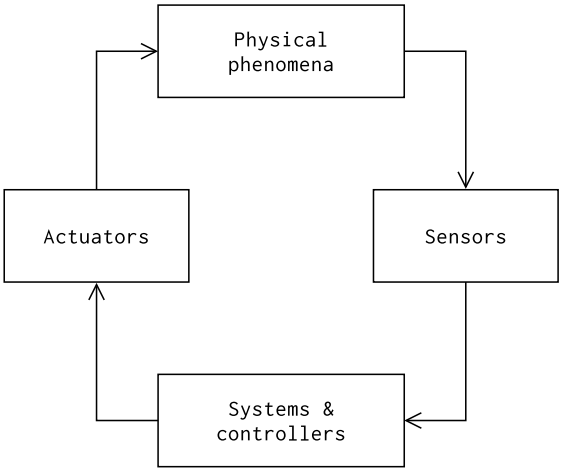


Figure 2.1 – Cyber-Physical System simplistic illustration

In this section we will detail the elements that make up a CPS, while detailing their concepts.

2.1.1 Systems

2.1.1.1 Systems theory and analysis

A CPS is, by definition a system [Wol09].

System

A system is a set of elements, determined by a boundary separating it from its environment. The arrangement of the elements of a system has at some point formed its state, and the interaction of the system with its environment is done through interfaces that can be inputs and outputs. A system can also have functions, which will be abstract in our work, and may require resources, which will not be considered.

Figure 2.2 illustrates the definition of system, and Figure 2.3 illustrates a system in a control loop.

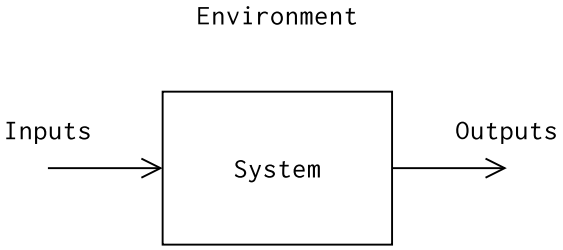


Figure 2.2 – Illustration of a system

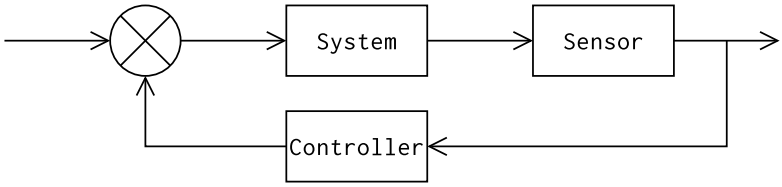


Figure 2.3 – Control of a system through a control-loop

The more global definition of the system environment is then defined. From a system's point of view, the environment is the set of elements outside the system that can influence its behavior.

When studying a system, inputs and outputs are generally expressed as functions over time, and the system as internal variables and functions that change this state and outputs. More specifically, we will define the notion of state as follows: A set of data containing the information necessary to define a system.

Three types of problems exist [SS89, FCZ, FH18]:

- The structure identification problem: The inputs and outputs are known, but not the system. This is the problem most often found in the scientific literature, related to the modelling we will see later.
- The control problem: The system and its outputs are known, and it is necessary to find the right inputs for it. It's an engineering problem.
- The direct problem: The inputs and the system are known, we want to know the outputs.

In the context of cyber physical systems at Airbus, all of these problems are important. The first one is at the modeling stage of the aircraft environment, the second one at the design stage, and the third one at the experimentation, testing and simulation stage. Nevertheless, in this work, the experimental stage is the one we will be working on. We consider the notion of experiment as follows:

Experiment

"An experiment is the process of extracting data from a system by exerting it through its inputs" [CK06].

Figure 2.4 illustrates an experiment conducted on a system. In this illustration, we see that a system under test receives stimuli from an experimenter (which could have been considered part of the system environment), and acts on its environment. The experimenter will be able to observe his actions, while the environment itself may have an impact on the system.

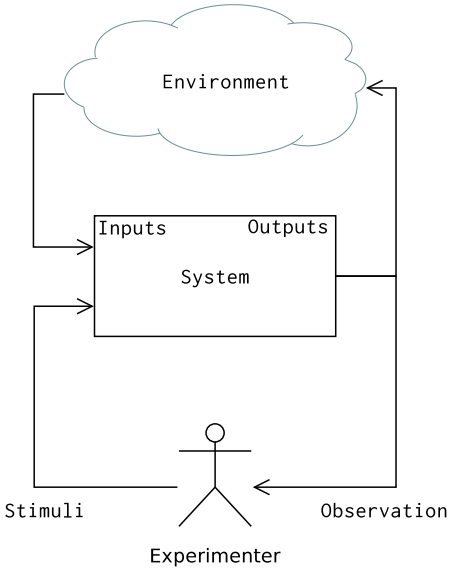


Figure 2.4 – Experimentation on a system

2.1.1.2 Type of systems considered

The CPS at Airbus are also mainly composed of embedded systems [Hea02]. An embedded system is a system to which a specific task is defined in a given environment, with limited resources, especially in terms of time. Embedded systems are often real-time systems. These systems are then linked together by networks [Ali14]. A network, in the meaning of a computer network, is a set of equipment that exchanges information in a codified way. In the context of a cyber-physical system integrating a high number of embedded systems dedicated to specific tasks, the notion of network is essential.

Three main components are part of the networks we will be dealing with:

- The controller: A controller is a system that allows to control other systems, placed in a control loop.

- The sensor: A sensor is a system capable of transforming the state of a measurable physical quantity into another physical quantity, most often more easily measurable, such as an electrical voltage that can be digitized.
- The actuator: An object that transforms energy into a physical phenomenon that provides work, according to command, to control an environment. The usable energies, and physical phenomena can be of an extremely varied nature, which will be omitted in this work in order to focus on the principle of control.

2.1.2 Complex systems

In this work, we are not interested in analyzing an isolated system, but in systems composed of very many smaller systems. Such systems are called complex systems.

Complex System

A complex system is a system composed of a high number of elements, in strong interaction, which makes the overall behavior of the system difficult to predict by observing its elements.

Complex systems are very diverse, typically the Web or datacenters, but also more fuzzy systems such as societies, weather models, and nowadays CPS such as assembly lines that involve a very large number of machines and resources [BY].

We will limit ourselves to the specific cases of complex technological systems, which can be called distributed systems. A distributed system is a set of autonomous and interconnected computing units. The calculation units have the ability to coordinate their work and share their resources in a transparent way for an external actor, who will only see one system. A distributed system has major characteristics according to the scientific literature:

- Resource sharing.
- Openness.
- Competition.
- Scalability.
- Fault tolerance.
- Transparency.
- Heterogeneity.

In our work, we have two notions of distributed systems. We consider distributed simulations of aircraft, which are themselves distributed systems. It is more specifically on the

distribution of the simulation that our work focuses on, taking into consideration the constraints coming from the aircraft. Aspects of the simulations will be discussed below.

Resource sharing is the means of accessing hardware, software and data by multiple parties. This notion is linked to competition. This is a very important notion of our work.

Openness characterizes the interface quality of the components of a system. A good openness also facilitates scalability. Although very important in the context of simulations, and their interfacing standards, it is for us a secondary problem in this work.

Competition is the ability of a system to execute cooperative and competitive instructions. In our work, we emphasize competition, and in particular the mastery of this competition.

Scalability refers to the ability to change size while still satisfying performance requirements, often used to address the transition to a very large scale. The solutions we propose in our work take into consideration scalability. Nevertheless, we will limit our tests and validations to the aeronautics industry.

The ability of a system to resist failure. Transparency and redundancy in a distributed system make it much more fault-tolerant than a stand-alone system. Although very important in the world of aeronautics, it is a minor element of our work.

Transparency is the ability of a system to be seen as a single entity by an external actor. Transparency remains the most cross-cutting concept. Resource sharing must be transparent, the location of a system, its movement and replication must be transparent, the scalability must be transparent. Our work has almost no focus on transparency.

Heterogeneity is related to openness, the ability of a distributed system to use different types of hardware, software and data components. In our work, heterogenity is a constraint to be taken into consideration.

2.1.3 Time in complex systems

The notion of time is difficult to manipulate in a distributed system, so concepts have been defined to simplify this manipulation.

2.1.3.1 Time representation

We have already introduced the link between time and systems by stating that inputs and outputs are expressed as a function of time. An implicit notion of events is then created: An instantaneous phenomenon that can modify the state of a system.

In a distributed system, this notion is important; it is more convenient to manipulate a notion of events than to study an entire system in relation to a single reference. We will come back to the different types of time later.

However, events often require the maintenance of an order relationship, so they are stamped with a common reference, a system-specific time scale, a logical time:

Logical Time

The *logical time* is an abstraction of time used to timestamp events [Lam78].

These logical times are expressed in relation to a logical clock. A *logical clock* is a system that can timestamp events with a logical time [Lam78]. Leslie Lamport defines a *logical clock* "C_i for each process P_i to be a function which assigns a number $C_i\langle a\rangle$ to any event a in that process."

Using this clock, temporal phenomena can then be analyzed in the distributed system under study. For example, we can express a time limit, or a delay (A duration of time.).

2.2 Modelling & Simulation

This section is used as a general introduction to Modelling & Simulation (M&S).

2.2.1 M&S concepts

Definitions

Simulation is a tool, often computerized, allowing actors to observe the results of actions without carrying out any experiments on a real element. Simulation is illustrated in Figure 2.5.

Simulation

An imitation of the operation of a real-world process or system [Ban10]. Simulation are experimentations carried out on models from which information are to be extracted.

The real elements can be in many forms. For instance, we can distinguish between: Physics (mechanical, optical, thermodynamics . . .)

- The chemistry
- The biology
- The finance sector
- The economy
- Computer science
- Industrial processes
- Maintenance
- . . .

The complexity and dynamics of these elements mean that simulations can be very different, and tools will be adapted as needed.

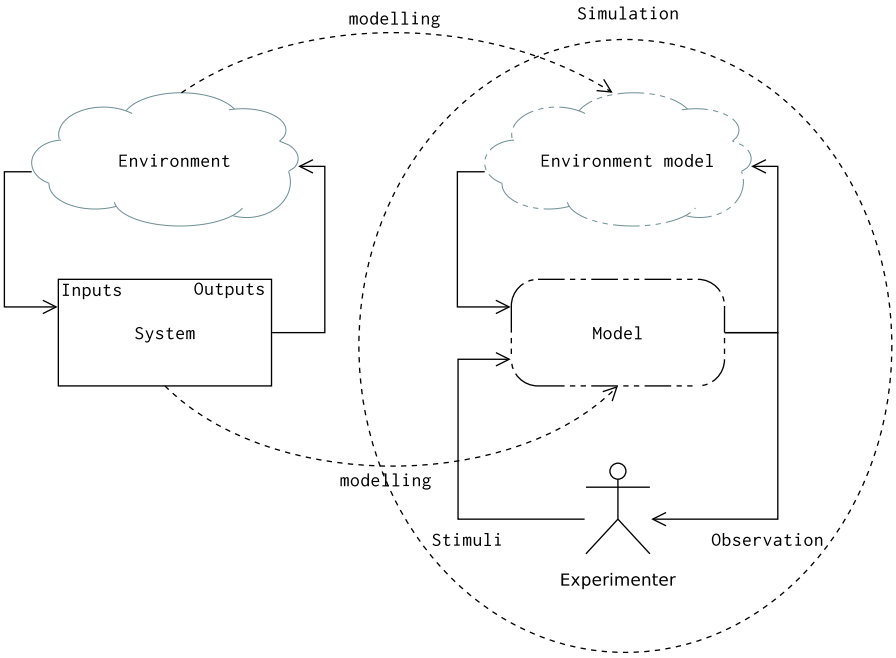


Figure 2.5 – Simulation, as an experimentation on a modelled system

Simulated elements that copy the behavior of real elements are called models. These models can be analog or digital, and it is the model, associated with its inputs and the result it produces, that is called simulation.

Model

A simulated element copying the behaviour of a real element. A model (M) for a system (S) and an experiment (E) is anything to which E can be applied in order to answer questions about S [CK06].

In addition, we will also distinguish between simulation and emulation [McG02].

	Software	Hardware
Simulation	✓	
Emulation	✓	✓

Table 2.1 – Simulation and emulation difference in simulation of computer software

Simulation is the replication of the conceptual behavior of a real element, while emulation is the replication of all the internal mechanics of the real element. Table 2.1 presents a synthetis of this definition.

2.2.1.1 Simulation paradigms

Among the simulation paradigms, three stand out:

- Discrete: The simulation status only changes when discrete events occur.
- Continuous: The state of the simulation changes continuously with time and is represented by equations .
- Hybrid: The system has both continuous and discrete elements.

There are also other paradigms such as Stochastic simulations, where the inputs of the simulation depend on the law of probabilities. But this is beyond the scope of this work.

In this work, we focus on hybrid simulations, with a discretized continuous part using Ordinary Differential Equations (sODEs) [APS98].

We will also define two additional features that we would like to find in a simulation:

- Reproducibility — This is said of an experiment, when the observation made under specific conditions with given inputs always gives the same outputs. The reproducibility of a measurement cannot guarantee its accuracy or precision if a reliable reference is missing.
- Fidelity — Functional similarity between a model and the system it represents.

2.2.1.2 Time representation in simulation

As in systems, simulation involves rigorous time manipulation to take advantage of this discipline.

This manipulation of time requires a definition of the order relationship:

- Causal order: An order relationship allowing causality to be expressed, when there is a direct precedence between two events. Causality is the relationship between cause and effect. In science, the principle of causality defines cause and effect as two phenomena, and stipulates that cause must precede effect. This is a necessary assumption for any coherent mathematical modeling, which has never before been invalidated by experience.
- Chronological order: An order relationship that allows the existence of one event to be expressed without causality before another.

In our work, we will manipulate these two orders, the causal order for certain constraints that can be expressed, and the other chronological order for the execution of our simulation, via three important times, defined below:

- Physical time.
- Simulation time.
- Wallclock time.

Physical Time

The time in the physical system [Fuj00].

Simulation Time

A modelling of the physical time, used by the simulation [Fuj00]. Richard M. Fujimoto defines *Simulation Time* as "a totally ordered set of values where each value represents an instant of time in the physical system being modeled. [...] For any two values of simulation time T_1 representing physical time P_1, and T_2 representing P_2, if $T_1 < T_2$, then P_1 occurs before P_2, and $(T_2 - T_1)$ is equal to $(P_2 - P_1) \times K$ for some constant K. If $T_1 < T_2$, then T_1 is said to occur *before* T_2, and if $T_1 > T_2$, then T_1 is said to occur *after* T_2"

Wallclock Time

A hardware clocke linked to time elapsed during simulation execution [Fuj00].

The relationship between these times, and causality, is illustrated in Figure 2.6.

2.2.2 Application to the CPS

As discussed in the introduction, the field of aeronautics is our main target in this research work. A specific set of simulations targeting aircrafts, in particular Full Flight Simulator (FFS) and Flight Simulation Training Device (FTD), already exists [KJ98, Lee17].

- FFS: Physical simulator allowing the simulation of the movements and acceleration of an aircraft and all of its systems. It is composed of a platform capable of representing up to six degrees of freedom, and often imitates a cockpit. Illustrated in Figure 2.7.
- FTD: Similator similar to a full-flight simulator, but sometimes simplified, for flight and navigation procedures, including or not the reproduction of a cockpit and visual. Illustrated in Figure 2.8.

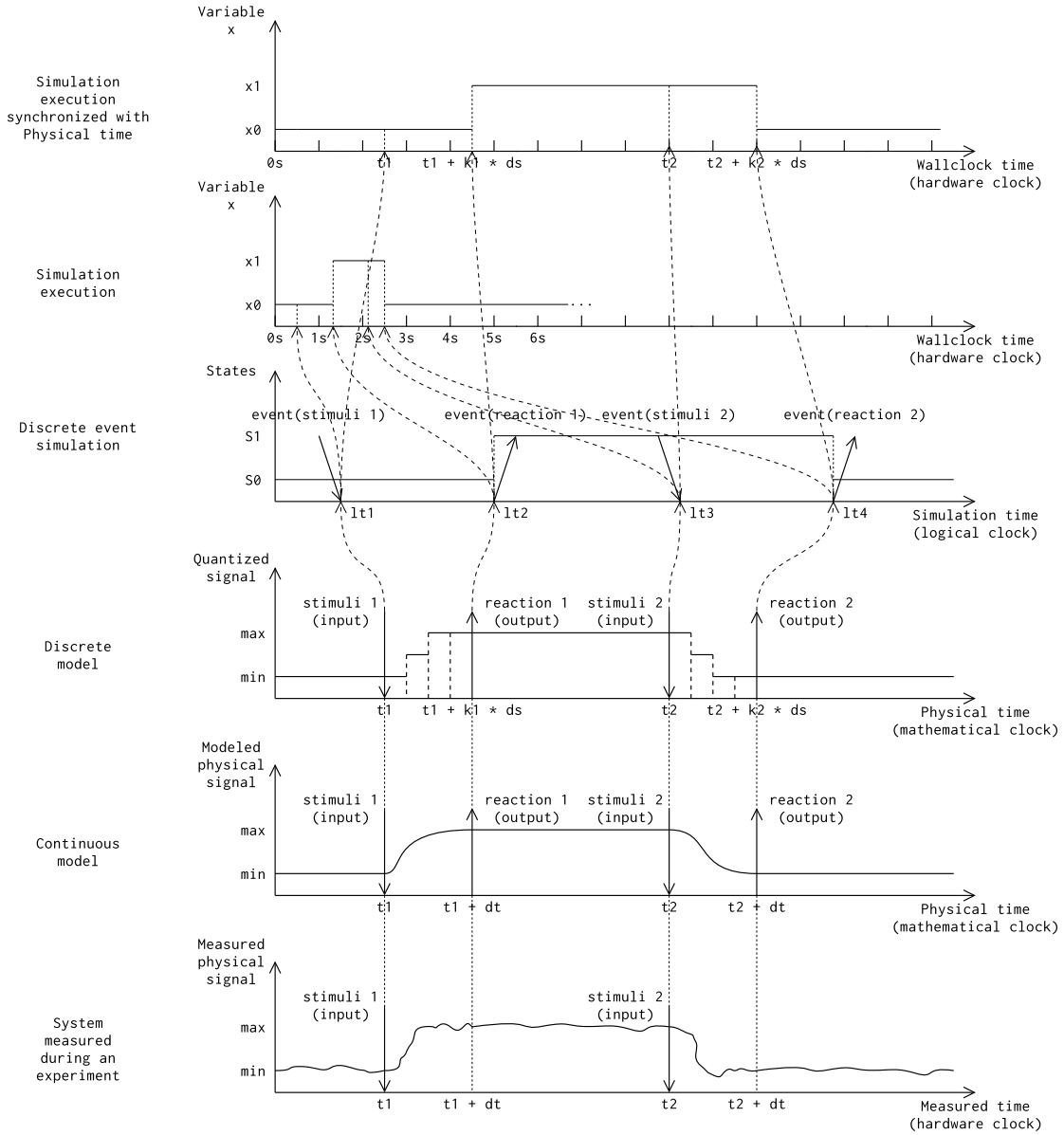


Figure 2.6 – Wallclock, simulation and physical times relations illustration

In this type of simulation, one or more pilots use specific hardware to drive the simulation. Industry is increasingly drawing inspiration from this model for simulation and more generally today to simulate CPS containing many hardware, software and even a pilot as in the case of vehicles, which involves relatively recent techniques for integrating off-simulation components:

- — A technique used to include embedded devices in a simulation.

Figure 2.7 – Full flight simulator of an A350XWB at Airbus

Figure 2.8 – PRISE flight simulation training device, at ISAE-SUPAERO

- Human-in-the-loop (HitL) — A technique used to include human participants in a simulation.
- Software-in-the-loop (SiL) — A technique used to include retargeted embedded software in a simulation.
- Model-in-the-loop (MiL) — A technique used to include a model in a simulation, the considered models often refers to very high level abstactions of system behavior.

The use of simulation increasingly closer to reality, with ever finer boundaries between cyber and physical part, is now generating the willingness of industry and academia to take an interest in digital twins [BR16]. A digital twin is a digital replica of a system. Such a replica makes it possible to apply best practices from the IT world to the industrial world, such as version management. In addition, they allow to analyze and correct errors arriving on real hardware in simulation, by learning from a data flow of real systems. Digital twins are still rarely used today because they require the implementation of methods and tools to update the model according to the system.

We believe that our work is a small contribution to the execution of models that will allow to take one more step towards the industrialization of digital twins, however we will not focus our work in this perspective.

2.3 Real-time and scheduling

The purpose of this section is to introduce to real-time systems and scheduling, in particular to discuss the concepts of constraints, scheduling, and common solutions.

2.3.1 RT concepts

Below is a proposition for defining Real-time (RT):

Real-Time

Real-time systems are systems for which the essential characteristic is that their executions are subject to time constraints, *i.e.* there is a time limit for the end of the execution, called a deadline, beyond which the results of the execution are no longer valid [CGG+14]. However, soft-real time is distinguished from hard real-time by a form of overrun tolerance, decided by the designers. Generally, real-time systems are embedded and reactive, and time constraints range from micro-seconds to hours. In our case, real-time simulations must respect millisecond constraints, just like the physical system they stimulate.

RT is one of the main components of solving the problem of allocating computing resources in systems with limited resources, for example in embedded systems. In this kind of system, having a timely response is just as important as having a correct response. We will

often seek, and this is the case in our work, to solve a scheduling problem, which we can define as follows:

Scheduling

Scheduling is a solution to a specific problem of sequencing, which consists in deciding on an order to process tasks or jobs in a set. The nature of these tasks can be extremely variable. Scheduling can refer to task scheduling, network scheduling or simulation scheduling depending on the context and nature of the tasks to be scheduled [Pin16].

In order to work on scheduling problems, a modeling of RT systems is done, and common entities are identified, in particular the following three:

- Tasks.
- Constraints.
- Resources.

A task τ_i in scheduling theory is an elementary entity with a start date t_i or end date c_i, a runtime p_i=$c_i - t_i$, which is often found in the computer literature under the name WCET, and which consumes resources k with an intensity a_{i_k}.
Depending on the scheduling problem, a task may or may not be split up, this is called preemptive or non-preemptive problems. Two tasks with no consistency constraints are called independent tasks. For the notation of a periodic task with a period T_i, the following quadruplet is used:

$$\tau_i = \langle r_i, C_i, D_i, T_i \rangle$$

Where:

- τ_i is the task i.
- r_i is the date of activation of the task.
- C_i is the duration of the task execution.
- D_i is the critical deadline to be respected (deadline from the alarm clock).
- T_i is the period of the task.

Tasks are illustrated in Figure 2.9.
A constraint expresses a restriction on variables. In the context of scheduling, variables can be:

- Temporal — whether in terms of time allocation, precedence or coherence.
- Related to resources — on their uses or availability.

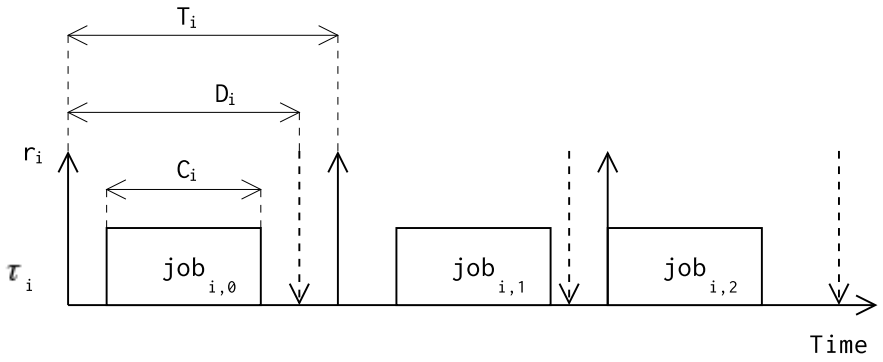


Figure 2.9 – Task definition

A resource is a mean to be used to perform a task, and available in limited quantity (*e.g.* a CPU). This quantity is called its capacity and is written A_k for a resource k.
A resource may be consumable, in which case its use reduces its number, or renewable if it does not. A renewable resource can be cumulative if several tasks can use it at the same time, otherwise it is disjunctive. A disjunctive resource can be single or multiple.
Typically, a single resource corresponds to a problem on a machine, and multiple resources to flow shop, open shop and job shop.

- The flow shop, in which n tasks must pass over all m machines in a single path, such that a machine can only process one task at a time.
- The job shop similar to the flow shop but with multiple paths.
- The open shop, similar to job shop, but the order of the machines is free.

The notion of task being too rigid for some scheduling, especially online, we also define the notion of job, already used in the Figure 2.9: A job is an instance of a task. Referring to job scheduling rather than task scheduling means that several machines can schedule the same task, and it is these instances that are to be scheduled.

In addition, major types of scheduling are distinguished

- Online — scheduling is dynamic and is the responsibility of one or more elements of the system to assign tasks or jobs.
- Offline — the scheduling is pre-calculated and the system enforces this scheduling without modification during the run time.

Nevertheless, in our work, we will only study offline scheduling using tasks. This choice will be detailed in our work, but we wanted to specify the existence of other types of scheduling, while specifying that the work provided here does not address them.

Lastly, the final significant scheduling concept in this work is the notion of cyclical scheduling. Cyclic scheduling is a specific scheduling class in which a task scheduling is repeated until

the objective of scheduling is achieved.
These schedules are part of predictive schedules, and in this work, we will use the notions of minor frames and major frames, respectively the low and high level frames of the mutli-level structure of time frames of multi-periodic scheduling problems [HS92]. Aperiodic tasks exist in other works, they are however negligible in our work and have not been taken into account.

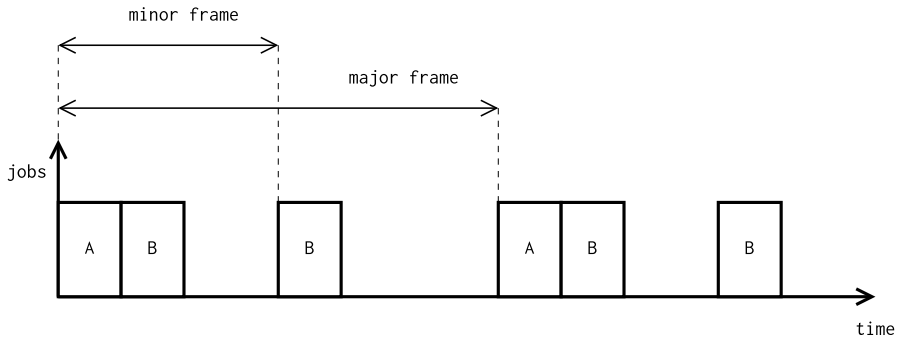


Figure 2.10 – Cyclic scheduling of tasks, with minor and major frames

Cyclical scheduling are illustrated in Figure 2.10.

2.3.2 Temporal aspect of RT

2.3.2.1 Execution, parallelism and synchronisation

When talking about task scheduling, the sequence in which tasks are performed is important, but it should not be forgotten that some tasks can be performed at the same time. More precisely in parallel, or concurrently [Bre09]. In computer science, concurrency is an emulation of parallelization using different execution stacks, often called tasks, which can be threads or processes. The operating system uses a scheduler to allocate processor access to threads or processes. Access to the processor is the source of concurrency.

This notion of parallelism opens the question of the propagation of task results in a scheduling, and their synchronization or not.

Synchronous etymologically means “at the same time”. Two phenomena are said to be synchronous if they are sharing a common time. This implies that these two phenomena must originate at the same time and be carried out at the same speed. In practice this is impossible, especially in computer science. Systems are experiencing a phenomenon of clock drift. Two systems having the same time at a given moment and running at the same speed actually have a small delta difference, which shifts their time. We designate synchronous two systems that are regularly synchronized, *i.e.* their clocks are reset to a common time regularly. The problem of clock synchronization is still open today, especially in distributed systems, because there is no network that guarantees fixed latency (*i.e.* no jitter). It is also impossible to guarantee that two synchronized systems share the same absolute time. However, there are protocols with very good results, such as PTP (precision Time Protocol).

Asynchronism is opposed to synchronism. It characterizes actions that do not occur at the same time. In the field of computer science, two phenomena are called asynchronous if they are not synchronized. It should be noted that there is a third concept specific to signals, which is found in telecommunications, beyond synchronism and asynchronism. Two signals are said to be plesiochronous if duplication or deletions are made to compensate for clock shifts.

Regardless of the synchronization, phenomena of varying delays may exist in the transmission of results between tasks. These delays can be composed of latency and jitter [KSL99].

- Latency is the time difference between cause and effect in an observed system. There are several types of latencies in engineering and in particular, all following the same concept, but applied differently. For example, network latency is the difference between receiving and sending a packet. In simulation, latency is the time difference between the initial input and the distinction of the output by the simulator user.

- Jitter is the variation in latency over time.

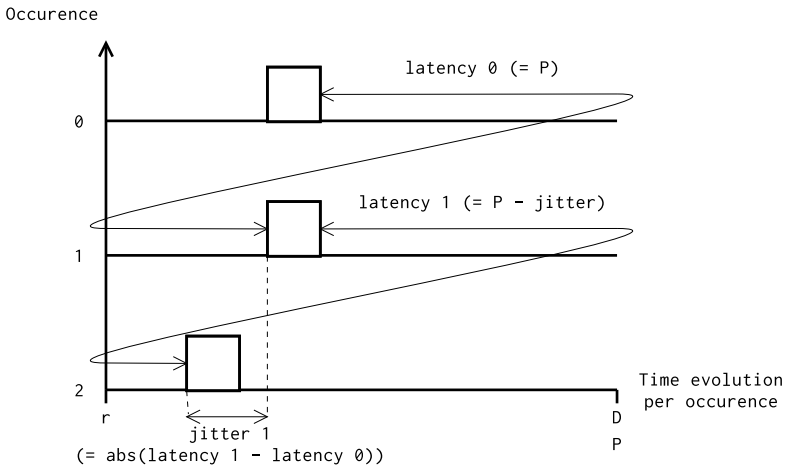


Figure 2.11 – Latency and jitter illustration

Figure 2.11 illustrates jitter and latency in scheduling.
In some cases, it may be possible to abstract from the notion of time and reason only on the order of execution, this is then called sequencing.

Sequencing

Refers to a set of ordered tasks where only the order counts, regardless of the execution dates. In a sequencing process, the notion of tasks is simplified.

In this work, we will be addressing scheduling and sequencing.

2.3.2.2 Task scheduling

Task scheduling is about calculating task execution dates. To do this, a time organization is applied to a set of tasks using resources while taking into account constraints. A scheduling aims to satisfy one or more objectives, it is on the basis of these criteria that the quality of a scheduling can be judged. Scheduling can have certain characteristics:

- Admissible: A scheduling is considered admissible if it respects all the constraints.
- Semi-active: A scheduling in which a task cannot be advanced without changing the sequence on the resource.
- Active: A scheduling in which no task can be started earlier without delaying the start of another task.
- Without delay: Scheduling in which the execution of a task must not be delayed. Scheduling without delay is active.

Scheduling policies can also be distinguished in class:

- Dynamic / Static: In a static scheduling, the weights (*i.e.* priorities) of the tasks are defined before the execution of the sequence. In the dynamic, these can vary.
- Idle or not: A scheduling in which dead times can be inserted instead of triggering available tasks, unlike a scheduling without delay.
- Preemptive or not: A task scheduling can be preempted.
- Online / Offline: Scheduling is offline if it is predetermined in advance. Otherwise it is online.
- Centralized / Distributed: Online scheduling is distributed if scheduling decisions are made by a local algorithm on each node. It is centralized when the same decision is made by a single node, whether or not the system is distributed. This node is called the privileged node.
- Optimal or not.

To check the schedulability, different methods exist:

- By simulation.
- Model checking (exhaustive exploration of the system's state space).
- The analytical approach.

Finally, the methods for solving scheduling problems are also divided into categories. The decision is correct if it guarantees the optimality of the solutions found, otherwise it is heuristic, when we observe that it is correct. We should also note that for a scheduling in which the order counts but not the time locations of the tasks, the term sequencing is preferred.

In a distributed system, additional difficulties arise. There is no offline centralized scheduling that is not pre-emptive and whose complexity is not at least polynomial if not NP-complete. Preemptive, it is at least NP-difficult [BRH90, HLV96], see the computational complexity.
In addition, in a distributed system, it is difficult to maintain a consistent view of the state of the system, especially when there are time constraints. This includes synchronization problems and clock drifts. Migrating a task can be expensive, which implies a real problem for the choice of assignment. Network scheduling has a very high impact, and constraints of reliability and availability of systems and networks appear. The precedence constraint is a time constraint, it is the most common constraint expressed in scheduling. This constraint can be expressed in the form of a priority relationship, which is noted for two tasks, τ_A and τ_B, with τ_A having to be executed before τ_B:

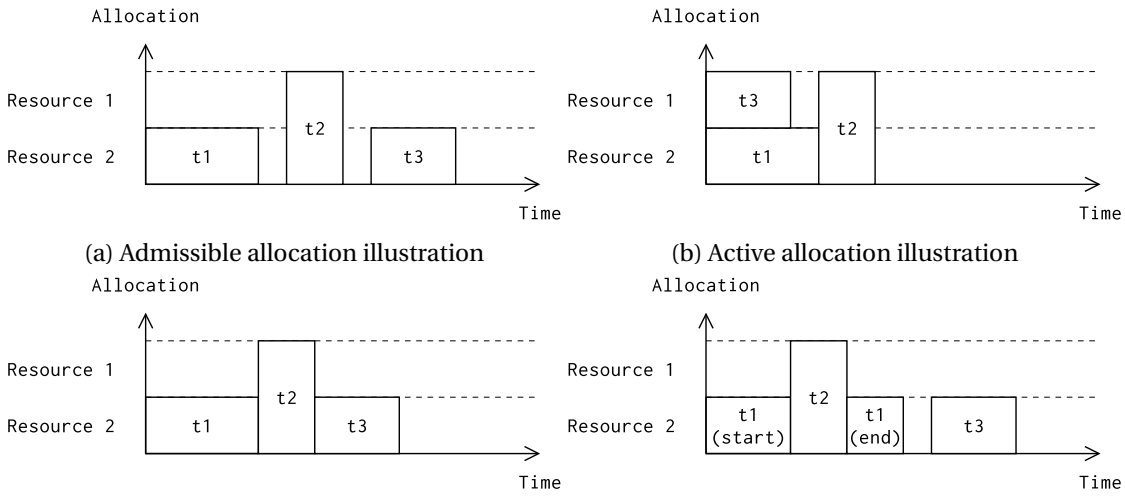
$$\tau_A \prec \tau_B$$

More formally, we will note, with t_S the reference date associated with the execution of a job of the task τ_S during a cycle (*i.e.* its temporal location), and C_s its duration:

$$t_B - t_A = C_A$$

More generally, when we talk about the temporal location of a task in relation to another, with $d_{A,B}$ a positive temporal constraint, $t_B - t_A \geq d_{A,B}$:

- $d_{A,B} = C_A$ — This is the case of the previously stated precedence.
- $0 \leq d_{A,B} < C_A$ — The constraint is weak, and only the starting of τ_B after τ_A is important.
- $d_{A,B} > C_A$ — The constraint is strong and imposes an additional delay.



(a) Admissible allocation illustration

(b) Active allocation illustration

(c) Semi-active allocation illustration

(d) Task preemtion illustration

Figure 2.12 – Ilustrations of scheduling general characteristics.

Task scheduling and the precedence constraint are illustrated in Figure 2.12

2.3.3 Computability and solutions generation

Scheduling problems, especially offline, are more or less complex, in the sense of computing complexity.

Computational complexity

Complexity theory is the theory of theoretical computing, which formally studies the amount of resources, especially time, but also memory space, that an algorithm needs to solve a problem. Problems can be grouped into complexity classes.
We speak of a deterministic or non-deterministic problem, depending on whether or not it is necessary to use a deterministic turing machine to solve them.
Although it can be shown that the deterministic problems that can be solved in polynomial time P are included in non-deterministic polynomial time NP, the opposite has not yet been demonstrated or refuted.
This is an open problem that we will not be addressed in this work. Most specialists speculate that NP-complete problems are not solvable in polynomial time, and we will admit this conjecture. It is possible to determine the complexity of a problem before knowing its solution.

Scheduling problems can have analytical solutions, which can be found in polynomial time, but very often the problems are NP-Complete. We will see that this is the case with scheduling problems in our work

NP-Complete problems are quite counter-intuitive, despite the formal expression of scheduling problems, as well as the possibility of quickly finding an intuitive solution to small problem instances, it is also possible not to find a solution within a reasonable time for the larger instances.

More formarly, NP is a class in the complexity theory meaning Nondeterministic Polynomial time. A problem is NP if it can be verified in polynomial time in relation to the size of the input *i.e.* it can be quickly verified that a candidate solution is indeed the solution to the problem. The NP problem is NP-Complete if all the problems of the NP class are reduced to it via a polynomial reduction; *i.e.* the problem is at least as difficult as all the other problems of the NP class, so it is not possible to quickly find the best solution[Wol06].

Finding the optimal solution to a NP-complete problem is extremely difficult. According to complexity theory, there is no exact solution. More generally, empirically finding solutions to small instances of problems is possible, but working with large instances requires setting up methods to approach solutions, often in the form of algorithms.
These algorithms are called heuristics. The heurisitics make it possible to obtain an acceptable solution to the problem under consideration, but it is impossible to guarantee the optimality of this solution [RNP10].

However, it is not possible to prove that a solution based on heuristics is optimal. [Wol06] and [BK06, BK] provide a good overview of all scheduling classes for which the minimum and/or maximum complexities are known.

Part II

State of the art

“If I have seen further it is by standing on ye sholders of Giants. ”

— Isaac Newton in a Letter to Robert Hooke, 1676.

“You stood on the shoulders of geniuses to accomplish something as fast as you could, and before you even knew what you had, you patented it, and packaged it, and slapped it on a plastic lunchbox, and now you wanna sell it. ”

— Ian Malcolm in Jurassic Park, 1993.

Table of Contents

3 Distributed simulation principles **43**

3.1 The birth, use and potential of Distributed Simulation 43

3.2 Benefits and drawbacks of simulation distribution 48

3.3 The specific characteristics of distributed simulation 50

Chapter 3

Distributed simulation principles

Contents

3.1 The birth, use and potential of Distributed Simulation 43
3.1.1 The foundation of the domain 43
3.1.2 The current use in industry 46
3.1.3 The future challenges of Distributed Simulation 48
3.2 Benefits and drawbacks of simulation distribution 48
3.2.1 Execution time 49
3.2.2 Model composability and Interoperability 49
3.2.3 Privacy and data integrity 49
3.2.4 Hybrid simulation 50
3.3 The specific characteristics of distributed simulation 50
3.3.1 Clock synchronisation 50
3.3.2 Time management 51

3.1 The birth, use and potential of Distributed Simulation

3.1.1 The foundation of the domain

Distributed Simulation (Dist-Sim) is a field of research close to both simulation and distributed computing. This field was described by R. M. Fujimoto in [Fuj90]. This field that was born in the 1970s, and is still active today, in particular R. M. Fujimoto regularly resumes his work to add recent advances [Fuj00, Fuj01, Fuj16, FBB+17].

Distributed Simulation

Distributed simulation is a field of computer science and simulation that addresses the execution of a simulation in a parallel or distributed manner on a computer architecture, usually composed of multiple computers connected by a network.

A distinction is made between parallel simulation and distributed simulation by the type of computer resources used.

- A parallel simulation will be on the scale of a computer, or even a room containing strongly linked computers.
- A distributed simulation will be on the scale of geographically distributed computers.

At the beginning, it was a question of accelerating simulations by using distributed computing. K. M. Chandy and J. Misra proposed to overcome the sequential execution of simulation, and to solve the problem of variable sharing through the use of message communication defined in [Den75] in a neighborhood [CM79]. K. M. Chandy and J. Misra then stated that empirically, "The correctness of a distributed system is proven by proving the correctness of each of its component processes and then using inductive arguments". However, we believe that a distributed system is not necessarily fully represented by its modeling, for example because of asynchronous phenomena, and that in these cases it is relevant to focus on scheduling, in addition to the validity of the models.

J. Kent Peacock, J. W. W. Wong and Eric G. Manning have taken up Chandy's work to provide a distributed taxonomy for simulation, and have begun to talk about synchronization, event-driven simulation, time-driven simulation and inter-process communication [KPWM79]. At this stage, the major problem of distributed simulation is the deadlock. K. M. Chandy and J. Misra suggested waiting for a deadlock to arrive before synchronizing the simulation to get out of it [CM81], while J. Kent Peacock, J. W. W. Wong and Eric G. Manning tried to avoid it.

Deadlock

In distributed simulation, simulation nodes usually have message queues to process, in order to send their own messages.

A situation in which all simulation nodes are waiting for a potential message from another node, blocking any progress in time from that node until the date of potential receipt of the message, thus blocking itself and the other nodes is called a deadlock.

P. F. Reynolds will then propose a distribution algorithm that no longer simply waits for

or prevents deadlocks from arriving, but will voluntarily block the simulation artificially on a regular basis [Rey82].

This alternative method, close to R. M. Fujimoto's lookahead, avoids causality problems in a short time, and is a way to treat deadlocks. However, as the authors say, it is "poor in those case where the relative frequency of communication is high", without however defining what a "high frequency" is. We know that in our case of distributed simulation of cyber-physical systems, this frequency is in the range of ten to one hundred hertz, which is acceptable.

Lookahead

The notion of lookahead was created to avoid causal constraints.

The lookahead means that a simulation node that has sent a message will not send one until at least its next lookahead.

For example, when a model X, with a lookahead of L_X, has emitted to another model Y a stamped message at the time T_s, Y infers that it will not receive any messages from X before $T_s + L_X$, and can therefore resume execution accordingly.

The lookahead, although introduced as an artificial blocking mechanism, can be justified by the behaviour of the modelled system. The following phenomena have been listed by R. M. Fujimoto as possible jusification for a lookahead:

- Limitation on communication — Following an event, when a system enters into an internal process before communicating again with the outside world.
- Physical limitation — Depends on the speed at which a system can respond to an event.
- Tolerance and inaccuracies — Depends on the ability of a system to return a correct response despite an error. For example, when the system entries are filtered.
- Non-preemptive behavior — When a system enters a phase that cannot be interrupted by an external phenomenon.
- Precomputing simulation activities — When a system is known to depend only on internal steps for a period of time.

Thus, if a system S_1 runs periodically, with a period P_1, it is possible to model it by M_{S_1}, having a period of P_1, and a lookahead $L_{M_{S_1}}$ also equal to P_1.
An M_{S_2} model of an S_2 system that receives a data from M_{S_1} at a given time t will know that up to $t + P_1$, nothing can come from M_{S_1}.

In parallel, J. Misra continues to work on introducing in [Mis86] the "null message" to

prevent deadlocks when a simulation node knows it has nothing to publish, so it should not block the others.

Null Message

The null message is a specific message that does not correspond to an activity in a simulation model, and was designed to avoid deadlocks.

It allows a simulation node X to tell the other nodes with a $T_{s,null}$ message that it will not send any stamped messages before $T_{s,null}$.

The null message can have a strong impact on the network load, which can be reduced when used with the lookahead.

Lookahead and "null message" were used by R. Righter and J. C. Walrand in [RW89], where the authors propose two time evolution strategies in a distributed simulation, those strategies will be described in subsection 3.3.2.

At this level, the field of distributed simulation is established, as defined by R. M. Fujimoto in [Fuj90].

3.1.2 The current use in industry

In 2008, S. Straßburger, T. Schulze and R. M. Fujimoto proposed to identify future trends in the field of Dist-Sim, based on a peer study [SSF08].

During this study, the authors identified the use of distributed simulation within organizations, both to simulate the organization as such, which is justified by their growing complexity, but also to improving the overall product life cycle of future products.

In addition, the authors point out that the most important application of distributed simulation is in the areas of joining and integrating computer resources for conducting complex distributed simulations as well as in the execution of distributed training sessions.

This application, which can be associated with system interoperability introduced by A. Tolk in [TM03], nevertheless lacked semantic interoperability at the time of the study. This semantic interoperability is therefore, in the authors' view, one of the main challenges that distributed simulation should have addressed in order to be commercially useful today.

Finally, the authors noted that there is a growing interest in the use of distributed simulation in the high-tech industry (*e.g.*, the automotive and aeronautics industries, as well as manufacturing). The interest of ISAE and Airbus through this doctoral thesis in simulation scheduling is the proof that the authors have identified a real possibility of using distributed simulation today. This approach is also found in other industries. These forecasts, which have proved to be correct, can be found in more or less recent papers, such as [KAK+10, Tay19].

However, the current use of distributed simulation depends very largely on its adoption by industry, and therefore on its economic interest, as highlighted by P. Lendermann, M. U. Heinicke, L. F. McGinnis, C. McLean, S. Straßburger, and S. J. E. Taylor in [LHM+07]. The

authors, who are references in the field of distributed simulation, have asked themselves the question of the usefulness of their field outside the academic aspect, for industries.

First and foremost, the authors argue that a general prerequisite for the use of distributed simulation in industry is the economic interest of this simulation, which can come either from the existence of a complex simulation that is more easily executable when it is divided into smaller, interconnected simulations. Or the existence of a problem that can only be solved by combining existing models. We will add that we also see the case of a simple simulation, but which must be played a large number of times to get interest, although it is closer to the academic world again.

Afterwards, the authors point out that in this first case it is difficult to identify a priori a simulation that will benefit from a distribution, in particular it does not exist at the time of the theoretical baseline study to ensure that one can benefit from the distribution.

We can find in the scientific literature different applications related to distributed simulation that are currently mature and industrially usable.

3.1.2.1 Speeding up execution time

In [VMB12], the authors use distibution to reduce limitations, including reducing the round-trip time of a simulation by incorporating several calculation cores into the simulation execution. This is one of the most common uses of distribution.

3.1.2.2 Enabling components reuse

In [NAT13], the authors address the problem of the use of complex and large simulation involving numerous models. The authors consider the question of model reusability in this type of simulation, and are relying on the interoperability between components in their demonstrator. The use of reusable components has many benefits, and is now common practice, and in some areas of engineering, a standard.

3.1.2.3 Parallelizing experimentations

We can find several distribution strategies in the scientific literature. If Dist-Sim can be obtained with the interoperability of independent components, we can also find examples of projection of the same execution that can have several instances on an architecture parallelizing these instances [DGST09].

3.1.2.4 Simulation of large and complex models

It is also possible to find examples of very large simulations, containing a very large number of simulation components distributed on a large number of computers [COM15].

3.1.2.5 Resource rationalization

Finally, the last currently available feature of the Dist-Sim is the possibility of rationalizing its resources, and consider in particular the use of grid computation allowing to allocate

simulation components [KWTM11].

3.1.3 The future challenges of Distributed Simulation

To the current academic and industrial uses, there are also opportunities for future use of simulation distribution.

3.1.3.1 From grid computing to cloud computing

Beyond the rationalization of resources, the industry is currently seeking a better use of its resources, which are often heterogeneous. Among the current research areas for industrialization of Dist-Sim, we can identify as the main area of research that of simulation allocation to distribute on cloud architecture [DM14, Nel16].

It should be noted that some research work already addresses technical issues and proposes solutions [HSVS14, RCV12], while comparing this approach with the more traditional grid approach [GGB15].

In addition, one case of application to avionics was treated with this approach [LLCS14].

3.1.3.2 From parallel experimentations to big data

The next step in using cloud computing is to manipulate the very large datasets that can then be produced. We can find in the scientific literature examples of integration of big data technology for the manipulation of very large numbers of results [Kac15], up to automatic learning [Tol].

3.1.3.3 From to Internet of Things integration

A further development path for the Dist-Sim is the consideration of a large number of hardware in the loop. In the scientific literature, the case is mainly mentioned with the birth of the Internet of Things (IoT) [WG17], but the need is broader and also impacts the industry [TSS+], including Airbus, and actors in aeronautical simulation [CR09, Hil18].

3.1.3.4 Real-time concerns

The Hardware-in-the-loop (HiL) integration is strongly linked to the RT aspect of a Dist-Sim, which is highlighted in [CR09, Hil18], and the main field is usually aeronautics [PSGk14]. However, it should be noted that this is an area that is under development, and for which much work is still in progress, up to the point of focusing on the basis of the Dist-Sim in RT [CS11]. Although the field is limited, we can see the emergence of the RT needs of the Dist-Sim when dealing with an important innovation, the digital twins [TCQ+18].

3.2 Benefits and drawbacks of simulation distribution

The industrial use of the Dist-Sim would not be possible without a clear advantage. Although mentioned earlier, we shall try to detail these in greater detail in the following.

These advantages have limits, or even negative impacts, that we shall also attempt to highlight.

3.2.1 Execution time

The most important advantage of the Dist-Sim is the ability to accelerate simulation execution. Among the execution acceleration, it is common to have renderings 200 to 300% faster [CBP02, MD17] but more spectacular results were also observed, with improvements to 1350 %, by parallelizing a simulation on 128 nodes [COM15].

However, this parallelization effort is often complex, and stakeholders may not have guarantees on possible gains [BD15], but it is increasingly common to find work to support the simulation distribution [AT17].

Finally, some studies highlight losses in execution speed due to latency and jitter phenomena in simulations [SO10], phenomena that are still poorly mastered nowadays.

3.2.2 Model composability and Interoperability

Another important advantage of the Dist-Sim is the composability of the models. Some Dist-Sim projects bring together many academic and industrial actors [TRC+14], from different countries as is commonly done by North Atlantic Treaty Organization (NATO) [BKB], and include the capitalization on their production and reuse of model blocks.

Interoperability can be linked to composability, especially when it comes to bringing teams from different countries together to work on simulations [BKB]. However, it should be noted that interoperability is now becoming more formalized, and a concrete set of interoperability levels is available in the scientific literature [TM03], while work exists in the interoperability proof [Mas06, TB07].

Interoperability and composability are still modern issues, and we see the application of recent technology such as containerization application in the Dist-Sim [vdBSC17].

Composability, however, has limits that can be reached very quickly in an industrial context, in particular on the amount of effort required to both make a reusable product, and make to reuse a product [Bau13, BEHK14].

Interoperability is also limited, and systems such as Run Time Infrastructure (RTI) designed with interoperability requirements may not comply with it [Gra03].

Finally, Dist-Sim is a type of Distributed System (Dist-Sys). The Dist-Sims inherit the more general problems of the Dist-Sys. Dist-Sys are more complicated systems than they appear, and distribution is a problem in its own. Even the knowledge of a domain in a non-distributed context does not allow to foresee the issues inherent to its distribution, and it is common to fall into traps, especially in terms of interoperability [RGO].

3.2.3 Privacy and data integrity

Privacy and data integrity are problems of the Dist-Sim that are not very well addressed today, although they are relevant to modern security issues.

There are some traces in the work of at the NATO [BKB], and some premises work on CERTI [BCSZ98].

We have not found any examples of a negative impact of a security breach in the scientific literature due to a Dist-Sim.

3.2.4 Hybrid simulation

Finally, some work has been able to take advantage of the composability of models to propose solutions inherent in hybrid simulations, with areas sometimes not very conducive to classical simulation [BKK02].

There are also works related to the geographical distribution of these hybrid simulations [MS, SMM06], or to the study of their complexity in the context of large simulations [ISKM+16, FDMPR17]

We did not find any examples of problems in hybrid simulation inherent in Dist-Sim in the scientific literature.

3.3 The specific characteristics of distributed simulation

Dist-Sim introduces a set of specific characteristics, in particular on the temporal and spatial management of the simulation.

Several notions of time were introduced by R. M. Fujimoto [Fuj98], these notion were introduced in subsubsection 2.2.1.2, as a reminder:

- Physical time — the reference time of the physical system.
- Simulated time — Representation of physical time during the simulation.
- Wallclock time — the time that can be described as real, during which the simulation is executed.

The existence of these different times raises challenges in terms of time coherence. In order to respect the principle of causality, time management must respect a strategy for time advancement and synchronization.

3.3.1 Clock synchronisation

Simulated time clocks are logical clocks, as introduced by L. Lamport [Lam78].

There may or may not have been a global clock for synchronizing local clocks, or directly a point-to-point synchronization, but overall the problem remains complex [KPMW80]. Clock synchronization is an important element of the Dist-Sim time advancement.

It should be noted, however, that research in the field of time advancing is not a stagnant issue since the formulation of the main principles [CM79, Jef85], and even today work is still being carried out to improve the Dist-Sim synchronization algorithmically [IMPQ17], or through hardware [LR09].

3.3.2 Time management

Concretely in the Dist-Sim, two solutions for advancing time are considered [Fuj00] that will be described in the following.

Proper description of the progress of time has been made in previous work done at the ISAE by JB. Chaudron [CSSA11, Cha12].

3.3.2.1 Conservative approaches

Conservative approaches try to maintain the correct order of execution of events at all times, based on P. F. Reynolds' solution. Each node of the simulation can regularly progress, waiting for its synchronization with the others.

To do so, the logic clock of each simulator in the simulation will move forward as long as it is certain that no events can occur to the simulator in the past.

However, this type of approach can create interlocking situations where all simulators are waiting for events from others. Several solutions have been proposed to solve this type of problem, including the addition of a lookahead [Fuj90] defined as a contract for which a simulator announces that it will not send any events. But also algorithms such as NULL messages [CM79].

3.3.2.2 Optimistic approaches

The optimistic approach is in opposition to the conservative approach: the time advancement of each simulator evolves without addressing temporal inconsistencies, until these occur, through a message received for a date in the past. The simulator then rolls back to the date of the inconsistency, and resumes its execution from that point [Jef85].

This approach is more complex than the conservative one, particularly because of the elements that simulation model designers must consider, and many rollback approaches exist and are being developed up to now [CBP02, MHSS05, VJ14, TPQ+17]

In simple words, the conservative approach avoids violating causality constraints while the optimistic approach allows it, but must provide a mechanism to recover. If the CPS simulation is connected to a physical system, recovery is impossible and only the conservative approach can be considered. The main objective of time evolution algorithms has been to optimize execution times, in particular by accelerating simulations.

3.3.2.3 Real-time

However, some work has addressed an advancement of time synchronized with wallclock time, allowing real-time simulation of the system, RT [ASC10, CSSA11, Cha12].

Other approaches are proposed in the scientific literature, usually with critical systems with high added value [PSGk14] by directly using services from an RT Operating System (OS), and low-level Input / Output (I/O) cards.

We will see that in the case of Airbus, both RT approaches are adopted, but each requires specific solutions. More precisely, the synchronization approach with wallclock time a posteriori allows to directly manipulate logical times before synchronization, simplifying the

manipulation of models, while the use of RT services implies the modeling of these services in logical time before making model manipulation.

Part III

Cyber-physical system simulation scheduling formalism

“In creating a software architecture, system considerations are seldom absent. For example, if you want an architecture to be high performance, you need to have some idea of the physical characteristics of the hardware platforms that it will run on (CPU speed, amount of memory, disk access speed) and the characteristics of any devices that the system interfaces with (traditional I/O devices, sensors, actuators), and you will also typically be concerned with the characteristics of the network (primarily bandwidth). If you want an architecture that is highly reliable, again you will be concerned with the hardware, in this case with its failure rates and the availability of redundant processing or network devices. On it goes. Considerations of hardware are seldom far from the mind of the architect.

So, when you design a software architecture, you will probably need to think about the entire system - the hardware as well as the software. To do otherwise would be foolhardy. No engineer can be expected to make predictions about the characteristics of a system when only part of that system is specified. ”

– Rick Kazman in Software Architecture in Practice, 1998.

“We’ll build bridges of love between two worlds. ”

– Pocahontas in Pocahontas II: Journey to a New World, 1998.

Table of Contents

4 Problem statement **57**

4.1 Cyber-physical system simulation scheduling 57

4.2 Simulation scheduling and real-time scheduling 71

5 Characterization of the model scheduling **78**

5.1 Strategy for formalizing the scheduling of Cyber-Physical System 79

5.2 Simulation Logical Architecture description 80

5.3 Simulation Execution Architecture description 86

5.4 Allocation of logical architecture on execution architecture 89

5.5 SLA requirements and verification 91

Chapter 4

Problem statement

Contents

4.1 Cyber-physical system simulation scheduling 57
4.1.1 Simulation frameworks, simulation middlewares, and their executions 57
4.1.2 Analysis of the main frameworks used in this thesis 59
4.2 Simulation scheduling and real-time scheduling 71
4.2.1 Similarities between simulation scheduling and RT 71
4.2.2 Conditions for an eligible scheduling 76
4.2.3 Scheduling tolerability 77

4.1 Cyber-physical system simulation scheduling

In this part, we present our main contribution, which is to generate, a priori, a valid CPS simulation scheduling. This involves defining what we mean by CPS simulation scheduling, and under which criteria a simulation scheduling can be considered valid.

In this section, we will define the scheduling of CPS simulations.

4.1.1 Simulation frameworks, simulation middlewares, and their executions

An impressive variety of simulation software is currently available on the market and at Airbus to simulate CPS. Each of these tools has its own specificity and meets certain specific needs.

The need we have to cover here is the correct execution of a CPS simulation, and more specifically the simulation of aircraft.

Aircraft are systems including controllers tightly interacting with the environment to stabilize the vehicles, which are defined as CPS [Lan12]. 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 knowledge 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 can be expressed for the integration.

These hypotheses allow to express the simulation scheduling CPS by the composition of the existing simulation components.

The softwares used to integrate simulation components are simulation frameworks and middlewares.

Framework

A set of libraries for software development. In the context of a computer simulation, the framework will define the execution, synchronization and communication between the components of the computer simulation.

Middleware

Middleware is a software that creates a network between different applications. This exchange network can take different forms, one of the most well-known being the use of messages. Middleware is responsible for enabling computer applications to interact, cooperate and transmit information to each other.

By definition, simulation scheduling is an emerging property of frameworks and middlewares, which can be abstract in a view containing a scheduler, scheduling the models as shown in Figure 4.1

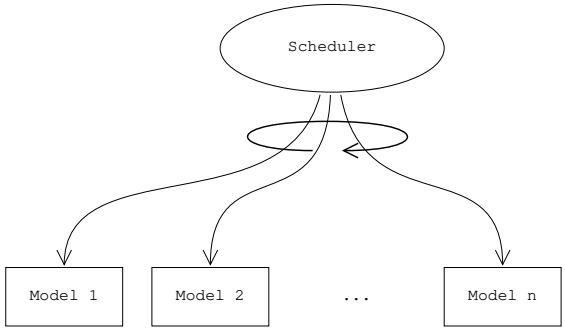


Figure 4.1 – High level view of simulation scheduling in frameworks and middlewares

The concepts of framework and middleware have grown without considering the control of the execution order of their components. Nevertheless, this order exists, there is always an implicit order of execution. Simulations are, *in fine*, software components, with sequences of instructions to be executed in a precise order, just like frameworks and middlewares.

4.1.2 Analysis of the main frameworks used in this thesis

All the potential simulation frameworks and middleware are far too varied to benefit from a rationalization and modeling of their behaviors. This would be equivalent to repeating and possibly renaming a computer software execution description formalism.

In order to propose a simpler and more usable solution, it is necessary to address a smaller group of frameworks and simulation middleware, and more precisely a subset of this group sufficiently representative.

Pragmatically, it is necessary to refocus on the main need to identify the frameworks and middlewares to be addressed.

4.1.2.1 The Airbus Distributed Simulation Scheduler

4.1.2.2 DSS introduction

Distributed Simulation Scheduler (DSS) is an Airbus user-space cross-platform framework, using AP2633 models.

AP2633

The AP2633 model is defined by Airbus in order to have a process to guarantee the interoperability of simulation execution. An AP2633 model offers an entry point, via a function, and a state machine using predifined global variables. The states of the state machine, with the entry point, allows the framework to schedule the AP2633 models. The states being:

- **LOAD** data loading, file opening, models configuration and connections creation ;
- **INIT** internal initializing ;
- **REINIT** force the model to a point ;
- **RUN** simulation loop ;
- **HOLD** simulation stop on a timestep ;
- **UNLOAD** output writing, files and connections closing ;

AP2633 models consume and produce data, these data are global variables, with a name used to exchange by homonymy, or by aliases depending the configuration.

4.1.2.3 DSS architecture

DSS is based on a two-tier architecture. A first level, high-level, which manages synchronization and exchanges in the simulation, and a second level, low-level, which implements the execution of the simulation. The high-level view of the simulation scheduling is illustrated in Figure 4.2.

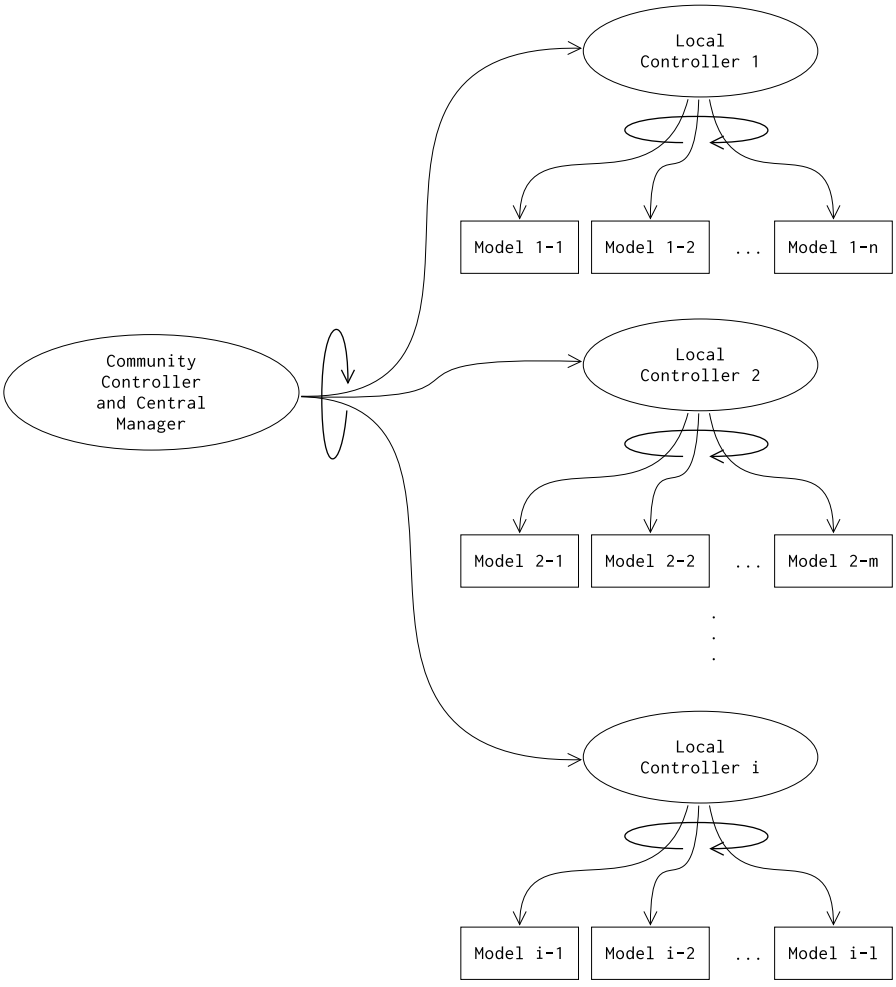


Figure 4.2 – DSS high level view of simulation scheduling levels

Those levels are implemented by two kinds of actors, illustrated in Figure 4.3:

- a **Global scheduler**: a Central Manager (CM) with a Community Controller (CC), computing short cycles and long cycles from information given by the local schedulers, choosing the synchronization period, and managing the local schedulers executions.

- **Local schedulers**: an Local Controller (LC), driven by the global scheduler, executing sequentially each AP2633 models provided depending on the configuration.

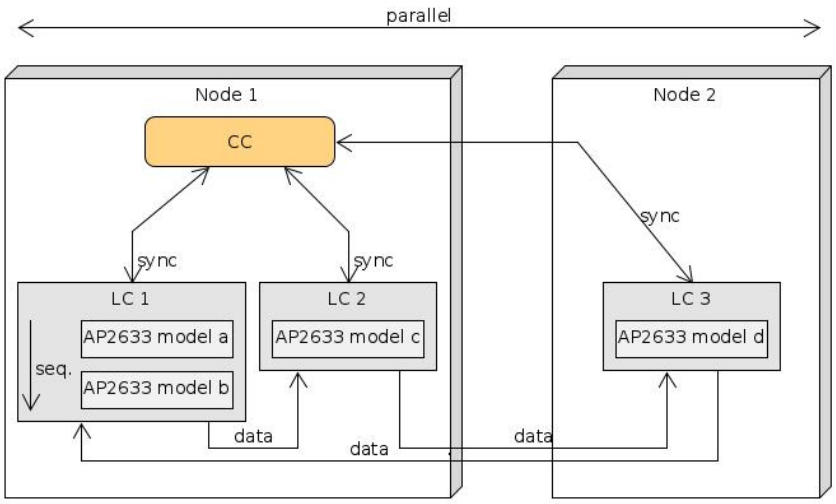


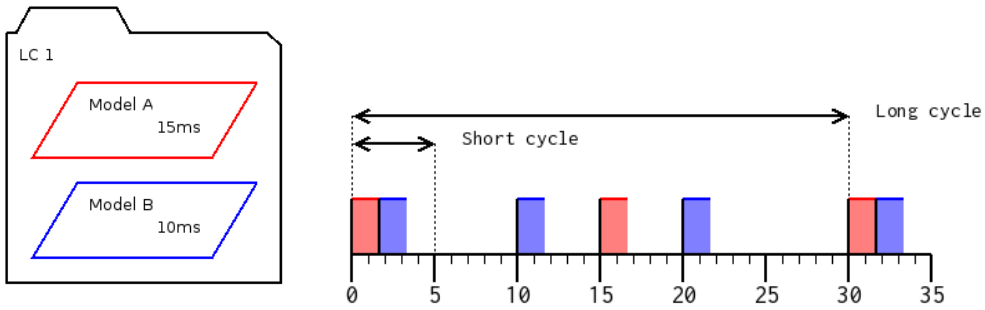
Figure 4.3 – DSS architecture

The major component of a DSS simulation is its configuration file. It contains the AP2633 models used, with their location and execution frequency. DSS use logical time to manipulate models, and is able to synchronize with wallclock time for RT needs.

4.1.2.4 DSS exection

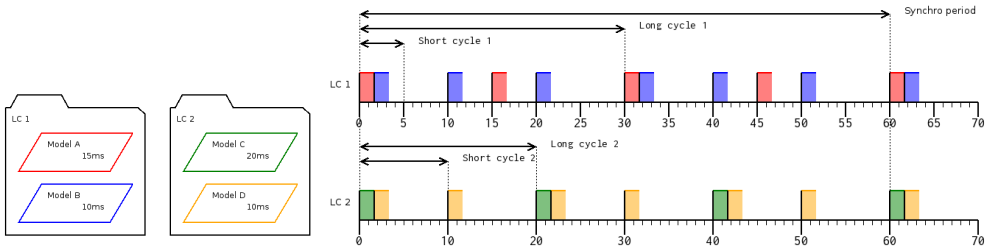
AP2633 models in a same LC are executed sequentially, AP2633 models in different LCs are executed in parallel.

The LCs periodically rerun the models, depending on the models periods. This execution leads to the concept of short cycle (Also called minor frame in the scientific literature), the maximum period needed to ensure each models can be run when needed (*i.e.* the tick). And the long cycle (Also called major frame in the scientific literature), the minimum period needed for a model execution pattern. Those concepts are illustrated in Figure 4.4 and Figure 4.5.



(a) partition of the models (b) temporal execution

Figure 4.4 – illustration of DSS short and long cycles and synchronization for one LC



(a) partition of the models (b) temporal execution

Figure 4.5 – Illustration of DSS short and long cycles and synchronization for two Local Controllers

Depending on the given configuration, the CC of DSS will choose the synchronization periods, Equation 4.3.

$$ShortCycle(LC) = GCD(model.period | \forall model \in LC.models) \quad (4.1)$$

$$LongCycle(LC) = LCM(model.period | \forall model \in LC.models) \quad (4.2)$$

$$SynchroPeriod(CC) = LCM(LongCycle(LC) | \forall LC \in CC.LCs) \quad (4.3)$$

Thus, a DSS distributed simulation with poor models distribution can lead to long synchronization period, and eventually degraded results.

4.1.2.5 The Airbus Simulation Framework for Integration and Design

4.1.2.6 ASPIC introduction

Simulation Framework for Integration and Design (Atelier de Simulation Pour l'Intégration et la Conception, ASPIC) is another Airbus framework using AP2633 models.

Unlike DSS, ASPIC is not cross-platform, and operates at the kernel level.

4.1.2.7 ASPIC architecture

ASPIC has a single-level architecture, integrated with the Linux kernel.

Periodic RT tasks are reserved by ASPIC, and AP2633 models are allocated to these tasks.

The AP2633 models of the same task must have the same execution period, equal to the Linux task period.

The choice of the allocation of the AP2633 models is made a priori on the tasks, in the same way as DSS via configuration files, and the Linux kernel then schedules these tasks. The high-level view of the simulation scheduling is illustrated in Figure 4.6.

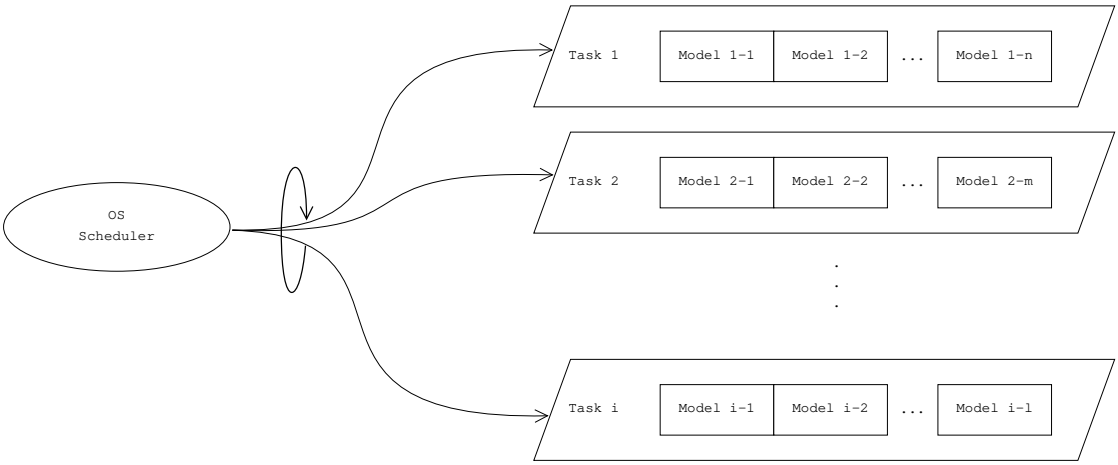


Figure 4.6 – ASPIC high level view of simulation scheduling

4.1.2.8 ASPIC execution

The execution of a simulation with ASPIC is mainly delegated to the Linux kernel.

Real-time Linux tasks are executed by a kernel scheduler, more precisely the Rate-Monotonic Scheduler (RMS).

Rate-Monotonic Scheduling

Rate-monotonic scheduling is a static real-time online scheduling, optimal for a system of periodic, synchronous, independent and on-demand tasks with a pre-emptive scheduler [LL73].
In Rate-monotonic scheduling, tasks with the shortest period are executed first, and lower priority tasks still being executed when a higher priority task is reactivated are paused, i. e. pre-empted, for the time required to execute the prioritised tasks.
A Rate-monotonic scheduling is permissible as long as its load is below 100%, which means that it has sufficient resources to meet all deadlines, i.e. to perform all periodic tasks, without any need to be woken up while it is pre-empted or running.

The execution of tasks being left to the Linux kernel, there is no synchronization, this execution is called asynchronous. Asynchronism is opposed to synchronism. It characterizes actions that do not occur at the same time. In the field of computer science, two phenomena are called asynchronous if they are not synchronized. It should be noted that there is a third concept specific to signals, which is found in telecommunications, beyond synchronism and asynchronism. Two signals are said to be plesiochronous if duplication or deletions are made to compensate for clock shifts.

Communication is also performed using the Linux kernel, by writing and reading in shared memory spaces at the time of model execution.

This method of execution is very efficient compared to DSS, however it is difficult to obtain a deterministic system.

Apart from the choice of allocating models to tasks, the integrator can vary the scheduling in different ways:

- Through the choice of sequence – The order in which the AP2633 models were placed in the task is respected when performing this task.

- With the addition of Offset – The integrator can add a delay before a task is awakened, so a task that should have been executed first can be postponed after the others have been executed, provided that the execution times of these other tasks are controlled to choose a consistent delay.

- Via the choice of task frequencies – Some AP2633 models are not limited to a single execution frequency, but may have different frequencies in a finite set, or in a bounded domain. Some models with no dynamics may even have a very wide acceptable frequency range. The integrator can manipulate the execution of the RMS by increasing or decreasing the frequency of a task when models allow it.

4.1.2.9 CERTI: an open-source RTI implementation

CERTI [BS02] is an implementation of the High Level Architecture (HLA) distributed simulation standard [II10].

4.1.2.10 CERTI based simulation architecture

HLA federates communicate through an RTI, and the federates use publication-/subscription-based mechanisms to exchange data. The RTI also ensure the time management between the federates, thus the synchronization which can, similarly to DSS, be synchronized with the wallclock time.

More specifically in CERTI, RTI is distributed between an RTI Gateway (RTIG) and RTI Ambassadors (sRTIAs). The RTIG, global, manages the federates creation and destruction, the data -publication/-subscription, and the time management. The RTIA, local, aims to lighten the work of the RTIG by performing the same requests as the RTIG would do, for which it can be locally resolved without going through the network.

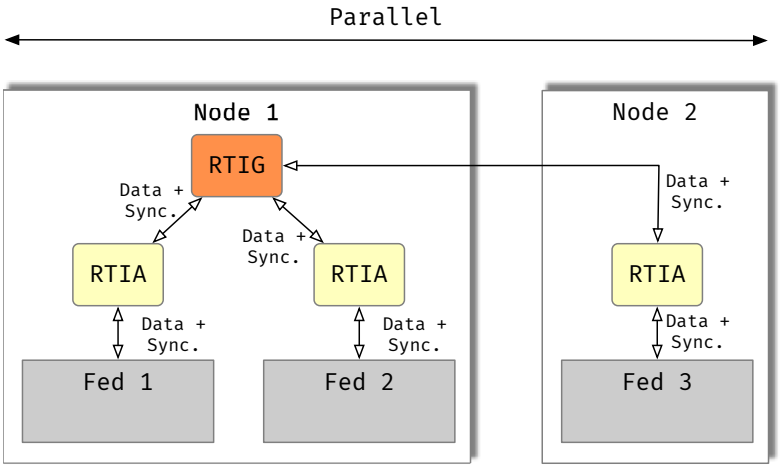


Figure 4.7 – CERTI-based simulation architecture

Our first contribution in this work was to provide a mechanism to CERTI for the sequential execution of models.

More specifically, a federate is a process generally with a single execution thread, parallelism being largely covered by the HLA standard. A process composed of a single thread is a set of instructions to be executed in a predetermined sequence.

If, in the same federate, instructions are executed not only for a single model, but for a model, and then for a second model, it is as if the two models had been executed sequentially. With this approach, we implemented a simulation framework called SEA proto-LP Allocation Nodes with Extensible inline Scheduler (SEAplanes), based on CERTI. More specifically, by using an entry point as well as DSS to schedule models, and the AP2633 state machine for model implementation, it is possible to find a behavior similar to DSS with CERTI.

The high-level view of the simulation scheduling is illustrated in Figure 4.8 and this architecture of simulation is illustrated in Figure 4.9. The complete description of SEAplanes will be done later in this work.

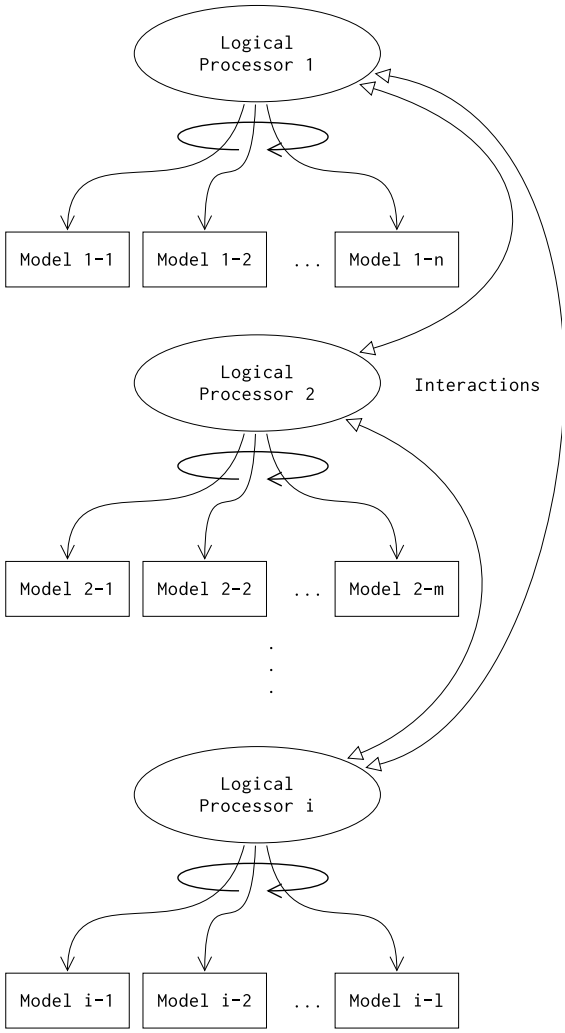


Figure 4.8 – SEAplanes high level view of simulation scheduling

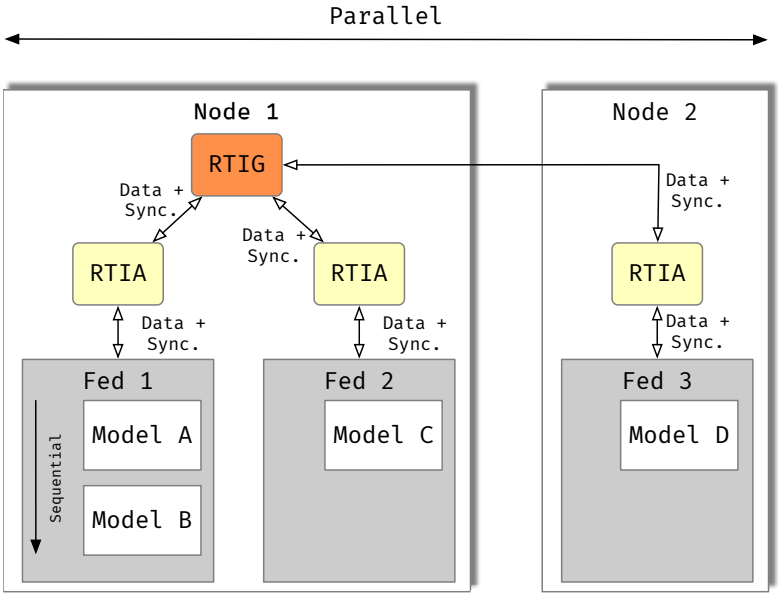


Figure 4.9 – sEAplanes-based simulation architecture

4.1.2.11 CERTI based simulation execution

The sEAplanes framework offers a similar behavior to DSS.

Federates are equivalent to DSS LC, running models sequentially, and the interaction between federates through RTI provides similar behavior to CC and CM.

The main difference is based on data exchanges, which are not a pairwise exchange between the federated states, as in DSS but through the RTI.

4.1.2.12 Technical comparison of the frameworks

4.1.2.13 Overview

- **DSS** is a two-level AP2633 models scheduler. Each AP2633 model is part of a simulation, and DSS allows them communicating using a data broker.
- **ASPIC** is a framework for distributing simulation models in tasks ordered by an OS.
- **sEAplanes** uses a standard from the Institute of Electrical and Electronics Engineers (IEEE), IEEE-1516, for software architecture. This standard defines methods and a framework to build global simulations comprised of smaller simulations.

4.1.2.14 Applications management

- **DSS** use a main software component called CM (Central Manager), that manage the dynamic negociation between the elements of the application set, for instance other applications discovery, or communication for data exchange. These applications are LC

(Local Controller). By default, they are asynchronous and do not communicate with the CM when the negociation is finished. A CM extension is the CC (Community Controller), used for the LCs synchronization. The execution in synchronous mode allowing to force a scheduling, and therefore to have a deterministic behavior, it is this mode that we address in our work.

- **ASPIC** allows a simulation integrator to distribute simulation models on OS tasks. The OS has the delegated responsibility to schedule tasks containing templates
- **sEAplanes** is implemented by a middleware, the RTI, as well as a library allowing the interaction of applications with this RTI. This middleware provides a set of services that are necessary to allow simulation entity synchronizing and exchanging data. Those entities are named federate in HLA, but renamed logical processor (LP) in sEAplanes to highlight their scheduling capability.

4.1.2.15 Services

- **DSS** provides a negotiation service, and depending of the chosen configuration, a synchronization service.
- **ASPIC** provides an allocation service, and guaranteed by the host OS of the real time constraints related to scheduling.
- In **sEAplanes** the HLA RTI provides services through an Application Programming Interface (API). These HLA services are:
  - Federation management ;
  - Objects management ;
  - Declaration management ;
  - Time management ;
  - Ownership management ;
  - Data distribution ;

  These services are encapsulated in the library, that provide unified interfaces to make the services easier to use, based on the simulation phases.

4.1.2.16 Collaboration with external products

- **DSS** can work in parallel with an external controller, asynchronously, managing external applications. DSS can manage itself the external applications, asynchronuously, if those applications are compliant with the AP2633 model.
- **ASPIC** cannot manage the scheduling of external components unlike DSS. However, it is possible to implement interfaces to exchange data with the outside world asynchronously. Finally, an actor who is familiar with OS programming will be able to read and write information in order to communicate with ASPIC. Nevertheless, this is not a standard behaviour.

- In **sEAplanes** being based on HLA, any HLA federates using the same RTI can theoretically communicate. Nevertheless, some technical choices in the simulations are left to the simulation designers, thus a well defined Management Object Model (MOM) might be necessary for good collaborations. During our work, we were able to verify this integration with JCerti and Ptolemy, as well as HLA Toolbox and Matlab.

4.1.2.17 Objects and interactions

- In **DSS**, LCs execute AP2633 models. Those models are exchanging data directly (via a shared memory) when in a same LC. The AP2633 models produce and consume a set of data. Each datum has a name, aliases, and is connected via array datum. Two models share a same datum when they use the same name (or alias). LCs exchange reports containing produced data for consumer. For two LCs in a same computer, the exchange can be via shared memory, or using the network. When two LCs are in different computers, the exchange can only be done using the network.

- In **ASPIC**, OS tasks execute AP2633 models. Those models are always exchanging data directly via a shared memory, asynchronously. Datum is similar to the DSS one.

- In **sEAplanes**, Federate Object Model (FOM) and Simulation Object Models (SOM) are written with an Object Model Template (OMT), describing shared objects, attributes, and interactions. FOM are contracts that affect the whole Federation. They can be either objects or interactions. Objects are persistent and we keep records of the exchange, while interactions are ephemeral. Those objects are managed through HLA services by the SEAplanes LPs.

4.1.2.18 Time management policy

- **DSS** forces a time stepped execution in logical time, that can be synchronized with a wallclock time, or executed in constraint time.

- **ASPIC** also forces a time stepped execution, but in RT using OS mechanisms.

- **sEAplanes** is compatible with different policies:

  - Event driven ;
  - Time stepped ;
  - Data flow ;
  - Mixed ;

  However, only time stepped policy has been implemented in SEAplanes for the moment. Nevertheless, the library is already open to the implementation of other strategies, and can interact with HLA federates using them. SEAplanes LPs are responsible of their inner logical time to schedule model, and synchronized this time with the federation through HLA services. Logical times can also be synchronized to wallclock time through HLA time management services.

4.1.2.19 Communications

- In **DSS**, communications can be done with Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). When using TCP, the communication is reliable while when using UDP, it's best effort. Local communication use user-space shared memory.
- **ASPIC**, communications are implemented by shared memory mechanisms at the OS level.
- In **SEAplanes**, communications use the network, or locally unix sockets. Communications can either be reliable with guarantees from the HLA standard, or best-effort. Message can also be timestamped or not for reordering when received.

4.1.2.20 Standard simulation

- In **DSS**:
  1. **Creating** the CM (and CC if synchronization). Creating the LCs, that register with the CM. Each LC received a list of already registered LC when registering.
  2. **Initializing** the data exchanges. Each LC contacts the LCs in its list, and negotiates the data exchanged.
  3. **Looping** the simulation. If synchronization, waiting for a signal by the CC. Computing, sending data and receiving others. If synchronization, sending end message to the CC.
  4. **Finalizing** and **destructing** the simulation entities.
- In **ASPIC**:
  1. **Creating** the OS tasks. Allocating the models on tasks.
  2. **Initializing** the shared memory for data exchanges.
  3. **Looping** the simulation. Executing lowest period tasks prioritary, preempting less priority tasks accordingly. For each model in each task, reading data from the shared memory, computing, and writing data.
  4. **Finalizing** and **destructing** the tasks and shared memory.
- In **SEAplanes**:
  1. **Creating** the federation, federate after federate. The federates enroll in the federation.
  2. **Initializing** the publication and subscription intentions. Establishing the time management policy, first synchronization, and registering of simulated objects.
  3. **Looping** the simulation. Time advancement, synchronization, computing, updating.
  4. **Finalizing** and **destructing** registered objects. Deactivating the time management policy. The federates exit the federation, and the last one destruct it.

4.2 Simulation scheduling and real-time scheduling

4.2.1 Similarities between simulation scheduling and RT

The analysis of previous frameworks allows to highlight recurrent patterns in the resource management of simulation frameworks.

4.2.1.1 Scheduling concepts

Regardless of the simulation architectures, it can be noted that there are two levels of execution, a higher-level in which entities are executed in parallel, with or without synchronization, and a lower-level, closer to the machine, in which models are executed in sequence, illustrated in Figure 4.10. This execution strategy introduced by A. S. Tanenbaum in [Tan87] as two-level scheduling is known to be efficiently processing scheduling that involves context switching. This scheduling category has been studied, and is now called partitioned scheduling, as opposed to global scheduling [CFH+04].

Global scheduling

In global scheduling, all scheduling resources are accessible to task jobs and a global scheduler can be seen as a single job queue, making it convenient for implementation. Global scheduling requires a load balancing strategy, a positive point being that there are optimal schedulers with this approach.
Compared to other approaches, global scheduling reduces response time and allows better use of resources.
On the other hand, the global scheduler has limitations such as job or task migration costs, synchronization problems, and the impossibility to predict the execution of the load balancer.

Partitioned scheduling

In partitioned scheduling, each task is assigned to a processor on which its jobs will be executed exclusively.
This is a scheduling class used in particular in critical industries, with solutions such as AUTOSAR or ARINC 653.
This use is due to the isolation between the cores, and the important study of scheduling frameworks, which often allows to pre-calculate the execution of scheduling, or to obtain a deterministic scheduling.
On the other hand, resources are often underutilized, and finding an optimal solution is equivalent to the problem of bin-packing, known to be NP-hard.

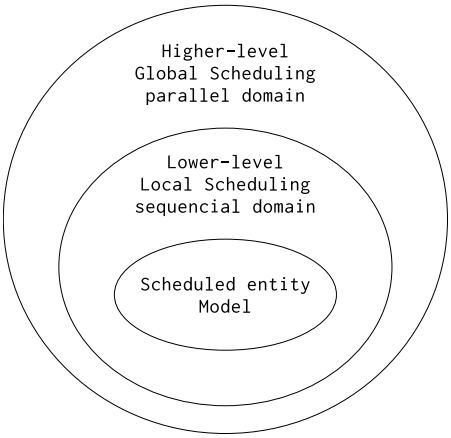


Figure 4.10 – Two-level scheduling

Global scheduling is now preferred to partitioned scheduling, especially due to the increase in computing power, and the ability to easily migrate tasks between processors. Nevertheless, in a strict context that implies determinism, a partitioned approach is more adapted as it reduces the complexity of interactions in a scheduling process. Links between the simulation frameworks and partitioned scheduling are listed in Table 4.2.

	DSS	ASPIC	SEAplanes
Global Scheduling	Community Controller and Central Manager	OS-based RMS	Logical processors interactions based on HLA time mangement service
Local Scheduling	Local controllers	Models sequences in tasks	Logical processors
Higher-level inter-communications	Network-based, UDP or TCP	OS-level shared memory	Messages, using HLA publication / subscription services
Lower-level intra-communications	User-level shared memory	OS-level shared memory	User-level shared memory

Table 4.1 – Table of links between the simulation frameworks of this work, and the partitioned scheduling

4.2.1.2 Schedulable components

In real time scheduling, the component that is scheduled is called a task. These tasks have been defined in the background, in subsection 2.3.1.

In simulation scheduling, the entity whose behavior most closely matches a periodic task is the model. Models too are executed periodically, with start and deadline. The major difference between a task and a model is the execution of an instance. In a generic task, the instance is often considered atomic, but for a model, several steps take place at each instance:

1. **Read** the data from the other models.

2. **Compute** the output from the current instance.

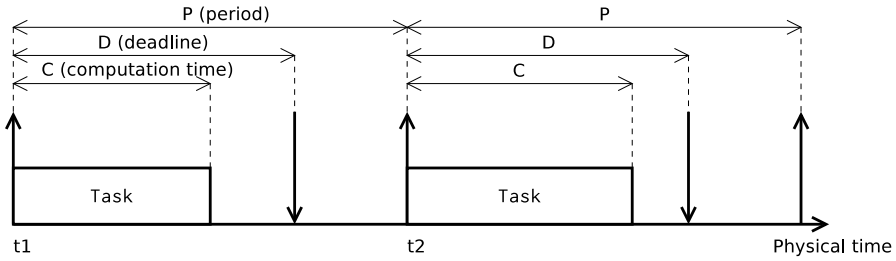
3. **Write** the results for the other models.

This relationship is illustrated in Figure 4.11.

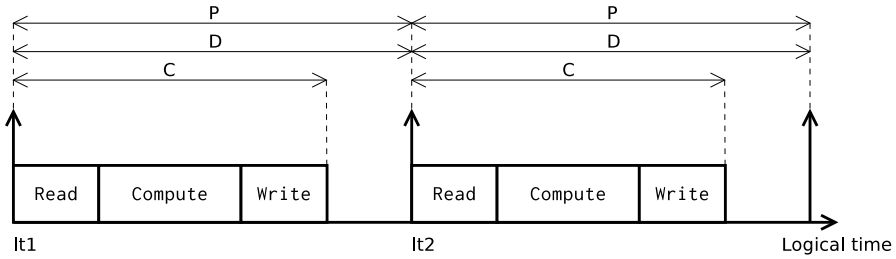
The challenge with this type of segmentation is that a model that has started, been preempted, and then continued to be executed may not have the same behavior if the preemption was done in the read phase, and may change the behavior of others if it happened during write. These concepts will be discussed in more detail in subsubsection 4.2.1.5.

4.2.1.3 Time relationship

RT, which is one of the particularities of Real-time scheduling, is not necessary, and not necessarily enforced in runtime in simulation scheduling. More precisely, most simulation frameworks execute and synchronize models in logical time, and ASPIC can be modeled to correspond to a logical time execution, as it is done with RMS and Architecture Analysis & Design Language (AADL). This removes the need to model complex software processes by considering only logical time processes, with a minimal adaptation for ASPIC.



(a) Task scheduling



(b) Model scheduling

Figure 4.11 – Relation between task and model scheduling

4.2.1.4 Precedence

In task scheduling, one of the major constraints is the precedence constraint. This constraint is defined in [Lam78] 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. A component A can produce a data for a component 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 Figure 4.12.

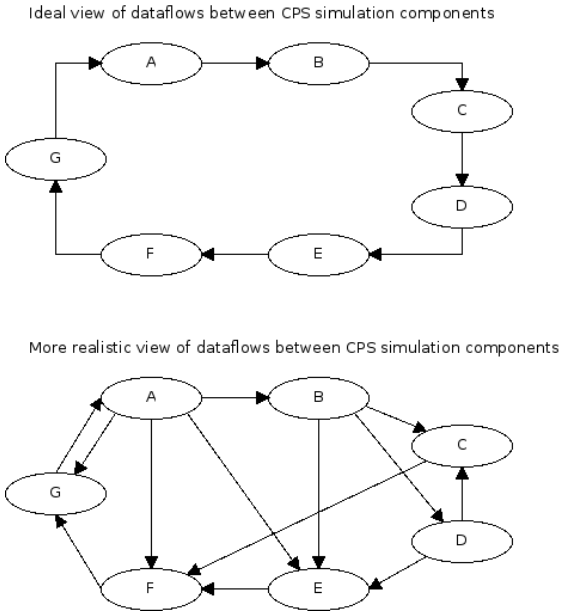


Figure 4.12 – Ideal and more realistic views of dataflows in a CPS.

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.

4.2.1.5 Preemption and overrun

Preemption is a characteristic of task scheduling. In subsubsection 4.2.1.2 these notions have already been mentioned.

In preemptive scheduling, a task can be interrupted during its execution, in order to execute another task. Most of the time, this mechanism of execution is based on priorities associated with tasks. Overrun happens when a task fails to meet its deadline. Preemption and overrun are illustrated in figs. 4.13 and 4.14.

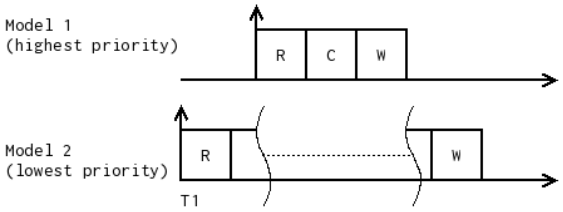


Figure 4.13 – Preemption in model scheduling.

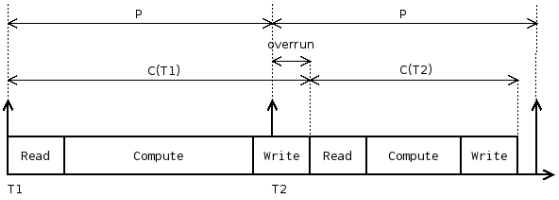


Figure 4.14 – Overrun in task scheduling.

In CPS simulation, preemption and overrun might be possible. As long as the latencies introduced in the previous subsection are respected. Nevertheless, if one requirement of the simulation is to be deterministic, one should avoid preemption and overrun as these two phenomena lead to simulations that are hard to reproduce.

4.2.1.6 Conclusion

The scheduling of RT tasks and models can be reconciled in a straightforward way, but there are some slight differences in terms of concept and scale equivalence. The equivalent concepts between RT scheduling and simulation scheduling are listed in Table 4.2.

RT multiprocessor scheduling	Simulation scheduling
Task	Model
RT	Logical time
From single-chip architecture to large scale signal-processing systems	From OS-based systems to message-passing distributed systems.

Table 4.2 – Table of equivalences between multiprocessor real-time scheduling and simulation scheduling

4.2.2 Conditions for an eligible scheduling

The scheduling of models being similar in its definition to RT scheduling, the acceptability of a solution to a scheduling problem can be expressed in the same way as that of a RT scheduling solution.

Thus, a solution to a simulation scheduling problem of CPS is acceptable if all simulation models can be run, respecting constraints as presented in subsubsection 4.2.1.4. More precisely, since these models are periodic, it is possible to validate the scheduling on a major frame.

The difficulty then comes from the fact that to choose an a priori scheduling, it must be modeled. To do this, it is necessary to identify the hardware that will be used for the simulation, identify the frameworks and middleware, as well as their behaviors, and characterize the models.

4.2.3 Scheduling tolerability

There is however a major difference between CPS simulation scheduling and RT scheduling.

A CPS simulation scheduling is useful to validate by system simulation, but these validations require to control the fidelity of the simulation, a concept not yet addressed from the scheduling point of view. In other words, simulation scheduling can be a valid solution to a scheduling problem, but has no interest from the point of view of a simulation user.

Thus, beyond the notion of acceptability of scheduling, it is necessary to be concerned with the validity of the result produced by scheduling.

Since simulation models are considered valid, this fidelity is based on the integration of the models, and in particular on compliance with the constraints of the simulated system described in the simulation.

In subsubsection 4.2.1.4, the relaxation of precedence constraint in a CPS simulation comes from the tolerance of the communications of the simulated CPS. In order to obtain the most useful scheduling, it is then necessary to identify a priori all the constraints that can be expressed on the simulated system, impacted by the simulation scheduling.

In the following, we will propose a method to manipulate these schedules. We will propose to formalize their descriptions, and set up a method to manipulate them.

Chapter 5

Characterization of the model scheduling

Contents

5.1 Strategy for formalizing the scheduling of Cyber-Physical System 79
5.1.1 Elements and categories in CPS simulation schedulings 79
5.1.2 Hierarchical modelling 79
5.1.3 Architecture Description Language examples 80
5.1.4 The categorization and allocation strategy 80
5.2 Simulation Logical Architecture description 80
5.2.1 Characterising cyber components 81
5.2.2 Characterising physical components 82
5.2.3 Extension to generic atomic model 83
5.3 Simulation Execution Architecture description 86
5.3.1 Schedulers and communications expressions 86
5.3.2 The sEA ADL 87
5.3.3 sEA representation of frameworks 88
5.4 Allocation of logical architecture on execution architecture 89
5.4.1 sLA partitioning 89
5.4.2 Partitions mapping 90
5.5 sLA requirements and verification 91
5.5.1 The precedence constraint 92
5.5.2 The latency constraint 92
5.5.3 The coincidence constraint 92
5.5.4 The affinity constraint 94

5.1 Strategy for formalizing the scheduling of Cyber-Physical System

5.1.1 Elements and categories in CPS simulation schedulings

A scheduling describes, through a given formalism, the execution of tasks, and the resource allocation over time, in order to be compliant with objectives, while respecting constraints. Generally, in computer architecture, tasks are threads or processes and the goal of scheduling is to minimise latency, maximise throughput or minimise response time.

In the previous chapter, we modelled the type of scheduling problem that is the simulation scheduling of CPS, and we related it to the elements of the RT scheduling. In this chapter, we will formalize the elements of CPS simulation scheduling, models, communication, and especially execution support.

The elements of the CPS simulation scheduling can be divided into two categories:

- A category related to the simulation, such as all the models that comprise a simulation, the links they maintain between them, and with the simulated CPS.

- A category related to simulation execution, closer to computer science, which covers model execution and their physical communications, as well as their interactions with the execution environment.

Formalizing these two categories together, while remaining open to extension, is a complex task.

5.1.2 Hierarchical modelling

Examples of modular and hierarchical languages already exist in the scientific literature.

One of these languages, particularly suitable for modelling, simulation and analysis of complex systems, is Discrete Event System Specification (DEVS). DEVS allows discrete event systems described by state transition functions and numerical approximations of continuous systems described by differential equations to be expressed.

In addition, there are many extensions to DEVS to address more specific problems. Especially in our case for simulations with paralellism, Parallel-DEVS is an extension that could have been adapted, allowing to verify the causality of events on a distributed system.

Nevertheless a limitation of DEVS is its complexity of use. Scheduling execution is an intrinsic property of a DEVS model, so in Parallel-DEVS, the time advance function of each DEVS atomic model would need to be redefined according to the choices that can be made over model periods, but also their placement over the simulation.

If the use of DEVS seems appropriate to us for the expression of the first category of the components of the CPS simulation scheduling, that related to simulation, it very quickly becomes complex to express the second category.

5.1.3 Architecture Description Language examples

At the Architecture Description Language (ADL) level, a good example of a language that seeks to model logical and physical elements together is the AADL. the core of the AADL is the declaration of components, and their implementation organized into packages. Components are categorized. Some components, close to logic such as data, thread and process belong to the software application category, while processors, memories, buses and devices belong to the execution platform category.

However, we did not adapt the AADL directly to our problem because of its limitations in system engineering.

The purpose of the AADL is to address embedded system modeling. The AADL allows you to express:

- The software architecture application software.
- The computer platform architecture.
- The architectural physical system.

These are three important elements in the problem of scheduling CPS simulation scheduling.

But the AADL does not cover the operational environment. We cannot express scenarios that a system must verify, or the actors interacting with the system. Thus it is complicated to express the behaviour of a CPS to be simulated that the simulation should follow.

5.1.4 The categorization and allocation strategy

We have decided to implement a strategy similar to the AADL categories. The approach followed consists in separating the architecture of the simulation models from the execution architecture of the simulation. Each of these architectures is then described in a formalism adapted to its specificities, while the formalisms are close enough to remain compatible.

A first formalism, the Simulation Logical Architecture (SLA), section 5.2, will therefore make it possible to express the elements of the simulation category. As DEVS seemed appropriate to the expression of this category, we have taken over elements of DEVS.

A second formalism, the Simulation Execution Architecture (SEA), section 5.3, will allow to express the computer architecture that will execute the simulation.

Finally, the compatibility between the two formalisms is ensured by the allocation, section 5.4, which allows to estimate the scheduling of the SLA models on the SEA platform, clarifying which requirements are verifiable with the allocation.

This allocation method is illustrated in Figure 5.1.

5.2 Simulation Logical Architecture description

CPS simulations are composed of two kinds of components, the physical components, and the cyber components.

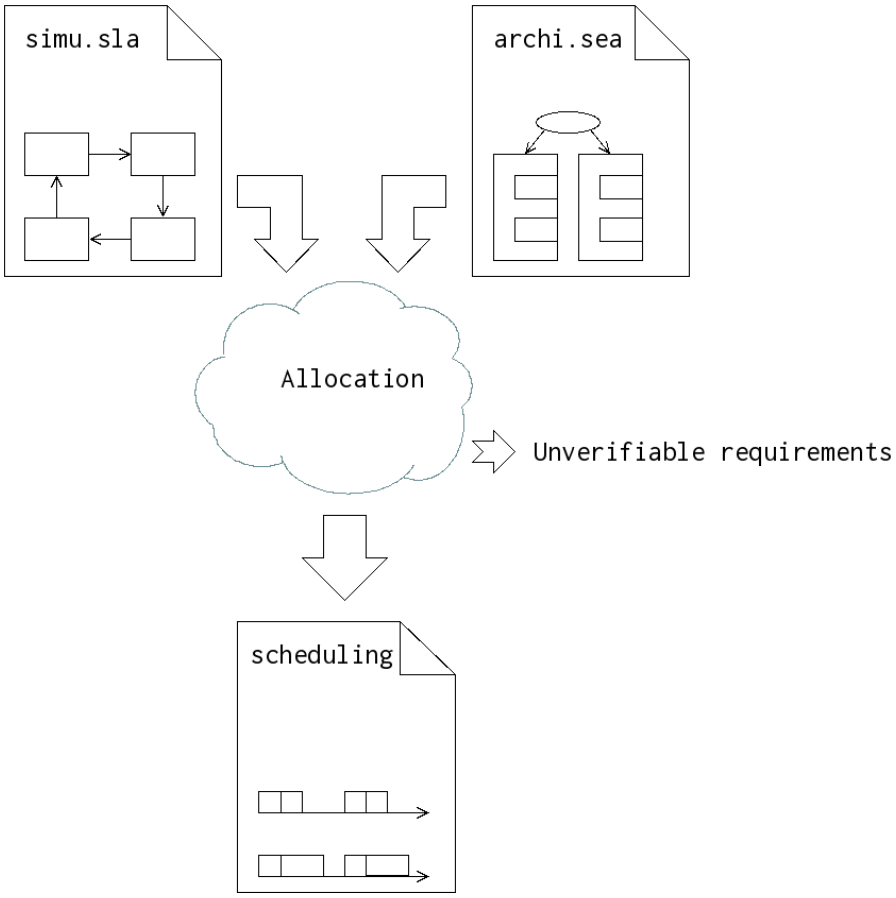


Figure 5.1 – Illustation of the CPS simulation scheduling workflow

5.2.1 Characterising cyber components

The cyber systems modelled in CPS simulations are, for instance Flight Control Systems (FCS), Human Machine Interface (HMI) or network devices (routers, switches...). These are processes executed by computers.

A process is a computational entity, often referred to as a task. Today, most OS does not allow direct communication between processes, in particular, this is the case of avionics [HJ12]. In these systems, the different processes are divided in space and time.

The partitioning in time exists because of the scheduling of processes, and the partitioning of space because two processes do not use the same memory without using OS mechanisms.

These notions of partitions are important for our components, they allow defining subsystem models which are totally isolated, apart from explicit communication.

In terms of formalism, this allows us to use a notion of buses for communication between our components, while the scheduling of tasks can be modelled by a periodical execution of models.

Finally, some models require a set of "previous" states to compute a new one. Considering

our components as periodical processes with total time and space partition, we can use the common definition of discrete system modelling, such as the state advancement, and output:

$$\begin{aligned} x_{k+1} &= F(x_k, u_k, t_k) \\ y_k &= G(x_k, u_k) \end{aligned}$$

With these characteristics, we can propose a formalism to express the cyber components:

$$cyber\ \ component\ \ =\ \ < I, O, S, S_0, \Delta_{in}, \Delta_{out}, f >$$

The set of inputs I are the data consumed by a component. For instance, for an altitude controller, the first input could be the current altitude, while the second input could be the altitude reference. It should be noted that these two inputs are not necessarily consumed together, we can imagine a component using the reference less regularly than the current altitude. Thus, inputs might be consumed by a component in our formalism with different frequencies.

The set of outputs O are data produced by a component. For instance, for an altitude controller, it could be commands for propulsion and elevator.

The states S and initial states S_0 of a component are different depending on the nature of the cyber components. For a controller, this is straightforward, but in general, its state is the vector of variables it manipulates, and the initial condition, their initializations.

The transition functions Δ_{in} are used to calculate new state based on previous ones, at the component frequency f. The output functions Δ_{out} use a set of recent states to compute the data to produce.

Certain processes do not use memory, the sets of states, initial states, and transition functions are empty, so their output functions only depend on the inputs.

5.2.2 Characterising physical components

According to their characteristics, continuous-time models are described by equations. These equations can be of different types, such as ODE, Partial Differential Equation (PDE), Differential Algebraic Equation (DAE) or Partial Differential Algebraic Equation (PDAE). In the following, we will focus on ODEs, for their simplicity in use and implementation, but the methodology is the same for the other classes of continuous-time models. ODEs are described by:

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

The ODE characterization for distributed simulation with real-time constraints is addressed in [CSSA16].

Since our simulations of CPS are subjected to real-time constraints, we will only focus on numerical methods that can adapt to the limitation of the computing resources (in space and time).

We consider time discretization of continuous dynamic systems. With a constant discretization interval of Δt, we have the following approximation:

$$\begin{aligned} x_{k+1} &\approx x_k + \Delta t \times F(x_k, u_k, t_k) \\ y_k &= G(x_k, u_k) \end{aligned}$$

Those characteristics allow us to define the following formalism:

$$physical\ component = < I, O, S, S_0, \delta_{in}, \delta_{out}, f >$$

The elements of a physical component are the same as defined for the controller in the cyber components. The differences are that the physical components consume and produce all their data at their own specific frequency f, with $f = \frac{1}{\Delta t}$, and they only need one transition function δ_{in} and output function δ_{out}.

5.2.3 Extension to generic atomic model

In order to simplify the generic atomic model of components, we do not address the problem of inter-compatibility yet, and consider a syntactical level of inter-compatibility, as described in [TM03]. The following formalism is largely inspired from DEVS, that we adapt to our context. We introduced operational information about the simulated CPS, for instance, the minimum and maximum latencies on data paths, the expression of periods simplify the estimation of time taken by a set of components in order to produce a data. Moreover, the definition of component in the early phases of simulation design can evolve quickly, and modifying the frequency of a component is faster than redefining the time advancement function. Furthermore, the coupling of components has to be flexible. At the component scale, input and output event are not considered, but ports. Two ports connected through a channel are producing and consuming data at frequencies defined by the connected component frequencies. Finally, components of simulation can interact at different rates, with different sets of other components. Thus we introduce sets of transition functions and output function, in contrary to DEVS considering one internal and one external function.

Note: in the following equations, lowercase parameters are single values, while uppercase ones are set of values.

We express the generic model of components c illustrated with some simplifications in fig. 5.2 as the following tuples:

$$c = < P_{in}, P_{out}, S_0, S, \Delta_{in}, \Delta_{out} > \tag{5.1}$$

Where:

P_{in} is the set of input ports, a port being data produced or consumed at a given frequency.

P_{out} is the set of output ports.

S_0 is a set of initial states.

S is a set of states, depending on the definition of the component.

Δ_{in} is the set of transition functions, with a transition function δ_{in} defined as how a set of data extracted from input ports changes the state of the component, at a given frequency.

Δ_{out} is the set of output functions, with an output function δ_{out} defined as how a value for an output port p_{out} is calculated from the current states and a set of data from input ports.

With transition functions Δ_{in} in Equation 5.1 defined as:

$$\delta_{in} = < I, S_{in}, s_{out}, \delta_{i^n \times s^m \to s}, f, t > \tag{5.2}$$

Where:

I is the set of inputs of the function.

S_{in} is the set of current and previous states.

s_{out} is the computed state.

$\delta_{i^n \times s^m \to s}$ is the main function, taking n inputs and m previous states in order to compute a new state: $i^n \times s^m \to s$.

f is the frequency of the transition function.

t is the function time budget.

And with output functions Δ_{out} in Equation 5.1 defined as:

$$\delta_{out} = < o, I, S, \delta_{i^n \times s^m \to o}, t > \tag{5.3}$$

where:

δ_{out} is the output function.

o is the calculated output.

I is the set of inputs of the function.

S is the set of current and previous states.

$\delta_{i^n \times s^m \to o}$ is the main function, taking n inputs and m states to compute an output: $i^n \times s^m \to o$.

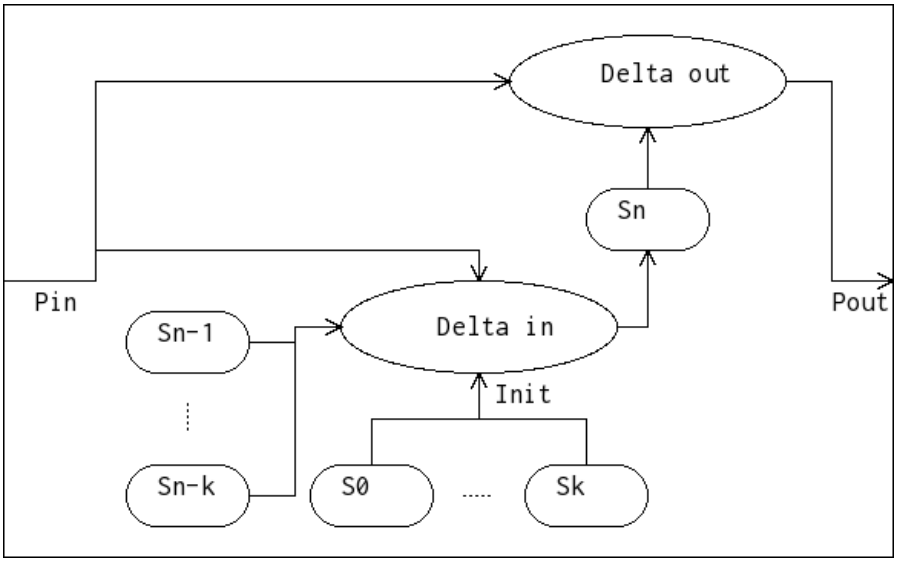


Figure 5.2 – Simplified view of a component.

t is the function time budget.

The component model defined is used in the SLA, and will be allocated to task in the execution architecture.

In order to be able to allocate this component, it is necessary to know its behavior and more precisely the time it will take to be executed.

It is not yet possible to have an accurate knowledge of this aspect, particularly because of the multi-processor architectures on which cache effects can occur. We have been inspired by AADL's strategy by adding a time budget to our components, which is how long it will take to be executed.

We propose to model a complete SLA as the set of its components, the channels between its components, and requirements, as the following:

$$\text{SLA} = < C, \Lambda, R > \tag{5.4}$$

Where:

C is the set of simulation components c.

R is the set of requirements.

Λ is the set of channels λ used by components to exchange data.

With channels Λ in Equation 5.4 defined as:

$$\lambda = < p_{in}, p_{out} > \tag{5.5}$$

Where:

p_{in} is the channel input port.

p_{out} is the channel output port.

We can verify some properties, for instance, every input port needed by components is supplied, or requirements are consistent. Requirements will be detailed in section 5.5

Nevertheless, the SLA does not allow verifying properties on the CPS simulation scheduling, we must now model the execution architecture able to run instances of the logical architecture of simulation.

5.3 Simulation Execution Architecture description

In this section, we want to define a model of SEA. This SEA can be implemented in several ways from a simple program with a single thread to a complex distributed architecture, thus our model must be abstract enough to represent this diversity.

An execution architecture of simulation can be viewed as a non-empty set of logical processors, with logical processors being able to execute the components defined in Equation 5.1, as periodic tasks.

Multiple models can be executed by a single logical processor, nevertheless, the logical processors respect the notion of time and space partition mentioned in subsection 5.2.1. In this work, we will call the process of binding multiple components to a logical processor a clustering. Running the SEA implies the distribution and clustering of components.

5.3.1 Schedulers and communications expressions

The tasks in a logical processor exist in a sequential domain, while logical processors exist in a concurrent domain. Moreover, depending on the implementation of the architecture, the concurrent domain might be a parallel domain, where logical processors can run totally simultaneously. We borrowed those notions of sequential and concurrent domain from the VHDL [BFMR12].

- Intraprocessor communications, direct between components.
- Interprocessor communications, occurring during logical processors synchronisation phases.

The different kinds of communication can have different natures, *e.g*. shared memory or network communication, implying a difference of performances.

The distribution/clustering of components on logical processors, and the communications create the notion of resources needed to express a scheduling, and the components are the schedulable tasks.

We are able to express the SEA as a two-level scheduler:

- A global scheduler: scheduling clusters.

- Local schedulers: scheduling tasks.

The global scheduler has no view on local schedulers tasks. There is no direct synchronisation between two tasks when they are on two different logical processors.

5.3.2 The sEA ADL

More specifically, we define a simple ADL considering logical processors and tasks, depicted in fig. 5.3, as the following:

$$\text{sEA} = < LP, gs, c > \tag{5.6}$$

Where:

LP	is the set of logical processors.
gs	is the global scheduler.
c	is the type of interprocessor communication between tasks in different logical processors, for instance shared memory.

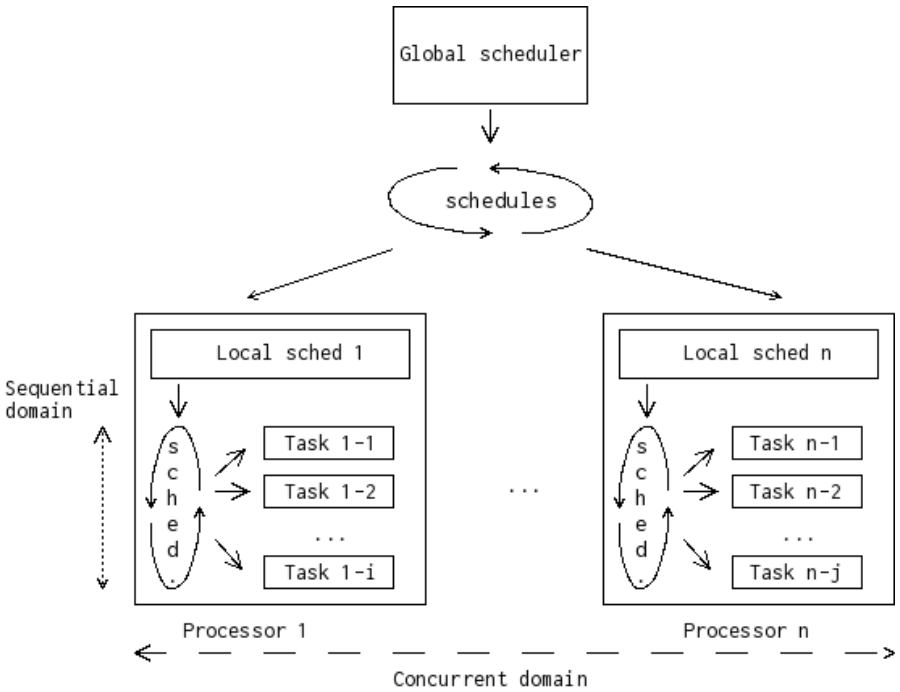


Figure 5.3 – The sEA with its double level of scheduling.

With the logical processors LP in Equation 5.6 as the following:

$$lp = < T, ls, c > \tag{5.7}$$

Where:

- T is the set of periodical tasks.
- ls is the local scheduler.
- c is the type of intraprocessor communication between tasks in the same logical processor.

The tasks T being RT tasks as defined in section 2.3.

5.3.3 SEA representation of frameworks

5.3.3.1 SEA representation of SEAplanes

In SEAplanes, models are executed in federates. Models instances in the same federate run sequentially. Models instances in different federates run concurrently. The logical processors and local schedulers are the federates, the local scheduling is the sequential execution on a federate.

The models are components of simulation hard coded or imported into a federate from a library, the library interface similar to Equation 5.2 and Equation 5.3.

The global scheduler is the synchronization of federates, achieved through HLA, and in particular through the RTIG and RTIAs. These same components allow the interprocessor communication.

Finally, intraprocessor communication is the shared memory on a federate.

5.3.3.2 SEA in DSS

The implementation representation of DSS as SEA is straightforward, the models are the AP2633 models, embedding implementations of components of simulation.

The logical processors and local schedulers are the LCs, the global scheduler, scheduling the LCs is the CC.

Intraprocessor communication is shared memory, interprocessor communication is Peer-2-Peer (P2P) network communication and shared memory, on synchronisation period defined by Equation 4.3.

5.3.3.3 SEA in ASPIC

The ASPIC implementation of an SEA is similar to the DSS one, the models being the AP2633 models and the logical processors and local schedulers being the OS tasks, and the global scheduler is the OS scheduler

Nevertheless, OS tasks are not able to schedule multi-peridioc models, thus when representing ASPIC with an SEA, LPs will only schedule models with a given period.

Finally, the intraprocessor and interprocessor communication are the same, achieved through shared memory. The impact on the SEA as such is minimal, although this rupture in the partitioning of communications makes it more complex to estimate communication times with ASPIC representation.

Table 5.1 summarize DSS, ASPIC and SEAplanes implementations of SEA.

SEA		SEAplanes	DSS	ASPIC
Sched.	Global	RTI	CC	OS scheduler
	Local	Federate	LC	Tasks
Comm.	Interproc.	Publication/Subscription	P2P on synchro period	Shared memory
	Intraproc.	Shared memory	Shared memory	Shared memory

Table 5.1 – DSS, ASPIC and SEAplanes implementations comparison

In the same way as for SLA, SEA must be translated from a mathematical model, has a format that is understandable for a computer.

5.4 Allocation of logical architecture on execution architecture

The distributing and clustering of schedulable components from an SLA on an SEA implementation is done in two stages: partitioning and mapping.

In this section, we discuss the impact of partitioning the SLA, and mapping this partition on an SEA.

5.4.1 SLA partitioning

Components have ports connected through channels. We want to divide our set of components into subsets, allocating a logical processor for each subset, and each one of the channels will be an interprocessor or intraprocessor communication when ported in an SEA, depending on partition and mapping.

In [DFRS92], the notions of partitioning and mapping on modular parallel architecture are treated in order to reduce the number of switching elements and to minimise communication times.

We adapted these notions in our work, with different constraints and objectives. For instance, due to the minimum and maximum latencies on channels, we are not looking for methods to reduce communication times, but for ensuring the consistency between constraints and partitioning/mapping.

The partitioning of SLA consists in splitting the set of SLA components into unordered subsets. From this partitioning, we are able to identify the type of channels that will be

instantiated between components, either intraprocessor for component in a same cluster or extraprocessor.

During this step, if we already have information about the SEA implementation, we can eliminate some partitions.

Fig. 5.4 illustrates some possible partitions for a single set of components. Ultimately, we can use set notation in order to represent the partitions. Regarding fig. 5.4, the partitions are:

- partition 1: $\{\{a,b,c,d\}\}$;
- partition 2: $\{\{a,b\},\{c\},\{d\}\}$;
- partition 3: $\{\{a\},\{b\},\{c\},\{d\}\}$.

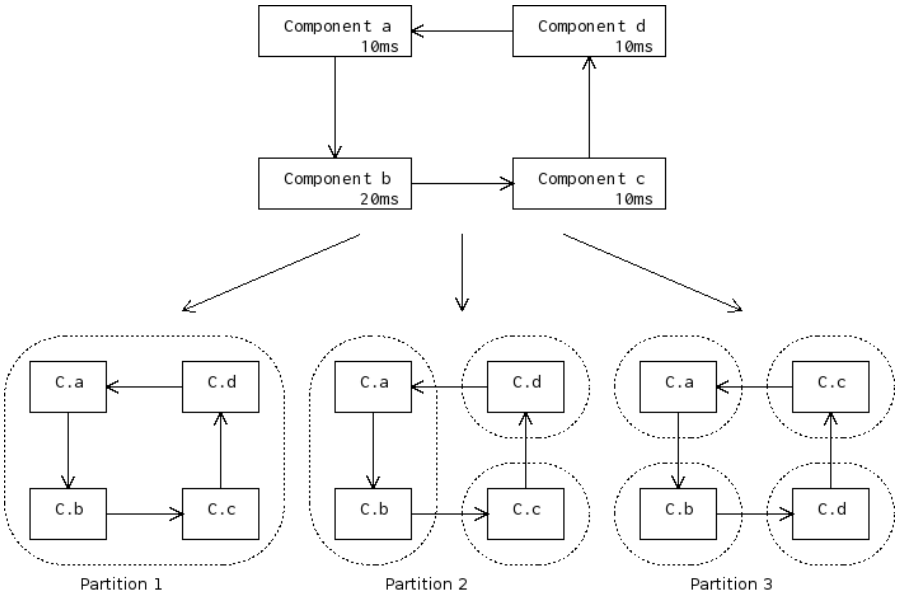


Figure 5.4 – Example of partitions from a single set of components

Nevertheless, these partitions are not yet linked to an execution. This occurs during the mapping to SEA step.

5.4.2 Partitions mapping

The set of tasks in an SEA logical processor is ordered. Mapping components from a partition to tasks from a logical processor implies the definition of a sequence. Considering partition 2, the ordering of set $\{c\}$ and $\{d\}$ are straightforward, but there are multiple solution for $\{a,b\}$: $\{a,b\}$ or $\{b,a\}$. This is where the problem of the algebraic loop discussed in subsection 5.2.2 is treated.

The ordered sets of tasks are sequentially executed by the local schedulers of logical processors, while the unordered set of logical processors is executed by the global scheduler.

To be more formal, we can describe the partitioning and mapping as the following function definitions:

- Let C be the set of sLA components.
- Let S be the set of C partitions, in the mathematical meaning, such that, for any S element s, the junction of s elements is C, and the superposition of s elements is empty.
- Let O be an ordered set of tasks.
- Let P be an unordered set of O.

$$partitioning : C \rightarrow S \tag{5.8}$$
$$mapping : S \rightarrow P \tag{5.9}$$

Depending on the sEA implementation, we are now able to verify requirements, such as the channel latencies requirements, and to adapt our partitioning and mapping.

Specifically, the decision of clustering components in the same subset is indirectly driven by the sEA implementation limitations, and component implementations, since the sLA does not consider execution. The mapping of partitions in the sEA implementation might lead to the identification of partitions that are impossible to execute.

For instance, different components will have different WCET. Depending on these WCETs, and the sEA logical processor capabilities, we are able to check that a given partition is executable or not. Another example is that interprocessor and intraprocessor communications have different costs. Once the sEA implementation is identified, we know the cost of these communications, we can then verify the latency requirements.

If the mapping is theoretically possible, but technically impossible, then we have to reiterate at the partitioning step.

5.5 sLA requirements and verification

In this section, we will detail the requirements introduced in subsection 5.2.3.

These requirements come from the simulation analysis of existing CPS. The two simplest constraints, called precedence constraint and latency constraint, are a direct projection of the application of scheduling on the simulation schedule, seen in subsubsection 4.2.1.4.

The coincidence constraint detailed below comes from the simulation analysis, and the affinity constraint of industrial practices.

This set of constraints is probably not complete, and we have implemented sLA and manipulation tools to be extensible.

5.5.1 The precedence constraint

The precedence constraint is the straightforward expression of the execution of one model before another, like the classic precedence constraint of real-time scheduling. This constraint is illustrated in Figure 5.5. In this figure, a system has been modeled as two models. In order to guaranty that this modelling will not affect the simulation output, we can express a precedence requirement.

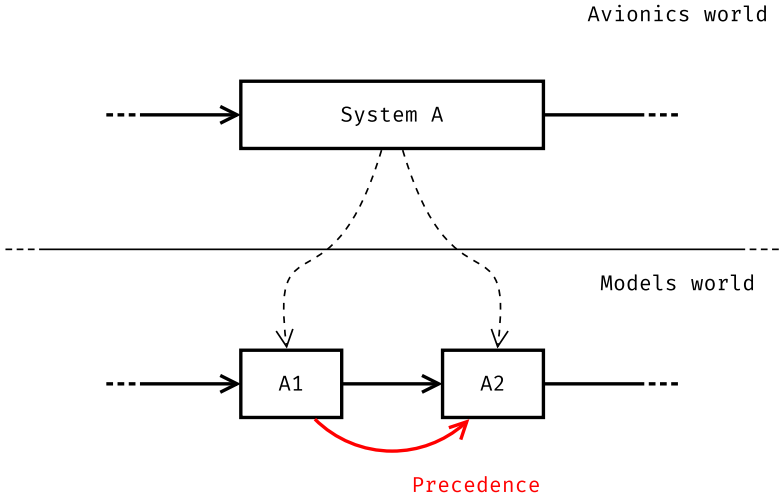


Figure 5.5 – Example of a precedence requirement due to system breakdown

5.5.2 The latency constraint

The latency constraint has been introduced to overcome the loop problem in the simulations. Such loops prevent the expression of simulation constraints based solely on precedence.

The introduction of a latency constraint allows to express a possible precedence, which can possibly be relaxed. The ability to express latency constraints makes precedence constraints artificial, it is preferable to express latency constraints that can be related to communications or asynchronism windows on the system, rather than giving precedence constraints that are easy to verify, but rigid.

This constraint is illustrated in Figure 5.6. In this figure, two systems communicating asynchrously have been modeled with two models. When the system A communicate with B, messages take from 10 to 50 milliseconds (ms) to be transmitted. We can express a latency constraint between the model A and B so the communication should be between 10 and 50 ms, without having to implement specific models for the communication.

5.5.3 The coincidence constraint

In the SEA, interprocessor and intraprocessor communications might have different costs on logical time latencies. Fig. 5.7 illustrates how different partitions can lead to different logical

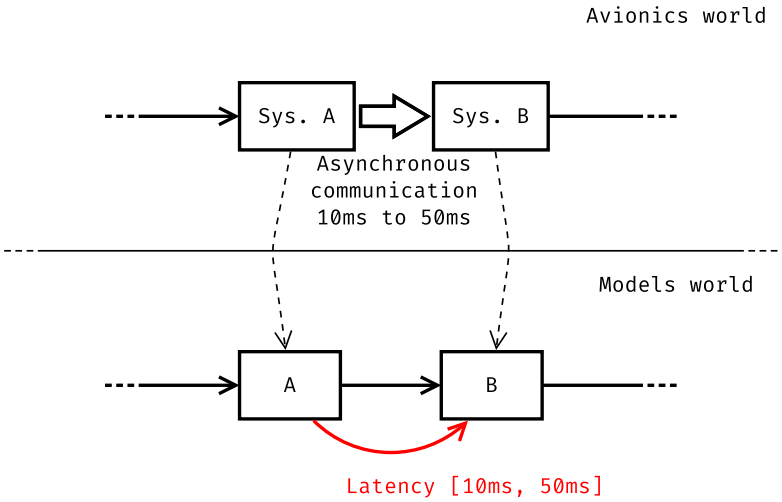


Figure 5.6 – Example of a latency requirement due to asynchronism

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.

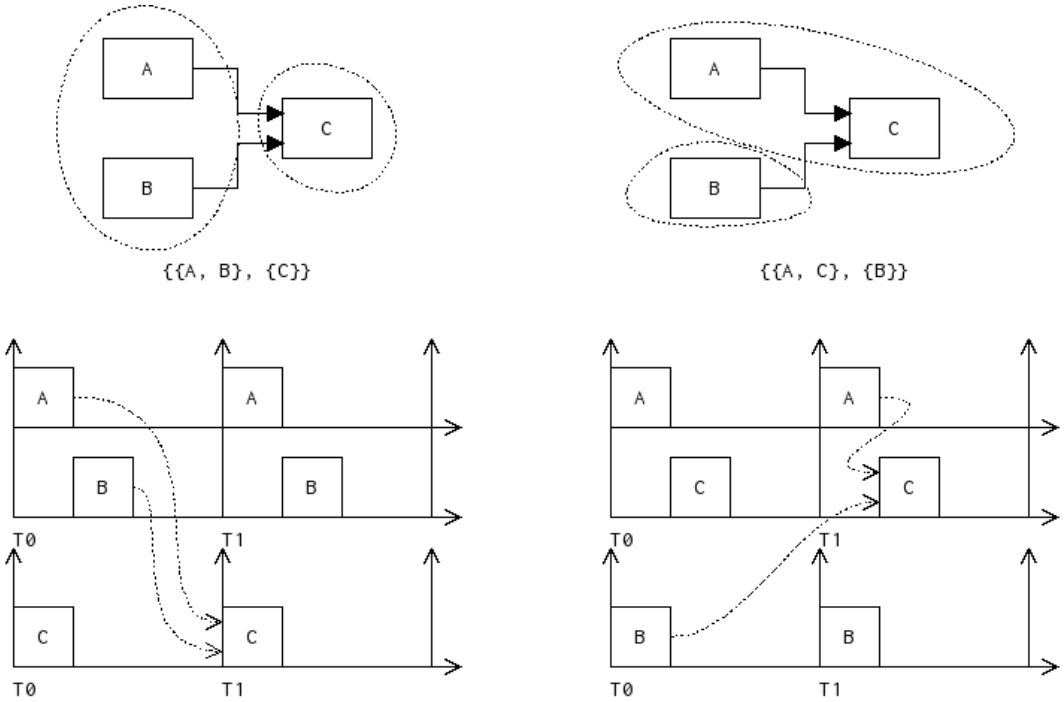


Figure 5.7 – Impact of the partitioning of components on data-flow latencies.

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, C production will be :

$$\begin{aligned} \forall lt,\ C[lt+1] &= A[lt]-B[lt] \\ &= lt-lt \\ &= 0 \end{aligned}$$

But for the second partition, C production will be :

$$\begin{aligned} \forall lt,\ C[lt+1] &= A[lt+1]-B[lt+1] \\ &= lt+1-lt \\ &= 1 \\ &\neq 0 \end{aligned}$$

If this subtracter has been designed without considering different delays, the second execution is invalid.

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.

5.5.4 The affinity constraint

The affinity constraint comes from industry practices.

It may be preferable to place components in the same logical processor, for example components using the same interface, which you want to place on a given node.

This constraint has no meaning in relation to simulation, but it allows to indicate needs coming from the use of this simulation. This constraint is illustrated in Figure 5.8.

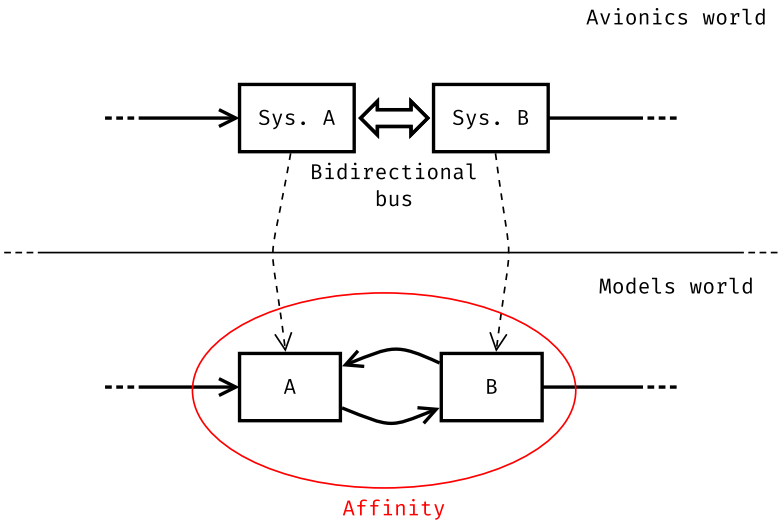


Figure 5.8 – Example of an affinity requirement due to systems interactions

Part IV

Solutions design, implementation, and applications

"On two occasions, I have been asked [by members of Parliament], "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?"... I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question. "

– Charles Babbage in Passages from the Life of a Philosopher, 1864.

"Trust The Computer. The Computer is your friend. "

– Allen Varney in Paranoia, 2004.

Table of Contents

6 Allocation tool and heuristics **101**

6.1 From methods to tools 101

6.2 Implementation of the allocation use case 106

6.3 Allocation heuristics 116

7 The Redundant ROSACE case study **121**

7.1 The original case study 122

7.2 The RROSACE case study 123

7.3 Implementation of the case study 127

7.4 Method and tool validation 142

Chapter 6

Allocation tool and heuristics

Contents

6.1 From methods to tools 101
6.1.1 sLA and sEA computer-readability adaptation 101
6.1.2 Existing MBSE adaptation approach 104
6.1.3 Full language definition approach 104
6.1.4 Hierarchical structure language use approach 106
6.2 Implementation of the allocation use case 106
6.2.1 Front-end modules 107
6.2.2 The allocation tool modules 114
6.2.3 Back-end adapters 115
6.3 Allocation heuristics 116
6.3.1 Greedy heuristics 117
6.3.2 Simulated Annealing 118

6.1 From methods to tools

The objective of this chapter is to propose tools for implementing the method described in the previous chapter.

It is essentially a question of translating the previous formalisms into computer-readable formats that can be manipulated and used.

6.1.1 sLA and sEA computer-readability adaptation

A computer program that reads sources in some languages, and translates them into another language, a target language, is called a compiler [ASU86], see Figure 6.1.

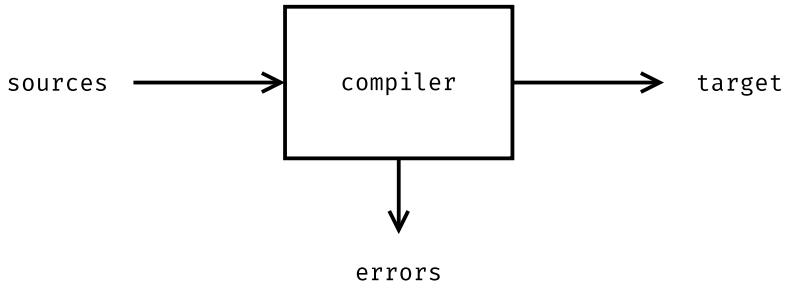


Figure 6.1 – Compiler system

The compilation is a set of operations, illustrated in Figure 6.2. The **lexical analysis**, converting a text into tokens, for example the “component” text in a “c” symbol. The **syntax analysis**, consisting in interpreting the structure of the previous tokens, at this stage the linear structure of tokens and represented as a tree. The **semantic analysis**, more advanced than syntactic analysis, completes the syntax tree of the previous operation by performing verifications, resolution and assignment. The **code generation**, transforming the syntax tree enriched from the previous operation into code that can be interpreted by a machine.

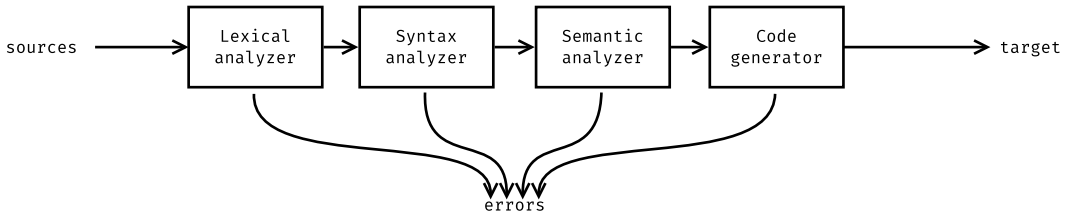


Figure 6.2 – Compiler processing

Nowadays, compilers are a modernized technology that can be multi-language, multi-target, thus compilers are generally divided into a three-stage compiler structure, as illustrated in Figure 6.4 The **front-end** handles the first compilation operations, lexical, syntax, and semantic analysis. The output of the front-end is an Intermediate Representation (IR). The **middle-end** optimizes the IR, independently of the final execution architecture. The **back-end** translates the optimized IR into target-dependent code.

In chapter 5, the allocation method is described as taking different languages as input, and outputting a computer executable scheduling.

The allocation is similar to a compilation, the link between the allocation tool and a compiler can be done, especially since the output is platform dependent. This link is illustrated in Figure 6.3.

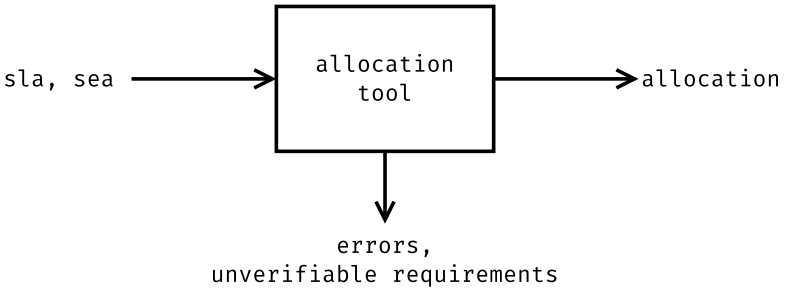


Figure 6.3 – Allocation system

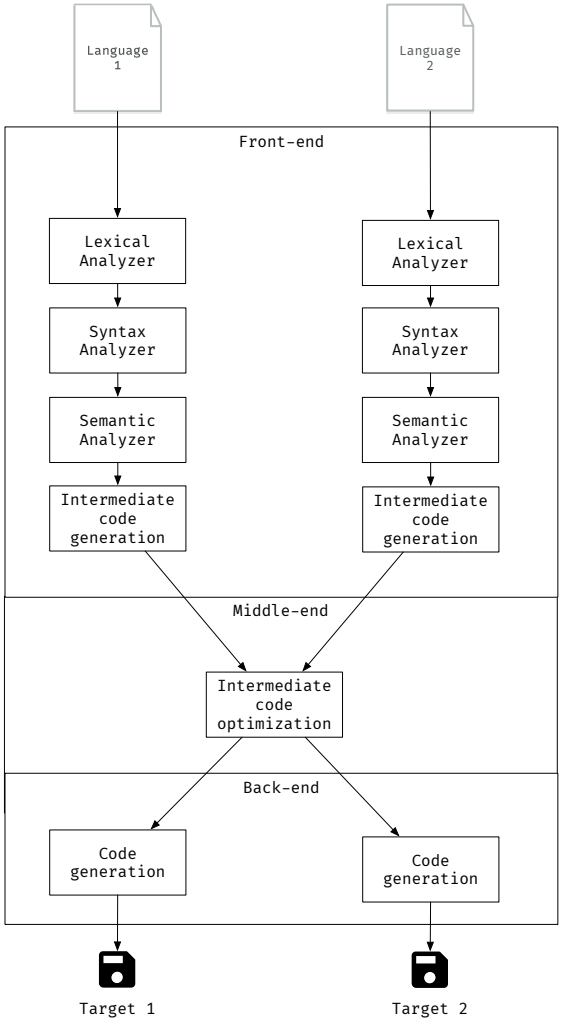


Figure 6.4 – Three-stage compiler structure, adapted for multi-language, multi-platform compilation

The implementation of the allocation tool requires certain steps to be implemented.

First of all, it is required to put the sLA and sEA in the form of languages that can be understood by a computer. Then it is needed to implement the front-ends, *i.e.* to create the analysis tools up to the intermediate generation of scheduling. A next step could be the implementation of the middle-end, the optimization of this intermediate scheduling, but it is not necessary, and in a pragmatic concern to save time in creating a Proof of Concept (PoC), it will be skipped. Finally, the last step consists in creating the back-ends, in order to translate the intermediate scheduling into scheduling for target architectures.

Each of these steps and the choices made are described in the following subsections.

6.1.2 Existing MBSE adaptation approach

The first step in creating the allocation tool is to adapt the sLA and sEA languages into languages that can be understood by a computer.

Since the languages are inspired by Model-Based Systems Engineering (MBSE), one approach could be to extend the previous MBSE to add missing features.

Adapting DEVS seems extremely complicated, modifying the sLA to not take into consideration execution at the time of expression compromises the use of existing tools, and there is no obvious approach to integrating the sEA and then allocation.

AADL [FG13] seems *a priori* more permissive. The components and channels of the sLA can be expressed quite directly as tasks, and there are similarities between processes and LP, as well as on allocation. In addition, some allocation work seems to make it easier in the future to improve the tool [HZPK07]. The main difficulty we expressed in the chapter 5 is the impossibility today to express operational constraints, and therefore to cover the requirements part of the sLA, it would be necessary to extend AADL on this point.

Other approaches by Unified Modelling Language (UML), Systems Modelling Language (SysML) and Modeling and Analysis of Real-Time and Embedded systems (MARTE)/Clock Constraint Specification Language (CCSL) [RJB04, FMS14, AM08] were investigated. In the main, UML is clearly not adaptable to real time constraints, and MARTE/CCSL does not allow an easy expression of sLA, despite the existence of promising work in the field of CPS [Mal15]. SysML is a good candidate covering all expression needs, but the work required to adapt a tool seems too long for a doctoral thesis PoC. In addition, SysML does not solve the scalability problem that exists with aadl.

In summary, the MBSE adaptation approach is possible with AADL and SysML, but hazardous because the adaptation times of existing tools are likely to be long, and there is no guarantee on the scalability of the tools.

6.1.3 Full language definition approach

A different and more interesting approach consists in implementing our own computer languages. This is an approach that has, for example, been chosen by OpenSim for OMNeT++, an extensible, modular, component-based simulation library [VH08, noa19]. OMNeT++ uses a specific language, NEtwork Description (NED), to express the topology of a network, a very simple NED file is provided to present the possibility of using a specific language in Listing 6.1.

The manual implementation of computer languages for SLA and SEA requires generating all the blocks of the compilation, retargeted for allocation, as shown in Figure 6.5.

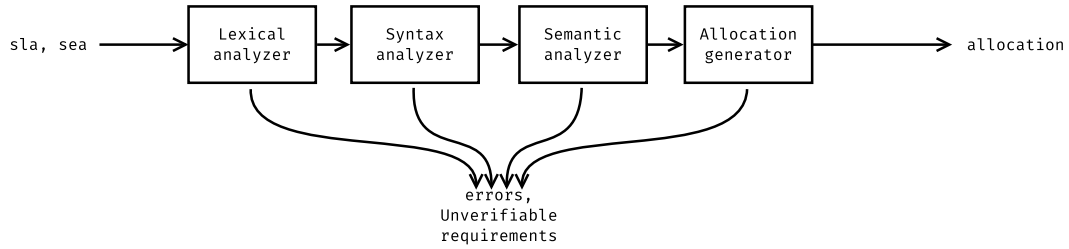


Figure 6.5 – Allocation processing

This implementation of modern compiler is now accessible as tools exist to simplify this task, this is the case of flex and bison, used by OMNeT++ for ned. [Lev09] Flex allows the generation of a lexical analyzer, while bison builds a compiler.

In addition, the Low Level Virtual Machine (LLVM) project is now a very mature project allowing the creation of compilers. It is certainly possible to develop your front-end with your homemade language as it is done in some projects [PS15], while some projects such as in [TGP07] were LLVM is used to address exotic architecture, here hardware circuits. Nevertheless, the difficulty of this approach makes it time consuming.

Listing 6.1 – Example of a NED file

```
simple Component
{
    gates:
        input in;
        output out;
}

network Components
{
    submodules:
        a: Component;
        b: Component;
        c: Component;

    connections:
        a.out --> { delay = 50ms; } --> b.in;
        b.out --> { delay = 50ms; } --> c.in;
        a.in <-- { delay = 100ms; } <-- b.out;
        a.in <-- { delay = 50ms; } <-- c.out;
}
```

While this approach is less time consuming than MBSE adaptation, it is still quite time-consuming, since the 4 blocks of Figure 6.5 would have to be implemented for each language, in addition to the allocation itself.

6.1.4 Hierarchical structure language use approach

A third approach consists in implementing the allocation tool using a language manually, but with the reuse of an already-existing language. For this purpose, many projects use formatting languages.

Such an approach avoids having to implement the first blocks of the compiler, as shown in Figure 6.6. However, there are technical limitations: Indeed, the use of a formatting language requires directly expressing SLA and SEA in the form of a tree, which is not really the case for SLA, closer to an oriented graph. The formatting language involves a verbal overhead, many characters not necessary for SLA and SEA must be expressed in order to fully meet the needs of the formatting format parsers. Finally, many language features do not exist in formatting languages, so it is not possible to check certain validities such as the type of time or the correct use of numbers at the time of parsing. Using a formatting language therefore requires manual semantic checks, especially on data typing.

Despite these limitations, the significant time saving in the implementation of a PoC makes this solution the chosen one.

Each language has its own particularities. Within the framework of our implementation, it is the popularity of the language that we believe to be the main criteria. The PoC should be easily understandable, and Airbus should be able to industrialize it in a simple way.

Thus it is eXtensible Markup Language (XML) [BPSM+08], widely used in web, mobile, office and server technologies, that we have chosen as our language.

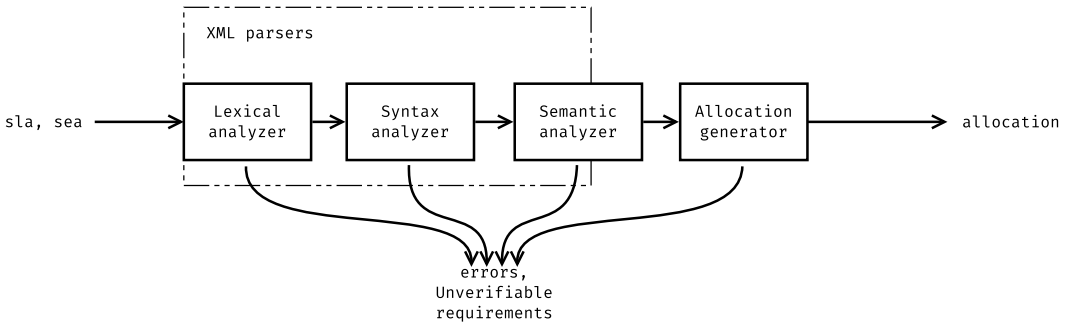


Figure 6.6 – Allocation processing simplified with XML parsers

6.2 Implementation of the allocation use case

The allocation tool uses back-end modules and a front-end adapter in order to generate usable scheduling. This is represented in Figure 6.7.

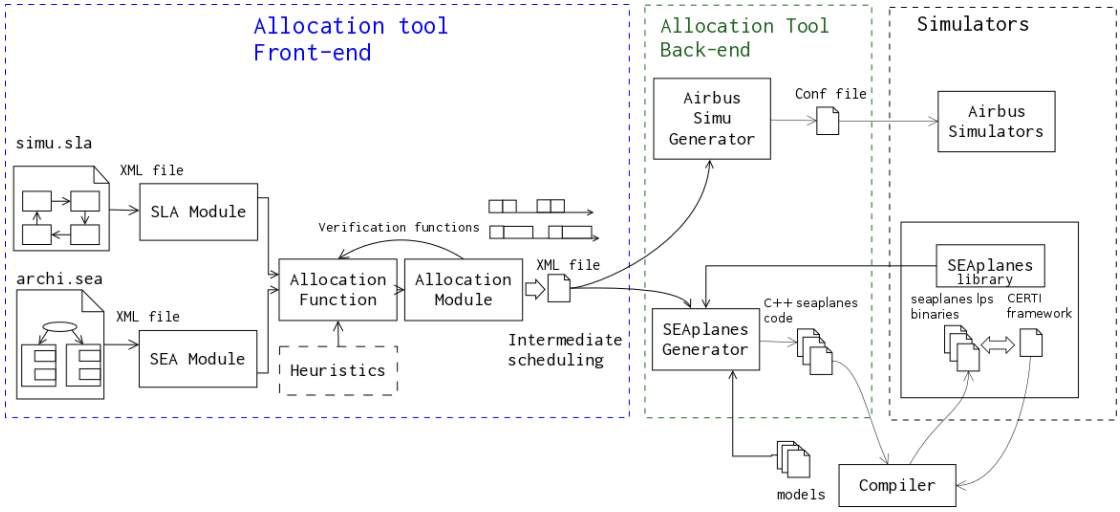


Figure 6.7 – Allocation tool modules interaction in allocation use case

The modules implementations will be explained in the following.

6.2.1 Front-end modules

In order to implement the allocation tool, two specific front-ends must be designed, one for each language, the SLA module and the SEA module in Figure 6.7.

Since we opted for the use of formatting languages to implement SLA and SEA, the main part of the work consists in setting up equivalences between the formalisms of chapter 5 and XML, and implementing one module per language, transforming XML into an object that can be manipulated by the allocator.

6.2.1.1 SLA module

The basic structure of the SLA is directly translated, the components are grouped together, as well as the channels and requirements. The Listing 6.2 presents the writing of a SLA in XML form. In the XML form, three XML element `components`, `channels` and `requirements` are contaning respectively `component`, `channel` and `requirement` elements. Those elements will be defined in the following.

Listing 6.2 – Basic structure of an SLA file

```
<?xml version="1.0" ?>
<sla name="sla_example" xmlns="">
    <components>
        <!-- ... -->
    </components>
    <channels>
        <!-- ... -->
    </channels>
    <requirements>
        <!-- ... -->
    </requirements>
</sla>
```

6.2.1.1.1 SLA components One of the problems with XML is its verbosity. As part of the PoC, we decided to technically limit our choices in order to manipulate an implementation as simple as possible, so we adapted our components.

First of all, for the sake of simplicity, we can consider applications with only an advancement and output function, and known initial states. This is not true in the general case, and an industrial implementation will have to reconsider this choice, but it is sufficient for a simple PoC.

Moreover, it is easier to consider that this function can simply be found from the name of the model to which it belongs, so we choose to remove from our implementation of the SLA components the elements that belong to the model, and to refer them by the model name alone.

The remaining explicit elements are those necessary for the allocator, namely the model ports, its period, coming from its progress and output function, and its time budget, this component implementation is in the form of Equation 6.1. XML form implement the component as an element `component`, with `period` and `time_budget` attributes. `component` ports are two subelements `ports_in` and `ports_out`, containing subelement `port` with their `label` attribute.

Writing example in XML is given in Listing 6.3.

$$c_{impl} = < component\ name, Ports_{in}, Ports_{out}, period, time\ budget > \quad (6.1)$$

Listing 6.3 – Example of a SLA components

```
<components>
    <component name="A" period="50ms" time_budget="1ms">
        <ports_in> <port label="x"/> </ports_in>
        <ports_out> <port label="y"/> </ports_out>
    </component>
    <component name="B" period="50ms" time_budget="1ms">
        <ports_in> <port label="y"/> </ports_in>
        <ports_out> <port label="z"/> </ports_out>
    </component>
    <component name="C" period="50ms" time_budget="1ms">
        <ports_in> <port label="z"/> </ports_in>
        <ports_out> <port label="x"/> </ports_out>
    </component>
</components>
```

6.2.1.1.2 SLA channels Another limitation of XML is that the data cannot be written as a graph but as a tree. Channels are arcs between components. It is possible to make each channel a branch of the tree, *i.e.* a subelement `channel`, but it is necessary to add information to the channels.

Pragmatically, we have decided to add the names of the input and output components, as in Equation 6.2.

$$\lambda_{impl} = << component_{in}, port_{in} >, < component_{out}, port_{out} >> \tag{6.2}$$

In XML form, for each name and port couple, we use a sub-element, respectively `from` and `to`, containing the `component` name and `port` attributes. This form is illustrated by examples with tree channels between *A*, *B* and *C* components in Listing 6.4.

Listing 6.4 – Example of an SLA channels

```
<channels>
    <channel>
        <from component="A" port="y"/>
        <to component="B" port="y"/>
    </channel>
    <channel>
        <from component="B" port="z"/>
        <to component="C" port="z"/>
    </channel>
    <channel>
        <from component="C" port="x"/>
        <to component="A" port="x"/>
    </channel>
</channels>
```

6.2.1.1.3 SLA requirements The requirements are all different, and we have such different representations, but they do not necessarily have the same importance. It is interesting to

be able to differentiate between an important requirement that covers a functionality that is

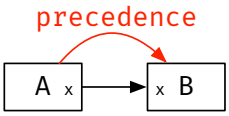


Figure 6.8 – Example of a concrete implementable precedence requirement

Listing 6.5 – A precedence in an SLA

```
1        <requirement weight="100">
2            <precedence from="A" to="B"/>
3        </requirement>
```

The latency requirement implementation The latency constraint is expressed on a data path. It consists in defining on a given path a maximum time limit, and possibly a minimum time limit. In SLA form, it is about defining for this concrete requirement a `latency` element, with as attributes a `delay_max`, and possibly a `delay_min`, as well as a set of sub-elements for paths. We have decided to represent this data path by a `path` sub-element containing all the channels that form the path. These channels contain component and input/output port information, and during a complete industrial implementation, the output ports of a component may have different frequencies.

However in XML, there is no notion of order between the sub-elements of an element, so it is necessary to add this information. For this purpose, the `channel` sub-elements will be extended into an `ord` sub-element, which contains the information of the order on the path given by an `index` attribute.

An example of a latency constraint between three components A, B, and C, shown in Figure 6.9, is given in SLA form in Listing 6.6.

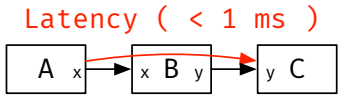


Figure 6.9 – Example of a concrete implementable latency requirement

Listing 6.6 – A latency constraint in an SLA

```
1       <requirement weight="100">
2           <latency delay_max="1.0ms">
3               <path>
4                   <ord index="0">
5                       <channel>
6                           <from component="A" port="x"/>
7                           <to component="B" port="x"/>
8                       </channel>
9                   </ord>
10                  <ord index="1">
11                      <channel>
12                          <from component="B" port="y"/>
13                          <to component="C" port="y"/>
14                      </channel>
```

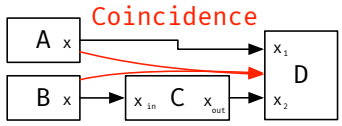


Figure 6.10 – Example of a concrete implementable coincidence requirement

Listing 6.7 – A coincidence constraint in an SLA

```
1        <requirement weight="100">
2            <coincidence>
3                <path>
4                    <ord index="0">
5                        <channel>
6                            <from component="A" port="x"/>
7                            <to component="D" port="x_1"/>
```

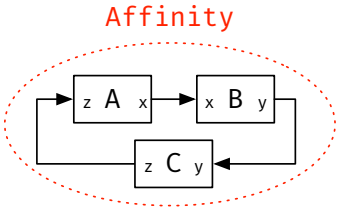


Figure 6.11 – Example of a concrete implementable precedence requirement

Listing 6.8 – An affinity constraint in an SLA

```
1       <requirement weight="100">
2           <affinity>
3               <component name="A"/>
4               <component name="B"/>
5               <component name="C"/>
6           </affinity>
7       </requirement>
```

6.2.1.2 SEA module

The SEA is easier to write than the SLA. It is not a description of a component network, but an architecture, which we already know has a two-level architecture.

The main component of this architecture are the LPs. As such, `logical_processors` element is characterized by `multiperiodic`, `real-time`, and possibly `number` attributes if there can only be a limited number of them.

We then express the communication between the LPs and within the LPs with the sub-elements, respectively, `intraprocessor_communication` and `interprocessor_communication`, and an attribute type characterizing it, such as `sharedmem` for shared memory, or `p2p`, with the possibility of expressing synchronization with a `sync` attribute.

The SEA is deliberately more expressive in order to easily add new simulation frameworks, while allowing the concrete expression of synchronous or non-synchronous behavior, periodic or not, in order to give sufficient indication to the allocation tool to make its calculations of scheduling and latency times.

The main work of adding new behaviours is at the level of the allocation module in which it is necessary to clearly express how to do the calculations.

Listing 6.9 illustrates the writing of components in the SEAplanes SEA.

SEAplanes can have as many LP as needed, so there is no set `number`, it will be ignored. SEAplanes can schedule models in a multiperiodic way, so the `multiperiodic` attribute is set to `True`. SEAplanes can support RT or not, the `real-time` attribute can be set to `True` or `False` depending on the scheduling needs.

For SEAplanes communications, the intraprocessor communication is of the shared memory type. The communication interprocessor is in P2P in a synchronous way.

Listing 6.9 – The SEAplanes SEA

```
1 <?xml version="1.0" ?>
2 <sea name="seaplanes" xmlns="">
3     <logical_processors multiperiodic="True" real-time="False">
4         <intraprocessor_communication type="sharedmem"/>
5         <interprocessor_communication type="p2p" sync="synchronous"/>
6     </logical_processors>
7 </sea>
```

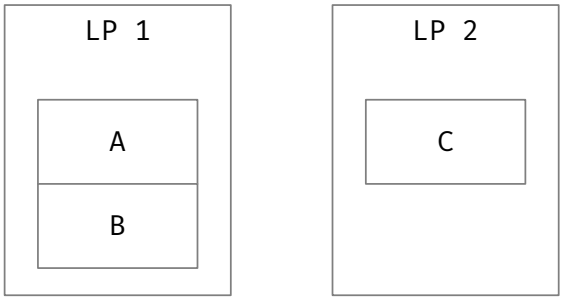


Figure 6.12 – Example of an allocation

Listing 6.10 – Example of an intermediate scheduling allocation

```
1 <?xml version="1.0" ?>
2 <allocation name="alloc" xmlns="">
3     <logical_processor>
4         <task name="A" ord="0" period="50ms"/>
5         <task name="B" ord="1" period="50ms"/>
6     </logical_processor>
7     <logical_processor>
8         <task name="C" ord="0" period="50ms"/>
9     </logical_processor>
10 </allocation>
```

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 [DL78]. 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. Heuristics will be described in section 6.3.

6.2.3 Back-end adapters

Adapters should produce scheduling compatible with targeted simulators. Their implementations depend mostly on the targeted simulators.

6.2.3.1 sEAplanes adapter

Presently, the generation of LP via the sEAplanes adapter is done in a semi-automatic way.

Models associated with LPs can be manually developed, automatically generated, from instance from Matlab, or retargeted from real target or older simulations. The block diagram in Figure 6.13 illustrates the association of simulation models in tasks on logical processors. Figure 6.14 illustrates an allocation with associated flows of an sLA on sEAplanes.

The integration of these models is then done by creating shared memory for internal exchanges, HLA publications and subscriptions for exchanges between LPs, and extending the initialization, finalization, and simulation loop instance methods, with respectively the functions of initialization, finalization, and model advancement.

1. The advancement functions are those of advancement and output described by Equation 5.2 and Equation 5.3.

2. The initialization functions allow to allocate memory for the management of model states, while initializing this memory to the initial states from Equation 5.1.

3. The finalization functions simply destroy the memory space reserved for initialization.

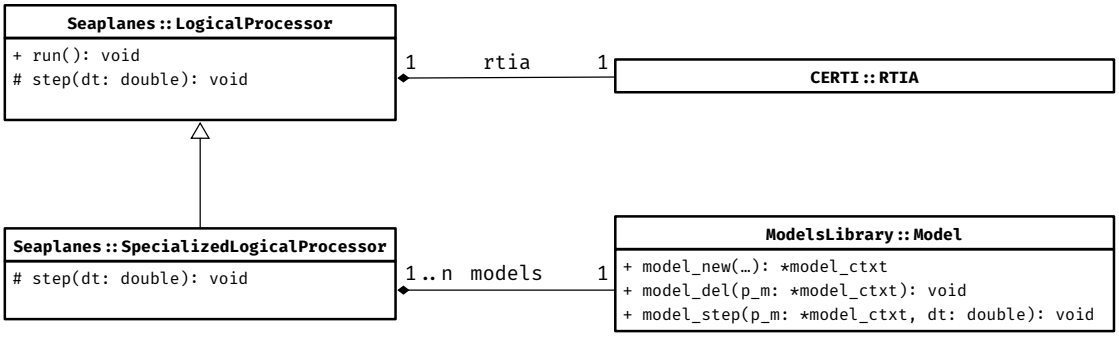


Figure 6.13 – sEAplanes integration of models

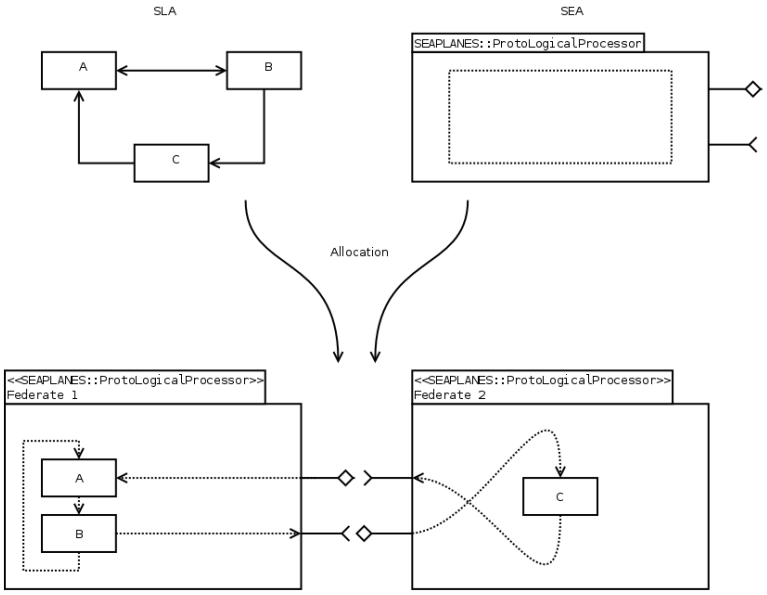


Figure 6.14 – SEAplanes allocation implementation

6.2.3.2 DSS and ASPIC adapters

The adapters used at Airbus do not generate code but configurations for DSS and ASPIC simulation frameworks.

The execution of the schedulings being ensured by DSS and ASPIC, there is no complexity in the adapters which only translates the intermediate schedules. The internal operations of these adapters are Airbus-specific.

6.3 Allocation heuristics

As presented in 6.2.2, the allocation problem is NP-complete, and similar to the Multiple Knapsack Problem (MKP), with some specificities specific to the scheduling of CPS such as:

1. The choice of a limited or unlimited number of LPs.

2. The choice of periods for certain components.

3. Allocation requirements.

There is no direct solution to NP-Complete problems, but it is possible to check in polynomial time, so the classic approach is to use heuristics. Heuristics are calculation methods. They quickly return a result, not necessarily optimal.

6.3.1 Greedy heuristics

The greedy heuristics are variations of the most known heuristics used to solve the MKP. 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. The algorithm to implement these heuristics is given in algorithm 1. It is possible to use the same algorithm for all greedy heuristics by adapting the type of all LPs.

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.

Table 6.1 summarizes the possibility of using type for a list to obtain a behavior of $*$-fit.

Table 6.1 – List type implementation for $*$-fit heurisitics

$*$-fit algorithm	α type
First-fit	List
Next-fit	Circular buffer
Best-fit	Binary heap, indexed by occupation
Worst-fit	Binary heap, indexed by the inverse of occupations

Algorithm 1: $*$-fit heuristics for SLA allocation on SEA logical processors

```
*-fit_heuristic (sla, sea, α lps)
    inputs : An SLA sla, an SEA sea, an empty logical processors container lps, of type
             α
    output : An allocation of components on logical processors
    if sea.nb_lp then
        /* Populating lps                                                    */
        lps ← sea.nb_lp × lp_∅
    foreach component ∈ sla.components do
        allocated ← False
        foreach lp ∈ lps do
            if allocable(component, lps, lp) then
                lp ← lp · component
                allocated ← True
                break
        if ¬allocated ∧ lps.extensible then
            if allocable(component, lps, lp∅ · component) then
                lps ← lps + lp∅ · component
                allocated ← True
        if ¬allocated then
            /* Simplified, removing least important requirement.             */
            sla.rs ← sla.rs − r
    return lps
```

6.3.2 Simulated Annealing

One of the limitations of the previous heuristics is the strong dependencies between the order of inputs and the quality of the output. Indeed, greedy heuristics do not know how to move an already allocated component, even when it is obvious that a movement can improve the output, so we have been working on implementing other heuristics, which do not have this problem. Our choice was to consider the Simulated Annealing (SA).

SA is one of the most common metaheuristics, one of the specificities of which is to do a search in a neighborhood. SA is Inspired by metallurgy: heating and cooling (annealing) cycles make the metal stronger. It allows to search for a global maximum or minimum, while avoiding being trapped in a local one. An energy is defined, which is reduced as the simulated annealing is performed.

The algorithm used, algorithm 2, requires fewer adaptations than greedy heuristics, partly because it is designed taking into account a number of parameters to be chosen. The choices we can make for these parameters are presented in Table 6.2.

Algorithm 2: Simulated annealing

simulated_annealing $(s_0, nrj, v, p, k_{max}, e_{max})$

inputs : A starting state s_0, nrj an "energy" calculation function, v a function that finds a neighbor, p a function that calculates a probability as a function of the energy variation and a temperature, k_{max}, a maximum number of iterations, and e_{max} the maximum allocation energy.

output : A final state s_f

$s_{tmp} \leftarrow s_0$
$s_f \leftarrow s_0$
$e_{tmp} \leftarrow nrj(s_{tmp})$
$e_f \leftarrow nrj(s_0)$
$k \leftarrow 0$
while $k < k_{max} \wedge e_f \neq e_{max}$ **do**
  $s_v = v(s)$
  $e_v = nrj(s_v)$
  if $e_v > e_{tmp} \vee rand1() < p(e_n - e_{tmp}, temp(k/k_{max}))$ **then**
    $s_{tmp} \leftarrow s_v$
    $e_{tmp} \leftarrow e_v$
  if $e_v > e_f$ **then**
    $s_f \leftarrow s_v$
    $e_f \leftarrow e_v$
  $k \leftarrow k + 1$
return s_f

Table 6.2 – parameters values

Parameter	Value
p	Arbitrary. Allows to get out of local maximums, but may slow down the execution.
k_{max}	Arbitrary. Depends on the number of components, must be large enough to cover a large neighbourhood, but not too large to limit the search time
nrj	Function of the fulfilment of the requirements, currently the sum of the weights.
v	Definition of neighbourhood: • A single change between partitions: distant neighborhood. • A single change between mappings: nearby neighborhood. Neighborhood is illustrated in Figure 6.15, with 3 components A, B and C.

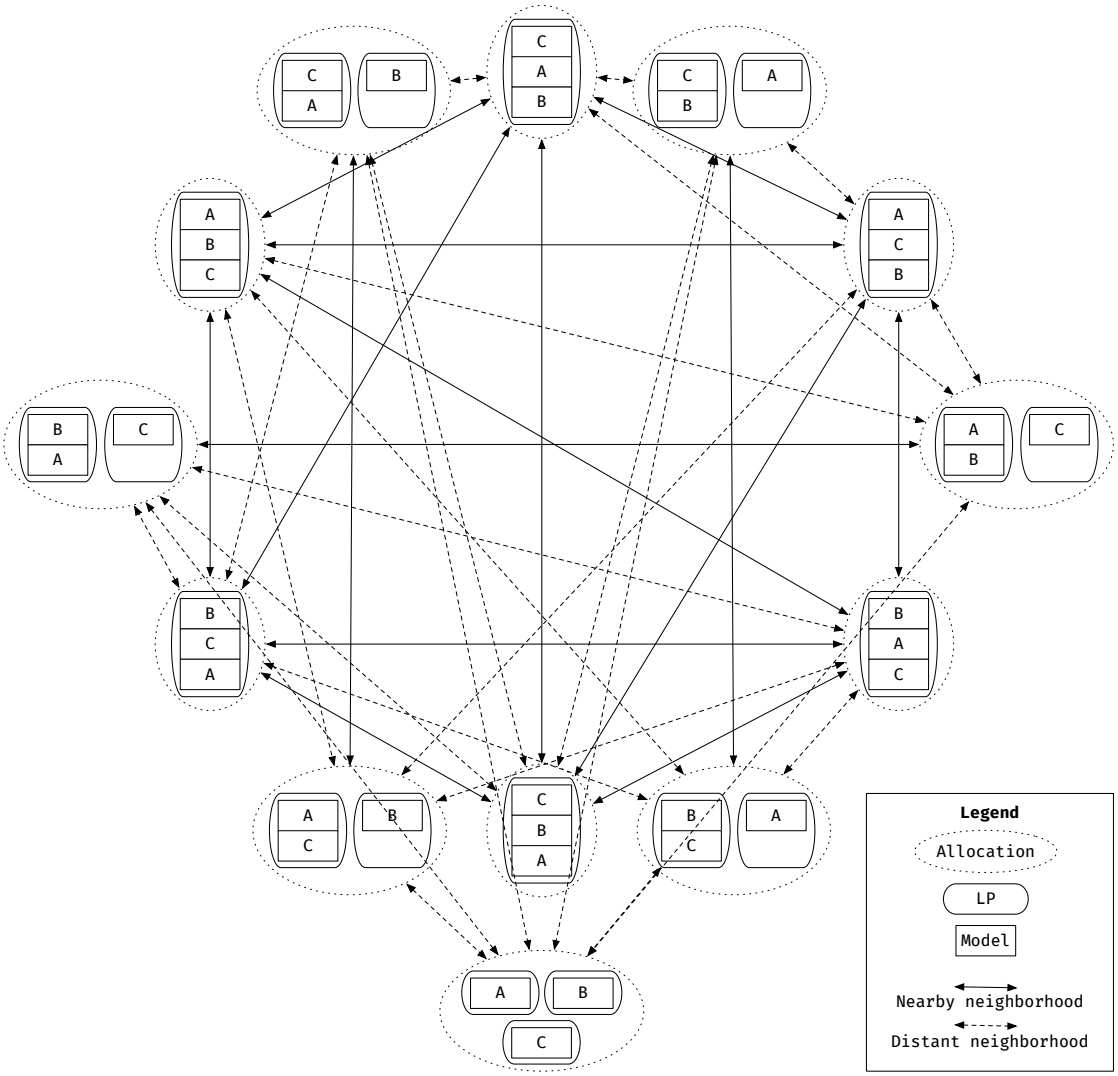


Figure 6.15 – Illustration of the neighborhood connections between different model allocations

Chapter 7

The Redundant ROSACE case study

Contents

7.1 The original case study 122

7.1.1 ROSACE introduction 122

7.1.2 Original operational scenarios 122

7.2 The RROSACE case study 123

7.2.1 Modifying ROSACE to RROSACE 123

7.2.2 From controllers to redundant Flight Control Computers 123

7.3 Implementation of the case study 127

7.3.1 RROSACE models library 127

7.3.2 Simple loop 130

7.3.3 Testing strategy 131

7.3.4 Results 139

7.4 Method and tool validation 142

7.4.1 Academic validation of concepts 142

7.4.2 Feedback of industrial experience 145

In order to validate our method during its development, and to perform tests during the implementation of the tool, we decided to implement a simple and precise case study.

We decided to capitalize on previous experiences in software engineering by building our case study on an open-source model identified in the scientific literature, and implemented our solution in a composable and reusable way to be able to perform tests with different frameworks, both academic and industrial.

7.1 The original case study

7.1.1 ROSACE introduction

Research Open-Source Avionics and Control Engineering (ROSACE) is a case study covering 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 [PSG+14]. 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-core.

A major challenge in designing the ROSACE controller is the need of interactions between control and software engineers. Control engineering and computer science does 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 limitation, such as tasks schedulability and network resources. This issue is particularly prevalent when designing CPS [HE11], endorsing our willingness to base our study upon 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

7.1.2 Original operational scenarios

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

case 1 The pilot set a new value to the inertial vertical speed (v_z).
v_z: $0meter(m)s^{-1} \rightarrow 2.5ms^{-1}$

case 2 The pilot set a new value to the true air speed (v_a).
v_a: $230ms^{-1} \rightarrow 235ms^{-1}$

case 3 The pilot wait for time (t) = 50second (s) then set a new inertial altitude (h).
h: $10000m \rightarrow 11000m$, with $v_z = -2.5ms^{-1}$

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

All of them are while in cruise phase, at equlibrium.
Cases 1 and 2 also have the following requirements:

case 1 **P1** Settling time $v_z \leq 10s$

P2 Overshoot $v_z \leq 10\%$

P3 Rise time $v_z \leq 6s$

P4 Steady-state error $v_z \leq 5\%$

case 2 **P1** Settling time $v_a \leq 20s$

P2 Overshoot $v_a \leq 10\%$

P3 Rise time $v_a \leq 12s$

P4 Steady-state error $v_a \leq 5\%$

7.2 The RROSACE case study

7.2.1 Modifying ROSACE to RROSACE

The original ROSACE case study was extended by adding redundant controllers, allowing us to play with scheduling errors; and we have also implemented a hybrid simulation.

We created tree versions of Redundant ROSACE (RROSACE):

- A discrete RROSACE, from the original case study available in the ROSACE repository [Des16a].
- A continuous RROSACE, from the continuous model provided by the ROSACE team.
- An hybrid RROSACE, from the two precedent versions.

As we wanted to have a simulation with the most relevant behavior, we chose a component breakdown that seems to be the closest to a real :

- Avionic systems, which are discrete, are represented by discrete models.
- Physical components, such as the engine, are represented by continuous models.

The choice of discrete or continuous model in the hybrid RROSACE depends on these constraints, and are illustrated in **??**.

The hybrid versions running on Matlab/Simulink will be used as reference. The discrete RROSACE might be used in our future works, as it is close to the future implementations. The continuous RROSACE has been created to generate the hybrid RROSACE and will not be reused in further works.

7.2.2 From controllers to redundant Flight Control Computers

7.2.2.1 Controllers packaging in FCCs

The FCC correspond to the ROSACE controllers. Three controllers exist in the original case study:

- inertial vertical speed controller.
- true air speed controller.
- inertial altitude controller.

We group them in a single component, called FCC, and add a monitoring logic, in order to use this FCC in command or monitor mode, depending on its usage, as depicted in Figure 7.1.

- An FCC that only takes inputs from filters and flight mode is a COM and produce the commands, see Figure 7.2.
- An FCC taking the same inputs, and the output of another FCC is a MON and produce relays, *i.e.* boolean values on either the monitored commands are correct or not, see Figure 7.3.

The monitors will also share with each other information in order to determine the master in law.

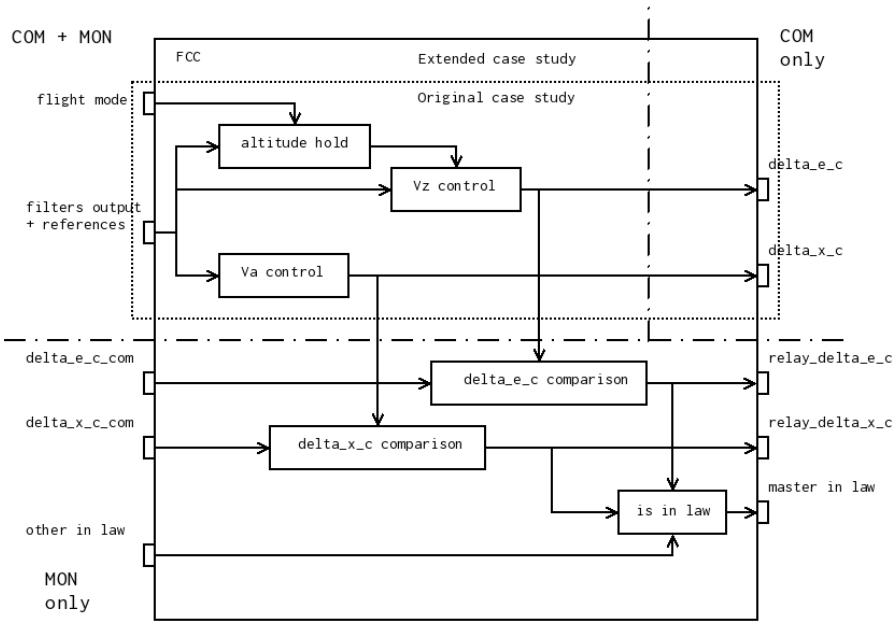


Figure 7.1 – Design of a new FCC, from controllers of original case study

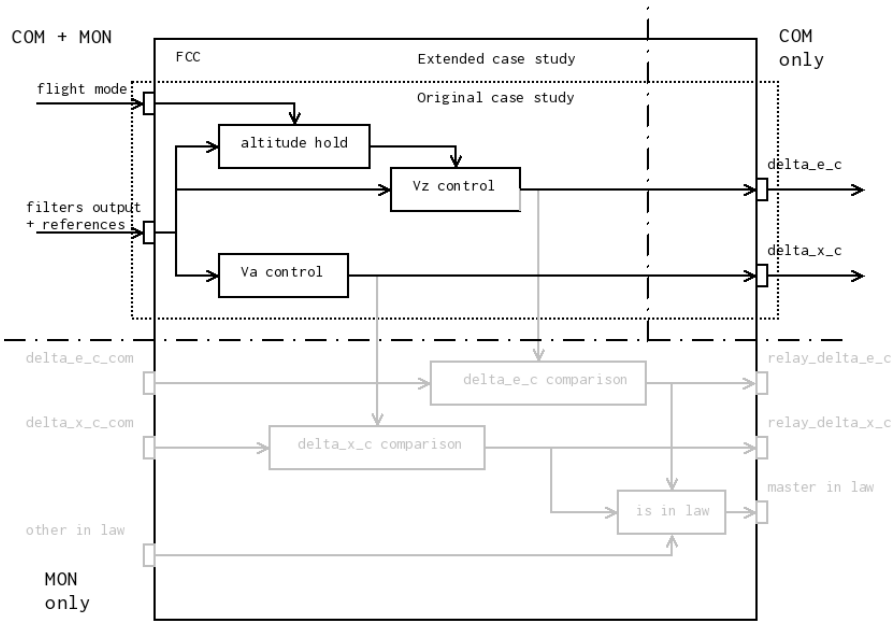


Figure 7.2 – Instantiation of an FCC COM

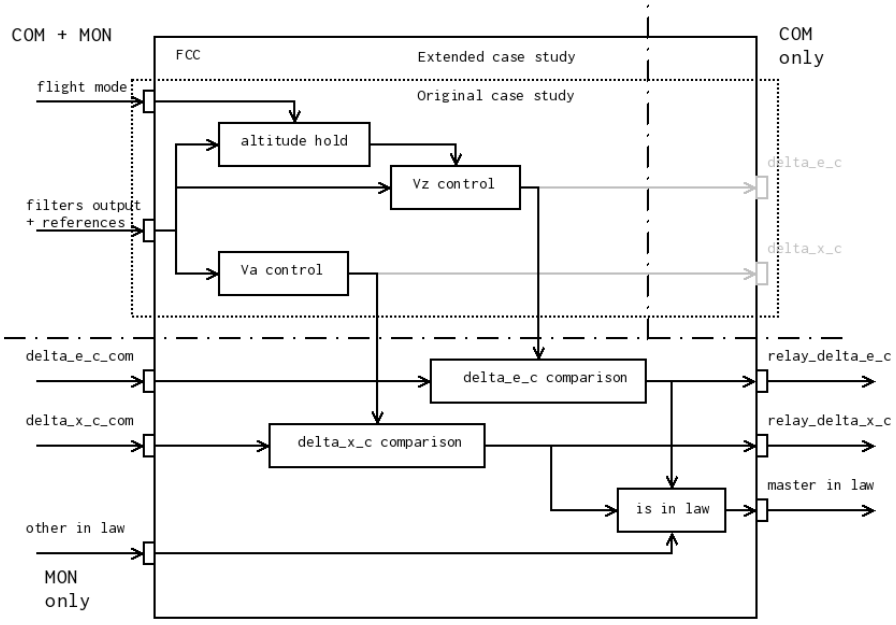


Figure 7.3 – Instantiation of an FCC MON

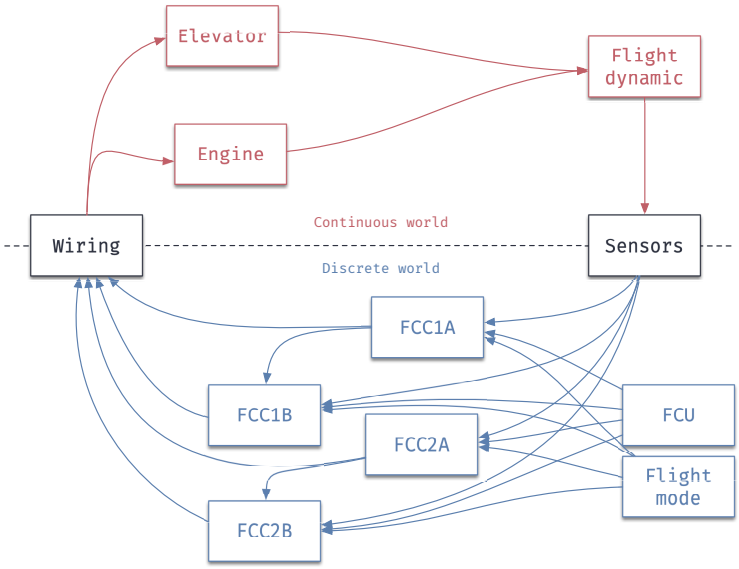


Figure 7.4 – The RROSACE case study components view

7.2.2.3 Injecting errors

In RROSACE, there are no redundant filters, and we only handle a unique error. To test the redundancy, we have to implement error injection in the models. This error injection mechanism is illustrated in **??**.

7.3 Implementation of the case study

7.3.1 RROSACE models library

7.3.1.1 Simulation blocks

The first step in the implementation of the case study was the analysis of the pre-existing case study, in order to clearly identify the functionality coverage by the simulation components, in order to perfect the composability of our solution.

7.3.1.2 Why and how a core models library

From the functionnal analysis, we identified components and their simulation blocks. Nevertheless, in order to use DSS and SEAplanes for our simulation, we need a common source base. HLA-CERTI could eventually use the previous Matlab blocks

- The library must be easily readable.
- The library must be easily extensible.
- The library must be portable.
- The library must be deterministic.
- The library must have the lowest impact on simulation execution time.

In order to achieve so, we use ANSI C language with only standard libraries [Rit78].

Every components in our analysis is a models library class instance. The models library classes is similar to MATLAB/SIMULINK blocks, generic. For instance, while there are 5 different blocks for sensors in ROSACE, the five blocks are of the same kind, but their parameters are different.

7.3.1.3 Library functionality

Model libraries contain models contexts and interfaces. We followed the Object-Oriented Programming (OOP) paradigm in our to keep a simple and understandable design.

- A model is a context with C-functions used to manipulate it. The context and functions follow the same concepts for every models in the library.
- A model is totally opaque to the library user.
- The context is an opaque structure. The user instantiate it thanks to a pointer on this structure.
- The library provides functions to allocate and free context.
- The library provides a function to make an iteration using the model.

- Eventually, when the model use too much inputs and outputs, the library provides structures to pass them to the function easily.

Nevertheless, we have to set limitations in order to obtain a good-enough library within a reasonable time. Currently, our models can only be discrete. Discrete models still discrete, but continuous models are discretized. Moreover, we get discretization parameters from the original ROSACE case study. Our library cannot instantiate any discretized model at any frequency. Only pre-calculated ones are available for now, these is a design choice that can evolve in future work.

7.3.1.4 Concrete implementation

The final step is the concrete implementation of the software library.

In order to present this implementation, we present two listings, corresponding to the implementation of the elevator :

- Listing 7.1 — the implementation of the elevator interface.
- Listing 7.2 — the implementation of the elevator body.

The other models were implemented in the same way.

Listing 7.1 – elevator.h

```
#ifndef ELEVATOR_H
#define ELEVATOR_H

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

/* The model structure */
struct elevator;
typedef struct elevator elevator_t;

/* The model init function */
elevator_t *elevator_new(double omega, double xi);

/* The model destruction function */
void elevator_del(elevator_t *p_elevator);

/* The model step function */
int elevator_step(elevator_t *p_elevator, double delta_e_c, double dt,
                  double *p_delta_e);

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* ELEVATOR_H */
```

Listing 7.2 – elevator.c

```
1 #include <constants.h>
2 #include <elevator.h>
3 #include <stdlib.h>
4
5 /* elevator model structur */
6 struct elevator {
7   /* elevator model parameters */
8   double omega;
9   double xi;
10  /* elevator model states */
11  double x[2];
12 };
13
14 elevator_t *elevator_new(double omega, double xi) {
15
16  /* allocating memory */
17  elevator_t *elevator = (elevator_t *)calloc(1, sizeof(elevator_t));
18
19  if (!elevator) {
20    goto out;
21  }
22
23  /* setting parameters */
24  elevator->omega = omega;
25  elevator->xi = xi;
26
27  /* initializing states */
28  elevator->x[0] = DELTA_E_EQ; /* initial state at equilibrium (
        precalculated in
29                                constants) */
30  elevator->x[1] = 0.0;
31
32 out:
33  return (elevator);
34 }
35
36 void elevator_del(elevator_t *p_elevator) {
37  if (p_elevator) {
38    free(p_elevator);
39  }
40 }
41
42 int elevator_step(elevator_t *p_elevator, double delta_e_c, double dt,
43                 double *p_delta_e) {
44  int ret = EXIT_FAILURE;
45  double x_dot[2];
46
47  if (!p_elevator) {
48    goto out;
49  }
50
51  if (!delta_e) {
```

```
52    goto out;
53  }
54
55  /* Setting output */
56  *p_delta_e = p_elevator->x[0];
57
58  /* Calculating new states */
59  x_dot[0] = p_elevator->x[1];
60  x_dot[1] = -p_elevator->omega * p_elevator->omega * p_elevator->x[0] -

61             2.0 * p_elevator->xi * p_elevator->omega * p_elevator->x[1]
                   +
62             p_elevator->omega * p_elevator->omega * p_delta_e_c;
63
64  /* Setting new states */
65  p_elevator->x[0] += dt * x_dot[0];
66  p_elevator->x[1] += dt * x_dot[1];
67
68  ret = EXIT_SUCCESS;
69
70 out:
71  return (ret);
72 }
```

7.3.2 Simple loop

In order to verify that the implementation of the composable simulation models, we have implemented a first execution, without distribution.

This is a sequential execution of the models according to their periods, starting with the elevator and engine, then following the order of production consumption, as illustrated in the sequence diagram in the Figure 7.5.

This simulation loop has been implemented in plain C.

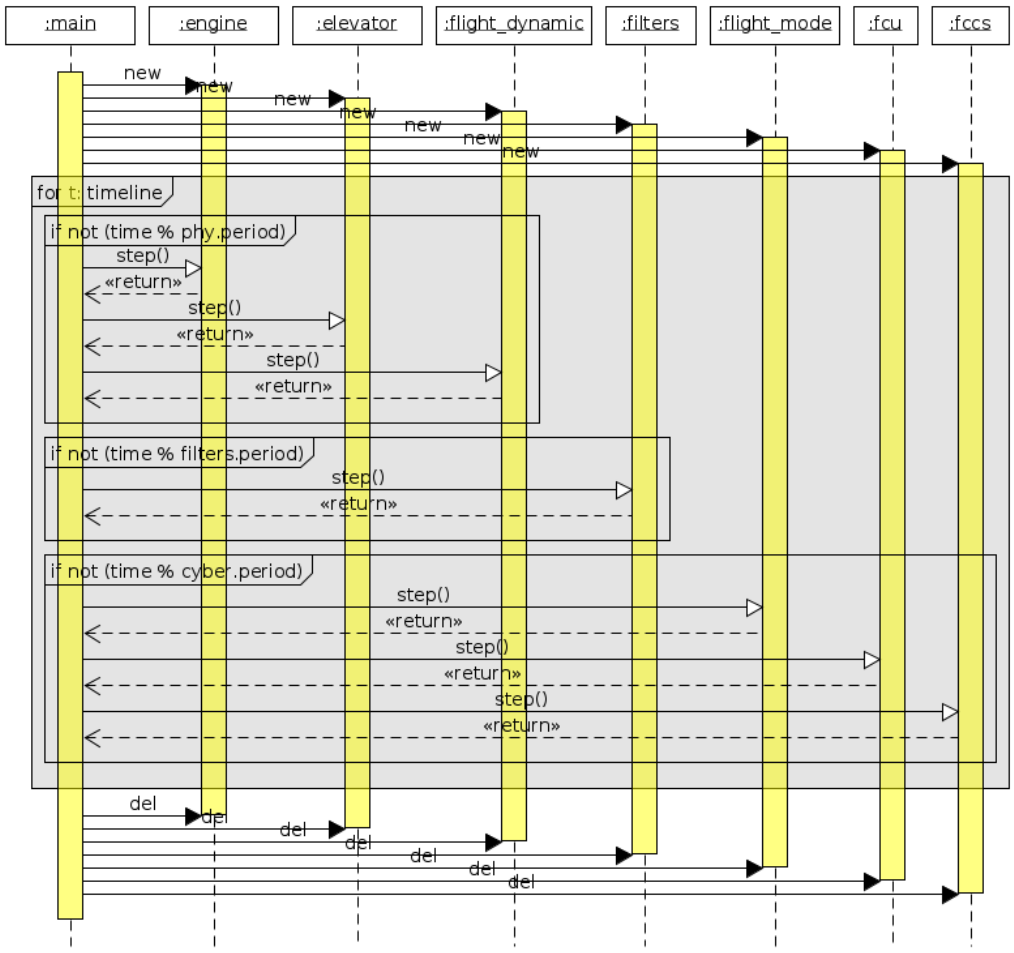


Figure 7.5 – RROSACE simple loop sequence diagram.

7.3.3 Testing strategy

Once the verification of the implementation of the models was possible, we set up a testing strategy, with several objectives:

- Check that the implementation of the library is correct.
- Check that the use hybrid simulations has a negligible impact.
- Check that the addition of redundancy has no impact in a case of normal operation, or with only one error.

7.3.3.1 Operational scenarios

We worked on 4 operational scenarios, taken from the original ROSACE case study.

Our scenarios have the same starting point: The plane is in cruise phase

7.3.3.2 Case 1: v_z control

The plane is in commanded mode. At $t = 0s$, the pilot set v_z to 2,5 meters per second ($m.s^{-1}$) The plane must adjust its parameters to reach the given v_z, while maintaining its v_a.

7.3.3.3 Case 2: v_a control

The plane is in commanded mode. At $t = 0s$, the pilot set v_a to 235 $m.s^{-1}$. The plane must adjust its parameters to reach the given v_z, while maintaining its v_a.

7.3.3.4 Case 3: h control: climb

The plane is in altitude hold mode. At $t = 50s$, the pilot set h to 11000 m The plane must adjust its v_z to reach the given h, while controlling v_a.

7.3.3.5 Case 4: h control: step climbs

The plane is in altitude hold mode.

- At $t = 50s$, the pilot set h to 10500 m.
- At $t = 400s$, the pilot set h to 11000 m.
- At $t = 750s$, the pilot set h to 11500 m.
- At $t = 1100s$, the pilot set h to 8000 m.

The plane must adjust its v_z to reach the given h, while controlling v_a.

7.3.3.6 Test cases

In order to prove that the new case study can be used as reference , we tested that requirements from the original case study are still met with the extended version.

In the following, we will present the results for the operational scenario Case 1 “The pilot set a new value to the initial vertical speed”. If nothing is stated, the nature of the case study is “discrete”.

The tests are the following:

Redundant controllers:
This test aims to check that the adding of redundant FCC in the discrete models does not impact the results.

Discrete to cyber-physical model:
This test allows verifying that the manipulation of discrete and continuous models is still valid in regard with the requirements in subsection 7.1.2.

The goal of this test is to verify that the CPS simulation, called "hybrid", still meet the original ROSACE requirements.

We compare the 3 main results, true air speed, inertial vertical speed and inertial altitude, from the discrete RROSACE case study, with the ones from the hybrid RROSACE case study.

Error injection:
The test consists in two injections:

- From $t = 10s$ to $t = 20s$:
  add 0.001 rad to elevator deflection command (δ_{e_c}) from FCC1;
- From $t = 30s$ to $t = 40s$:
  add 0.001 rad to δ_{e_c} from FCC2.

When the error is detected on FCC1, FCC2 become the master, and when the error is detected in FCC2, as FCC1 is no longer in error, FCC1 become the master.

7.3.3.7 RROSACE simulation with SEAplanes

SEAplanes is based on HLA, so we generated a FOM containing the attributes and object of the federation Listing 7.3. To create a federation, all the models present in this FOM must be simulated. The declaration of HLA object instances depends on the allocation breakdown.

Listing 7.3 – rrosace.fed

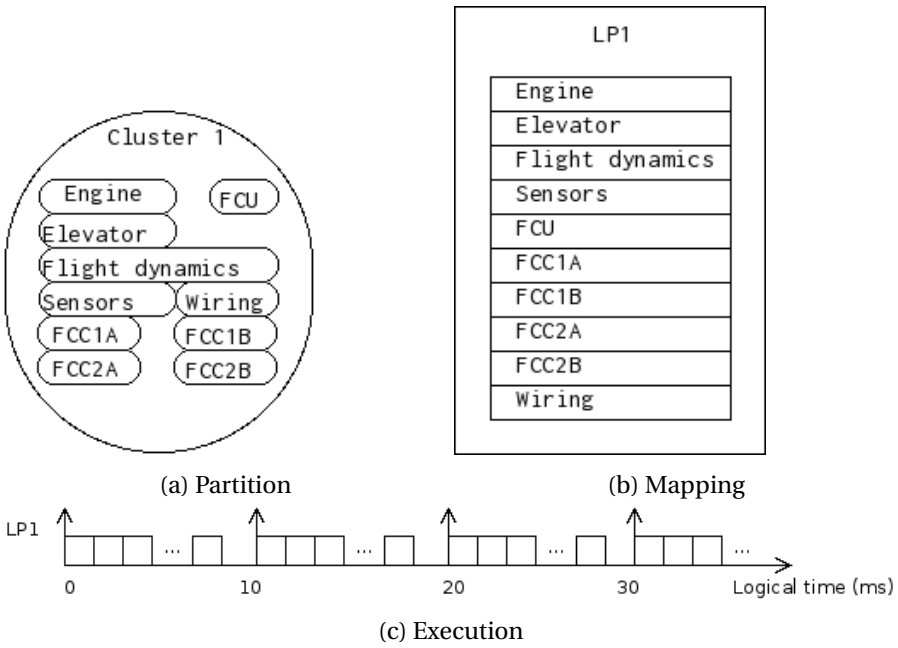
```
(FED
    (Federation RROSACE) ;; RROSACE tag
    (FEDVersion v1.5) ;; FED Version
    (spaces ;; No routing spaces for now ;; TBD
    )
    (objects

        (class ObjectRoot
            (attribute privilegeToDeleteObject reliable timestamp)
            (class RTIprivate)
            (class Aircraft ;; 200Hz
                (attribute key reliable timestamp)
                (attribute altitude reliable timestamp)
                (attribute verticalAcceleration reliable timestamp)
                (attribute verticalSpeed reliable timestamp)
                (attribute trueAirspeed reliable timestamp)
                (attribute pitchRate reliable timestamp)
            )
            (class Filters
                (attribute key reliable timestamp)
                (attribute filteredAltitude reliable timestamp) ;; 50Hz
```

```
22              (attribute filteredVerticalAcceleration reliable
                    timestamp) ;; 100Hz
23              (attribute filteredVerticalSpeed reliable timestamp) ;;
                    100Hz
24              (attribute filteredTrueAirspeed reliable timestamp) ;;
                    100Hz
25              (attribute filteredPitchRate reliable timestamp) ;; 100Hz
26          )
27          (class Reference ;; ?? Hz, might not be periodic.
28              (attribute key reliable timestamp)
29              (attribute altitudeRef reliable timestamp)
30              (attribute trueAirspeedRef reliable timestamp)
31              (attribute verticalSpeedRef reliable timestamp)
32          )
33          (class Elevator ;; 200Hz
34              (attribute key reliable timestamp)
35              (attribute elevatorDeflection reliable timestamp)
36          )
37          (class Engine ;; 200Hz
38              (attribute key reliable timestamp)
39              (attribute engineThrust reliable timestamp)
40          )
41          (class FlightMode ;; ?? Hz, same as the reference
42              (attribute key reliable timestamp)
43              (attribute mode reliable timestamp)
44          )
45          (class ControlCommand ;; 50Hz or 200Hz
46              (attribute key reliable timestamp)
47              (attribute elevatorDeflectionCommand reliable timestamp)
48              (attribute throttleCommand reliable timestamp)
49          )
50          (class ControlCommandPartial ;; 50Hz
51              (attribute key reliable timestamp)
52              (attribute elevatorDeflectionCommandPartial reliable
                    timestamp)
53              (attribute throttleCommandPartial reliable timestamp)
54          )
55          (class RelayControlCommandPartial ;; 50Hz
56              (attribute key reliable timestamp)
57              (attribute relayElevatorDeflectionCommandPartial reliable
                     timestamp)
58              (attribute relayThrottleCommandPartial reliable timestamp
                    )
59          )
60      ) ;; end ObjectRoot
61  ) ;; end Objects
62  (interactions
63      (class InteractionRoot reliable timestamp
64          (class RTIprivate reliable timestamp)
65          (class StopFreeze reliable receive
66              (parameter KillActivity)
67          )
68      ) ;; end InteractionRoot
```

```
69     ) ;; end Interactions
70 ) ;; end FED
```

For example, we can have a federation with only one LP scheduling all the components, as in Figure 7.6a. Such distribution is called centralized.



(a) Partition (b) Mapping

(c) Execution

Figure 7.6 – Centralized simulation of RROSACE

Or we can have a federation with one LP per component, as in **??**. Such distribution is called fully-distributed.

The binding of the components was done semi-automatically.

Simulation skeleton generation is possible, but the implementation of shared memory and HLA class, instance and attributes has not yet been done due to time limiation.

A very large number of federates covering many schedules have been generated.

Listing 7.4 and Listing 7.5 illustrates the creation of an LP, here for an LP scheduling only the elevator.

Listing 7.4 – ElevatorLP.h

```
1 #ifndef ELEVATOR_LP_H
2 #define ELEVATOR_LP_H
3
4 // std libraries
5 #include <iostream>
6 #include <memory>
7 #include <vector>
8
```

```
 9 // RROSACE federation libraries
10 #include <RROSACEFederation.h>
11
12 // RROSACE models libraries
13 #include <constants.h>
14 #include <elevator.h>
15
16 // Seaplanes libraries
17 #include <LogicalProcessorAttribute.h>
18 #include <LogicalProcessorObjectClass.h>
19 #include <LogicalProcessorObjectInstance.h>
20 #include <LogicalProcessorObjectInstancePublished.h>
21 #include <LogicalProcessorObjectInstanceSubscribed.h>
22 #include <ProtoLogicalProcessor.h>
23
24 #ifndef FEDERATE_NAME
25 #define FEDERATE_NAME Elevator
26 #endif
27
28 namespace Seaplanes {
29
30 class ElevatorLP final : public ProtoLogicalProcessor {
31 private:
32   // models to schedule
33   std::unique_ptr<elevator_t, void (*)(elevator_t *)> up_elevator;
34
35   // shared memory
36   double delta_e_c;
37   double delta_e;
38
39   // HLA objects
40   std::shared_ptr<Object> sp_elevator_class{Object::create(RROSACE::
         ELEVATOR)};
41   std::shared_ptr<Object> sp_control_command{
42       Object::create(RROSACE::CONTROL_COMMAND)};
43
44   // HLA attributes
45   std::shared_ptr<Attribute> sp_elevator_deflection_command_in{
46       Attribute::create(RROSACE::ELEVATOR_DEFLECTION_COMMAND)};
47   std::shared_ptr<Attribute> sp_elevator_deflection_out{
48       Attribute::create(RROSACE::ELEVATOR_DEFLECTION)};
49
50   // HLA object instance to publish and subscribe
51   std::unique_ptr<ObjectInstancePublished> up_elevator_out{
52       ObjectInstancePublished::create(RROSACE::ELEVATOR_OUT,
53                                       sp_elevator_class)};
54   std::unique_ptr<ObjectInstanceSubscribed> up_control_command_in{
55       ObjectInstanceSubscribed::create(RROSACE::CONTROL_COMMAND_IN,
56                                        sp_control_command)};
57
58   // inline model scheduling function
59   void scheduling() final;
60
```

```
61 public:
62   explicit ElevatorLP(const Name & /* federate_name */);
63
64   virtual ~ElevatorLP();
65 };
66
67 } // namespace Seaplanes
68
69 #endif // ELEVATOR_LP_H
```

Listing 7.5 – ElevatorLP.cpp

```
 1 #include <ElevatorLP.h>
 2
 3 #ifndef TIMESTEP
 4 #define TIMESTEP RROSACE::Timestep::PHYSICAL
 5 #endif // TIMESTEP
 6
 7 #ifndef LOOKAHEAD
 8 #define LOOKAHEAD TIMESTEP
 9 #endif // LOOKAHEAD
10
11 using namespace Seaplanes;
12
13 using std::move;
14
15 ElevatorLP::ElevatorLP(const Name &federate_name)
16     : ProtoLogicalProcessor(RROSACE::FEDERATION,
17                             federate_name,
18                             RROSACE::FOM,
19                             RROSACE::TIME_LIMIT,
20                             TIMESTEP,
21                             LOOKAHEAD),
22       // models init
23       up_elevator{
24         elevator_new(RROSACE::ELEVATOR_OMEGA, RROSACE::ELEVATOR_XI),
25         elevator_del
26       }
27 {
28   // binding attritutes with instances
29   bindAttribute(up_control_command_in,
         sp_elevator_deflection_command_in);
30   bindAttribute(up_elevator_out, sp_elevator_deflection_out);
31
32   // adding HLA object
33   addObjectClass(sp_control_command);
34   addObjectClass(sp_elevator_class);
35
36   // setting attributes to publish
37   addPublishedObject(move(up_elevator_out));
38
39   // setting attributes to subscribe
40   addSubscribedObject(move(up_control_command_in));
```

```
41 }
42
43 ElevatorLP::~ElevatorLP() {}
44
45 void ElevatorLP::scheduling() {
46
47   try {
48     // receiving data
49     try {
50       delta_e_c = up_elevator_deflection_command_in->getFreshValue<
              double>();
51     } catch (const AttributeNoFreshValue &) {
52       // ...
53     }
54
55     // scheduling models
56     // ...
57     {
58       auto ret = elevator_step(up_elevator.get(), delta_e_c, TIMESTEP, &
              delta_e);
59       if (ret == EXIT_FAILURE) {
60         throw std::exception("elevator␣step␣failed");
61       }
62     }
63     // ...
64
65     // sending data
66     up_elevator_deflection_out->setValue(delta_e);
67   } catch (const std::exception &e) {
68     std::cerr << "Error:␣" << e.what() << std::endl;
69   }
70 }
```

7.3.3.8 RROSACE simulation with DSS

DSS schedule AP2633 models, using configuration files AP2633 models contains one and only one of our model instance, with an exception for filters. Every AP2633 model is built in a dynamic library with a schedulable main. Those library and the configuration files define the simulation

RROSACE simulation needs all the AP2633 models, biding is described in Table 7.1.

AP2633 Models	RROSACE library file	RROSACE library structure	RROSACE model instances
Flight Dynamic	rrosace_flight_dynamic.h	rrosace_flight_dynamic_t	flight_dynamic
FCC	rrosace_fcc.h	rrosace_fcc_t	fcc1a, fcc1b, fcc2a, fcc2b
Elevator	rrosace_elevator.h	rrosace_elevator_t	elevator
Engine	rrosace_engine.h	rrosace_engine_t	engine
FCU	rrosace_fcu.h	rrosace_fcu_t	fcu
Flight Mode	rrosace_flight_mode.h	rrosace_flight_mode_t	flight_mode
Filters	rrosace_filter.h	rrosace_filter_t	vz_filter, az_filter, va_filter, q_filter, h_filter

Table 7.1 – Mapping of RROSACE models instances with AP2633 models for DSS implementation of RROSACE

The results we obtained with the simulations DSS are equivalent to the simulations SEAplanes. We have concluded that our SEAplanes framework is correct for the publication of results without having to be concerned about the copyright of the results of the industrial framework.

7.3.4 Results

7.3.4.1 Redundant controllers

Figure 7.7 shows the results of the comparing of the original ROSACE case study with the discrete RROSACE one. The difference between the original ROSACE case study and the RROSACE one for v_a, v_z and h is always null.

We can consider the RROSACE model as equivalent to the original ROSACE model.

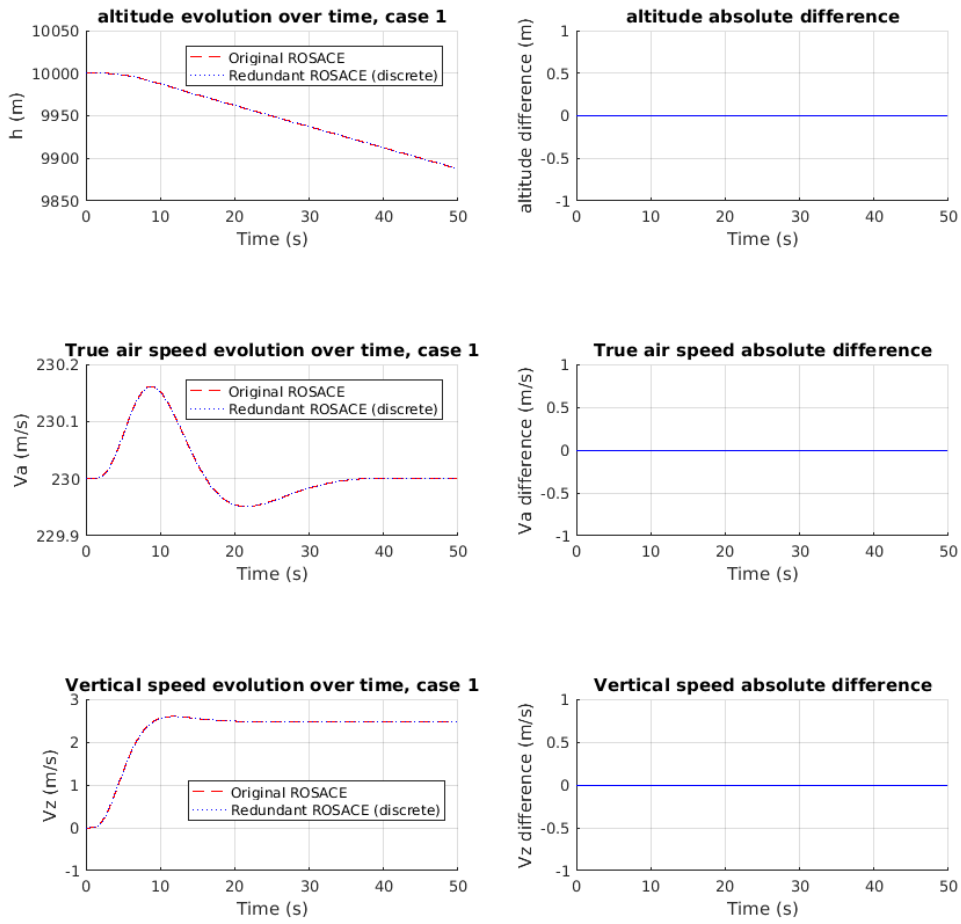


Figure 7.7 – Comparing of discrete RROSACE results with original ROSACE

7.3.4.2 Discrete to cyber-physical models

Figure 7.8 shows the results of the comparing of discrete RROSACE with hybrid RROSACE.

We can notice in the left side of this figure that these two models does not have the same results. However, the differences are negligible, and original ROSACE case study requirements are still meet (subsection 7.1.2). These differences are due to the use of solver by Matlab when simulating the components with continuous models.

Nevertheless, we can consider the hybrid models as sufficiently correct, and we can use it as reference for the RROSACE case study.

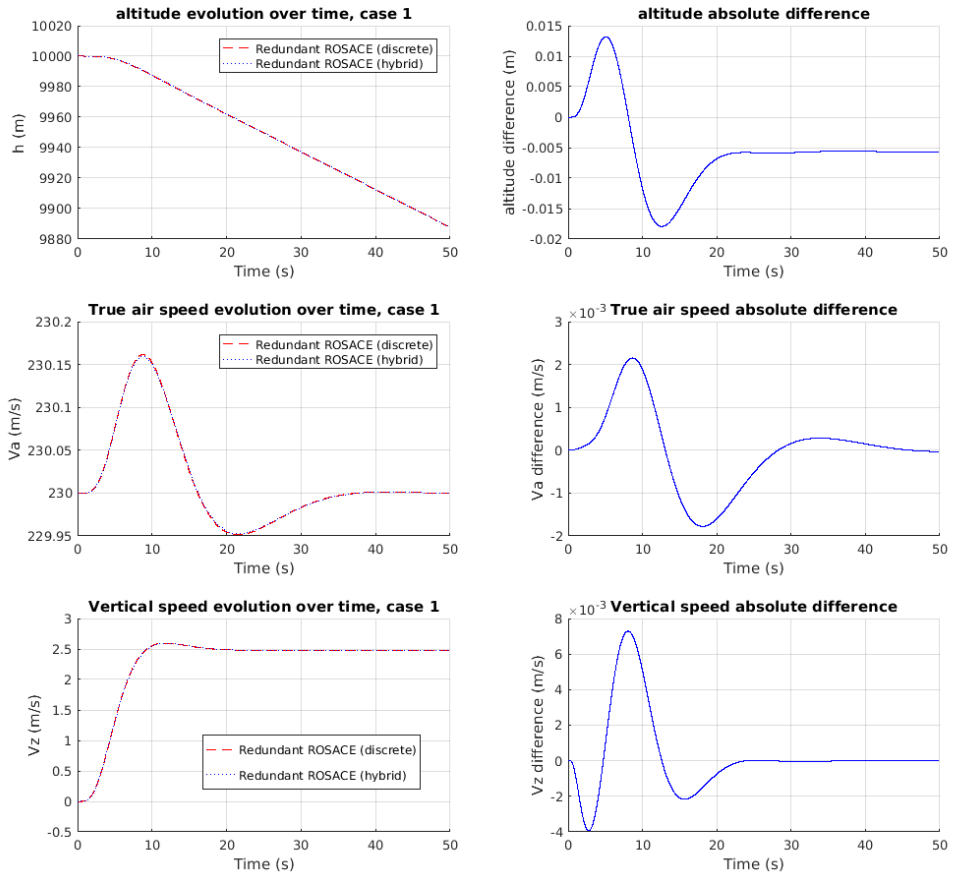


Figure 7.8 – From discrete to cyber-physical models

7.3.4.3 Error injection

The addition of the error injection feature is only useful for checking the case study, in particular to verify that the case tolerates a single error.

?? shows the error injection on δ_{e_c} from FCC1 and FCC2, their detection by the FCCs MON (translated by toggling relays), and the swapping of master in law.

The FCC1 relay is correctly toogle on when the error start, and off when it ends. Ditto for FCC2.

When an error is injected on FCC1 commands, FCC2 become the new master in law. When the error stops on FCC1 commands, FCC2 still the master in law, until an error occurs on FCC2 commands. Then FCC1 is master in law again, as expected.

?? shows the impact on true air speed, inertial vertical speed and inertial altitude values.

There are no differences between the hybrid Redundant ROSACE model with errors injection, and the one without errors injection. We can conclude that the redundancy mechanism works correctly.

So we concluded that we could use our case study RROSACE as part of the allocation test with constraints.

7.4 Method and tool validation

Once the academic case study was validated itself, it was possible to use it to validate the method and the tool.

7.4.1 Academic validation of concepts

The case study is simple enough not to have many constraints, so it was simple to manipulate the scheduling to expose coincidence problems on the redundancy of the FCCs. For other constraints, such as latency constraints, they do not exist in RROSACE. The worst latencies we have achieved with our distributions have not revealed a negative impact, so we have added artificial latencies for our validation tests in this case.

The validation of heuristics was not really interesting with the academic case study. Knowing the limits of greedy heuristics, it is quite easy to misorder input models to obtain poor output scheduling, and vice versa.

All the resources of the academic study are given in the Appendix A.

As an illustration, Listing 7.6 is the result of the allocation with the first-fit heuristic. The allocator always took the first one, and placed the components in it. Using this LP does not break any constraints, so the allocator does not try to manipulate another LP.

Listing 7.6 – rrosace_first_fit.is

```
<?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>
```

Listing 7.7 is the result of the allocation with the best-fit heuristic, the allocator always tries to use a LP as lightly loaded as possible. Not having limited the number of LPs, the allocator creates an empty LP for each component.

Listing 7.7 – rrosace_best_fit.is

```
<?xml version="1.0" ?>
<allocation name="alloc" xmlns="">
    <logical_processor> <task name="engine" ord="0" period="50ms"/> </
        logical_processor>
    <logical_processor> <task name="elevator" ord="0" period="50ms"/> </
        logical_processor>
    <logical_processor> <task name="flight_dynamics" ord="0" period="50
        ms"/> </logical_processor>
    <logical_processor> <task name="h_filter" ord="0" period="100ms"/> <
        /logical_processor>
    <logical_processor> <task name="az_filter" ord="0" period="100ms"/>
        </logical_processor>
    <logical_processor> <task name="Vz_filter" ord="0" period="100ms"/>
        </logical_processor>
    <logical_processor> <task name="q_filter" ord="0" period="100ms"/> <
        /logical_processor>
    <logical_processor> <task name="Va_filter" ord="0" period="100ms"/>
        </logical_processor>
    <logical_processor> <task name="fcu" ord="0" period="200ms"/> </
        logical_processor>
    <logical_processor> <task name="flight_mode" ord="0" period="200ms"/
        > </logical_processor>
    <logical_processor> <task name="fcc_1a" ord="0" period="200ms"/> </
        logical_processor>
    <logical_processor> <task name="fcc_1b" ord="0" period="200ms"/> </
        logical_processor>
    <logical_processor> <task name="fcc_2a" ord="0" period="200ms"/> </
        logical_processor>
    <logical_processor> <task name="fcc_2b" ord="0" period="200ms"/> </
        logical_processor>
    <logical_processor> <task name="wiring" ord="0" period="50ms"/> </
        logical_processor>
</allocation>
```

Listing 7.8 is the result, compressed, of the allocation with the best-fit heuristic with an additional artificial affinity constraint between body vertical acceleration (a_z) and v_z filters. The best-fit adapted its behavior to the placement of the filter to respect the affinity constraint.

Listing 7.8 – rrosace_best_fit_with_affinity.is

```
<?xml version="1.0" ?>
<allocation name="alloc" xmlns="">
    <logical_processor>
       <task name="engine" ord="0" period="50ms"/>
    </logical_processor>
    <!-- ... -->
    <logical_processor>
       <task name="az_filter" ord="0" period="100ms"/>
```

```
9         <task name="Vz_filter" ord="1" period="100ms"/>
10    </logical_processor>
11    <!-- ... -->
12    <logical_processor>
13        <task name="wiring" ord="0" period="50ms"/>
14    </logical_processor>
15 </allocation>
```

Listing 7.9 is the allocation error message with the best-fit heuristic when limiting the number of LPs to 2, by enabling errors in case of non-verifiable constraints. The best fit alternates the allocation of components in order to fill the LPs in an equivalent way, until the FCCs are processed, which are then allocated so as not to break the coincidence constraints between COM and MON.

Listing 7.9 – RROSACE allocation problem

```
1 No candidate logical processor found during allocation of component
      wiring. Try another heuristic or modify SLA or SEA. Current
      allocation:
2 Allocation alloc:
3     logical_processors:
4         logical processor:
5             tasks:
6                 task name=engine period=50ms
7                 task name=flight_dynamics period=50ms
8                 task name=az_filter period=100ms
9                 task name=q_filter period=100ms
10                task name=fcu period=200ms
11                task name=fcc_1a period=200ms
12                task name=fcc_2a period=200ms
13        logical processor:
14            tasks:
15                task name=elevator period=50ms
16                task name=h_filter period=100ms
17                task name=Vz_filter period=100ms
18                task name=Va_filter period=100ms
19                task name=flight_mode period=200ms
20                task name=fcc_1b period=200ms
21                task name=fcc_2b period=200ms
```

The allocation fails to place the wiring, which cannot receive the data from the FCCs at the same time. However, it is easy to change the order of the input components, and force the allocator to succeed in its allocation, as presented in Listing 7.10.

Listing 7.10 – rrosace_best_fit_2_lps.is

```
1 <?xml version="1.0" ?>
2 <allocation name="alloc" xmlns="">
3     <logical_processor>
4         <task name="engine" ord="0" period="50ms"/>
5         <task name="elevator" ord="1" period="50ms"/>
6         <task name="flight_dynamics" ord="2" period="50ms"/>
7         <task name="h_filter" ord="3" period="100ms"/>
```

```
8         <task name="wiring" ord="4" period="50ms"/>
9         <task name="Vz_filter" ord="5" period="100ms"/>
10        <task name="q_filter" ord="6" period="100ms"/>
11        <task name="fcu" ord="7" period="200ms"/>
12    </logical_processor>
13    <logical_processor>
14        <task name="fcc_1a" ord="0" period="200ms"/>
15        <task name="fcc_1b" ord="1" period="200ms"/>
16        <task name="fcc_2a" ord="2" period="200ms"/>
17        <task name="fcc_2b" ord="3" period="200ms"/>
18        <task name="az_filter" ord="4" period="100ms"/>
19        <task name="Va_filter" ord="5" period="100ms"/>
20        <task name="flight_mode" ord="6" period="200ms"/>
21    </logical_processor>
22 </allocation>
```

The existence of few constraints on the other hand made the SA quickly converge towards a solution with the maximum energy of the system. RROSACE allowed us to validate our method as it was developed, and allowed us to more easily implement the allocation tool by doing quick tests, but it did not allow us to get much information on the quality of our solutions, such as the convergence speed of the SA, for example.

7.4.2 Feedback of industrial experience

Once the allocation tool was implemented and validated with RROSACE, we tested it at Airbus on an industrial case study. The analysis and results are only accessible to Airbus. The feedback we can provide is that we wanted to see the scalability of our tool on a solution with hundreds of models and dozens of constraints.

The generation of scheduling with simple heuristics was too much subject to the input order of the models and few requirements could be validated most of the time. The application of the SA to on the other hand was enriching, and we found that the quality of the solution depended mainly on the choice of budget time and the estimation of the SEAs. However, we were unable to run a generated scheduling on a SEA, our attempts were unsuccessful due to adapter output problems, such as major frames that were too large to be scheduled as additional specific constraints, or specific allocation constraints that we had not identified *a priori*. This did not invalidate our method, but there is a lack of time and effort to generate functional adapters, and to take into consideration additional execution constraints, specific to Airbus simulators.

Part V

Conclusion

"The entire history of software engineering is that of the rise in levels of abstraction. Executable UML is the next logical, and perhaps inevitable, evolutionary step in the ever-rising level of abstraction at which programmers express software solutions. Rather than elaborate an analysis product into a design product and then write code, application developers of the future will use tools to translate abstract application constructs into executable entities. Someday soon, the idea of writing an application in Java or C++ will seem as absurd as writing an application in assembler does today. And the code generated from an Executable UML model will be as uninteresting and typically unexamined as the assembler pass of a third generation language compiler is today. "

– Grady Booch in "The Limits of Software" Lecture, 2002.

"All problems in computer science can be solved by another level of indirection... "

– Attributed to David Wheeler in Diomidis Spinellis, Another Level of Indirection, Beautiful Code, 2007.

"...But that usually will create another problem. "

– Attributed to David Wheeler in Diomidis Spinellis, Another Level of Indirection, Beautiful Code, 2007. Less quoted second line.

Table of Contents

8 Summary of contributions **151**

8.1 CPS simulation scheduling 151

8.2 RROSACE and SEAplanes 152

8.3 SLA, SEA and allocation method 153

8.4 Allocation tool 153

9 Perspectives **155**

9.1 Academic perspectives 155

9.2 Industrial perspectives 157

Chapter 8

Summary of contributions

Contents

8.1 CPS simulation scheduling . . . 151
8.1.1 Problem definition . . . 151
8.1.2 CPS scheduling concepts . . . 152
8.2 RROSACE and sEAplanes . . . 152
8.2.1 RROSACE case study . . . 152
8.2.2 sEAplanes framework . . . 152
8.3 sLA, sEA and allocation method . . . 153
8.3.1 Formalizing CPS scheduling . . . 153
8.3.2 Manipulating CPS scheduling . . . 153
8.4 Allocation tool . . . 153
8.4.1 Allocation tool implementation . . . 153
8.4.2 Scheduling optimization . . . 154
8.4.3 Industrial application . . . 154

The objective of this chapter is to summarize the contributions made to this Ph.D. thesis. A list of publications referenced in this section can be found in chapter .

8.1 CPS simulation scheduling

The first part of our research work involved the definition of the scheduling of a CPS simulation.

8.1.1 Problem definition

Considering the current complexity of CPS simulations, the potential impact of the model execution order, and the lack of tools to manipulate it, Airbus and ISAE have established collaborations including this Ph.D. thesis.

As a preliminary to the problem of scheduling, we have shown that these model execution orders are a scheduling, and we have defined the scope of our work.

We have carefully identified the Airbus needs, the assumptions on which to base ourselves, particularly on the breakdown of simulation models, their interactions, and the confidence that can be given to them.

We presented this work through a poster at the CPS summer school [Des16b].

8.1.2 CPS scheduling concepts

We have adapted RT concepts from the CPS simulation available to us, namely the ones used by ISAE and Airbus, and have defined the simulation tasks, their instances, and their execution in parallel and sequential mode.

We then proceeded to express the simulation constraints in the form of scheduling constraints, and expressed the most possible complete set within the limits of the examples of accessible simulations.

8.2 RROSACE and SEAplanes

Before implementing the simulation scheduling generation methods and tools, we implemented a case study and a simulation framework.

The objective was to have communication and testing tools available, which could be presented without any intellectual property problems. The main implementation criteria were simplicity and efficiency.

8.2.1 RROSACE case study

We have adapted our case study, Redundant ROSACE, from an existing case study, ROSACE, a longitudinal flight control loop. We did an analysis and extension work on this flight control loop to be able to highlight the simulation constraints we had identified. We then implemented simulation models that could be used at the ISAE and at Airbus. We made a feedback to the scientific team that worked on the original ROSACE, and published, on their website, our models [Des16a] and an electronic paper [DCCS17a].

8.2.2 SEAplanes framework

We then implemented a simulation framework based on an implementation of the RTI of HLA standard, CERTI, for interoperability within the ISAE, whose behavior was similar to an airbus simulation framework. This simulation framework, called SEAplanes [Des17], for SEA proto-LP Allocation Nodes with Extensible inline Scheduler, allows to easily implement a simulation using models, while completely controlling the scheduling aspect. Subsequently, and to simplify the tests and presentation, we implemented a Qt extension allowing to probe, export, and display the exchanged data [Des18c].

In parallel, we have manipulated this framework at the ISAE, to show the interoperability of this famework with ptolemy-HLA and Matlab with HLA toolbox, as well as the integration of code generated by Matlab, through our own case studies, during internships and projects.

We linked these two contributions and executed RROSACE on SEAplanes, with multiple schedulings.

8.3 SLA, SEA and allocation method

Our main research work consisted in setting up a formalism to express simulation scheduling, and a method to manipulate this scheduling.

8.3.1 Formalizing CPS scheduling

In order to express the scheduling of CPS simulation, we studied the scientific literature to adapt an existing language. We have found that the differences between simulation as such, its execution, and the requirements to be verified on a simulation, which can be impacted by its execution, do not allow for a language to express everything directly today.

We have chosen to take inspiration from some of the languages we have studied such as DEVS and AADL in order to propose two formalisms. the first one, the Simulation Logical Architecture, allows to express the composition of a simulation, the links between the components, and the properties that we wish to verify, by expressing them in the form of simulation requirements. The second formalism, the Simulation Execution Architecture, allows to describe the simulation execution architecture.

This work was published in [DCCS17b] [DCCS18a] and presented a at [Des18b].

8.3.2 Manipulating CPS scheduling

Based on the formalisms established previously, we have defined a method of scheduling manipulation. By coupling these two formalisms, it is possible to estimate the execution of the SLA components on the SEA architecture, and to verify the SLA requirements. The scheduling problem is then reduced to an allocation problem. The two main functions of allocation are partitioning, and mapping, respectively creating clusters of unordered models, and scheduling models in a cluster.

8.4 Allocation tool

The previous method may be applied to simulations with a large number of components and constraints, so we decided to implement a tool allowing automatic allocation [Des18a].

8.4.1 Allocation tool implementation

In order to implement an allocation tool, we have adapted our formalism to make them computer-readable. We have implemented modules to transform formalisms into objects that

can be manipulated by a computer tool, modelled the execution behavior of architectures and implemented functions to verify, on an allocation, whether a requirement is validated or not. As there is no direct solution to the allocation problem, we have implemented heuristics, whose operation consists in taking the components as inputs, trying to allocate them and checking at each step the requirements to validate as many as possible.

We presented this work at an international conference [DCCS18b].

8.4.2 Scheduling optimization

Allocation heuristics can produce results, but they are sometimes of very poor quality. More precisely, certain limitations, such as the impossibility of moving a component already allocated during an allocation making allocations very dependent on inputs, have led us to explore alternatives. In order to no longer depend on the order of inputs, and to avoid being trapped in local maxima, we decided to study the metaheuristic approach, and in particular the Simulated Annealing technique. We have successfully adapted the SA annealing to our allocation problem, in particular by linking the notion of neighbourhood to partitions and mappings, and the notion of energy to the weights of constraints.

8.4.3 Industrial application

Finally, we have used our tool in an industrial context. The tool, a PoC, lacked maturity and did not achieve better results than manual scheduling, largely because of the fuzzy areas on the simulation configurations that did not allow the development of a full-scale adapter. The experiences have nevertheless provided us with feedback that has allowed us to improve the tool locally. The results of the application with the academic case study lead us to believe that its adaptation problem is mostly a time consumption one. The elements of this work are specific to Airbus.

Chapter 9

Perspectives

Contents

9.1 Academic perspectives 155
9.1.1 Domain extension 155
9.1.2 Performance evaluation 156
9.1.3 Mastering scheduling 156
9.1.4 sEAplanes improvements 156
9.2 Industrial perspectives 157
9.2.1 Refinement of the allocation tool 157
9.2.2 Tool for guided deployment 157
9.2.3 Parallel catalyst and cost optimizations 157

The work provided in this doctoral thesis is at the crossroads between the academic and industrial worlds, perspectives exist both for the academic world and for the industrial world.

9.1 Academic perspectives

9.1.1 Domain extension

As part of the financing of the work on this Ph.D. thesis, it seemed more relevant to us to limit our field of application to Airbus simulations. A significant academic perspective is the extension to new domains. Some work may have a limited impact on the method and tools we have developed, such as adding new requirements, or attributes to our components. Other works may have a more significant claim, such as considering the impact of unreliable models in the scheduling.

This is particularly true today for the automotive world, which would benefit greatly from using our method to accelerate simulations, in order to validate *a priori* the emerging

behaviours of autonomous vehicles capable of learning, while at the same time the models historically used for automotive simulations are less accurate than those used in aeronautics.

Over the long-term, aeronautics will also be able to develop skills in simulating the behaviour emerging from autonomous vehicles through feedback from automotive experiences.

9.1.2 Performance evaluation

The objective of our work having been to obtain a method, relatively little time could have been allocated to the implementation of efficient tools. An interesting academic work could be to implement an open version of the allocation tool and study in detail the impact of all the parameters of the simulation and its expression on both the quality of the results of the tool, but also on the effectiveness of the tool itself.

This work could also be an opportunity to address more metaheuristics, and possibly to design one specifically for model allocation. For example, heuristics giving importance to models concerned with substantial weight requirements, and exploring solutions by favouring their allocation.

9.1.3 Mastering scheduling

A limitation in our results, which is not inherent in our method, is the control of the exact execution times of model instances. Adding time budget to estimate execution times is a common solution in languages describing execution such as AADL, and we have not attempted, as part of this thesis, to find a more efficient solution. On the other hand, we have identified research that addresses the exact estimation of computer program execution times on architectures [BCRS10], and we believe that taking this work into account in our solution could have an extremely positive impact, commensurate with the efforts required.

9.1.4 sEAplanes improvements

The current use of sEAplanes is the rapid deployment of distributed simulation.

Performances analysis should be carried out. This analysis could provide us with an important feedback that could teach us to better estimate communications performance and help us work on optimization issues in runtime. For example, by identifying the models that communicate most with each other, to bring them together, and thus limit communications between LPs, which is generally more expensive.

Furthermore, sEAplanes is now limited to static scheduling. This is completely voluntary, and very common in the aviation industry, but without going as far as the use of dynamic scheduling, some improvements could have an interesting impact on the use of resources.

One of these improvements could be the relocation of components. Depending on the phases of the simulation, the components will not be solicited in the same way. We can see it on general public flight simulators on a limited computer, it is often during the takeoff and landing phases that we observe the most lag. The interest with sEAplanes would be to be able to relocate components during runtime relocate using more LPs in order to be able to

access more resources when it needs them, and relocate to fewer when it no longer needs these resources so that they are saved, and eventually available for another SEAplanes.

In addition, relocating could ensure that the functionalities are checked only when it is appropriate. For example, constraints can be released and optimal placement facilitated for much of the simulation if what you want to verify will only occur for a few seconds at landing. This could be interesting if two features whose constraints cannot be validated at the same time on a static scheduling are in fact checked during two different simulation phases.

9.2 Industrial perspectives

9.2.1 Refinement of the allocation tool

The allocation tool presented in this work is a PoC, and is considered as such. Thus the tool was developed in a limited time, with minimal testing to ensure functionality, and extensive user feedback to identify additions between several revisions. The tool, as important as it was for the validation of allocation concepts, is not yet mature enough to be integrated into a significant industrial process, and the next important industrial step for its use is the refinement into a cleaner tool.

9.2.2 Tool for guided deployment

Once the tool is refined, the main industrial perspective is its implementation in design and maintenance processes. This is particularly true since in an mbse approach with rapid revision cycles, access to a tool that generates invisibly a scheduling of its models for the user, subtly highlighting all the unverifiable simulation requirements, could be a significant asset.

9.2.3 Parallel catalyst and cost optimizations

Orthogonally to engineering assistance, one of the industrial challenges today is the adequacy of resources. Our method implemented in a clean industrial tool would allow to easily deploy a simulation on an ephemeral architecture, during the time of the said simulation. Although it may sound like buzz-words, we still consider that we should not neglect the possibility of using our work to help industrials to parallelize their simulations, limit communications and idle time, and deploy on ephemeral architectures or even rent as cloud computing could be, but also in a more pretentious way by imagining a deployment on IT resources not used in a company as servers or underused computers, or in an enterprise 4.0 by using the power to calculate waste all the small non-critical computer systems often on standby.

Sixième partie

Résumé en Français de la thèse

“L’ordinateur s’installe chaque jour de plus en plus dans notre vie quotidienne. Ainsi que nous prenions le métro, le train ou bien l’avion, c’est lui qui régule le trafic et veille sur notre sécurité. Bientôt il nous sortira des embouteillages et au train où vont les choses on peut se demander quels sont les domaines de notre existence qui lui échapperont. ”

– Michel Chevalet dans “Le futur de l’informatique”, 1970.

Table des matières

10 Introduction **163**

10.1 Contexte et motivation 163

10.2 Approche 165

11 Formalisation des simulations de systèmes cyber-physiques **168**

11.1 Identification du problème 168

11.2 Caractérisation de l'ordonnancement de modèles 173

12 Conception, implémentation et application de la solution **179**

12.1 L'outil d'allocation et ses heuristiques 179

12.2 Le cas d'étude RROSACE 184

13 Conclusion **188**

13.1 Résumé des contributions 188

13.2 Perspectives 191

Chapitre 10

Introduction

Sommaire

10.1 Contexte et motivation 163

10.1.1 Usage industriel de la simulation 164

10.1.2 Processus actuel d'intégration de la simulation 165

10.1.3 Nouvelles exigences de l'automatisation de l'intégration 165

10.2 Approche 165

10.2.1 Analyse des procédés industriels 166

10.2.2 Expression des formalismes, méthode et mise en œuvre des outils . . 166

10.2.3 Mise en œuvre d'un cas d'utilisation simple et d'outils 167

10.2.4 Application au cas d'utilisation industriel avec des outils industriels . 167

10.1 Contexte et motivation

Le travail décrit dans cette thèse de doctorat est un projet joint entre le DISC de l'ISAE-SUPAERO, et l'équipe simulation EYYS d'Airbus Group, encadré par une CIFRE, et partiellement financé par l'ANRT.

Avec la numérisation des moyens de production, et la convergence du monde virtuel, de la conception et de la gestion numérique avec les produits et biens du monde réel, les entreprises investissent dans de nouvelles méthodes et de nouveaux outils pour être à la pointe de l'innovation.

La tendance actuelle est à la flexibilité des moyens de production, à l'intégration d'outils logistiques permettant de suivre des flux d'informations importants et à l'utilisation croissante d'outils de simulation, tout en optimisant l'énergie et les matières premières utilisées.

10.1.1 Usage industriel de la simulation

La simulation a aujourd'hui de nombreuses applications industrielles. En aéronautique, et plus particulièrement au sein d'Airbus, on distingue trois grandes utilisations de la simulation.

10.1.1.1 Simulation pour la formation des pilotes

L'application la plus ancienne et la plus connue de la simulation est la formation des pilotes. Ces simulations nécessitent une représentation moins complexe que la simulation du système, l'importance majeure étant l'expérience du pilote [WLC07, KJ98].

Les simulateurs pour la formation des pilotes sont appelés simulateurs de vol et sont basés sur des ordinateurs, avec une interface optionnelle ressemblant à un avion réel. Un exemple de simulateur, avec un cockpit d'A350, est donné dans la Figure 1.6, p. 11. Bien que considérés comme modernes, les simulateurs de vol ont laissé des traces dès 1910 dans les premiers temps de l'aviation avec le "tonneau antoinette" de Léon Levavasseur, Figure 1.7, p. 12. Ce premier simulateur de vol a été utilisé pour la formation des futurs pilotes, ce qui souligne le besoin identifié très tôt d'utiliser la simulation pour la formation des pilotes.

Cependant, il y a aujourd'hui une tendance émergente dans l'industrie de la simulation pour la formation aux processus de maintenance.

10.1.1.2 Simulation pour la conception des systèmes

Une utilisation plus récente de la simulation est le soutien à la conception de systèmes.

Ces simulations exigent un très haut degré de représentativité, et leur complexité fait qu'elles ne sont que très rarement exécutées en temps réel [ASY82].

Il s'agit d'une utilisation de plus en plus populaire dans l'industrie car elle permet, parallèlement au développement du système ou en avance de phase, d'affiner les exigences en étudiant l'impact des variations de paramètres.

La simulation est maintenant disponible dans de nombreux outils de conception.

10.1.1.3 Simulation pour les tests d'intégration

Il s'agit de l'utilisation la plus récente de la simulation. Utilisation qui n'est pas encore démocratisée.

Certaines industries, parmi lesquelles Airbus est un exemple, produisent des systèmes de plus en plus complexes. C'est beaucoup le cas dans des domaines comme l'aéronautique. Afin de maintenir le contrôle de ces systèmes, ils sont divisés en plusieurs sous-systèmes, qui sont conçus, mis en œuvre, testés et validés indépendamment. La Figure 1.8, p. 13, illustre les différentes possibilités d'intégration de la simulation pendant le cycle en V de développement de produit CPS [SKK+12].

Lorsque des systèmes sont intégrés, des problèmes peuvent survenir. Il s'agit de problèmes d'intégration, qui peuvent avoir de nombreuses origines, par exemple une mauvaise communication des interfaces, ou des aspects temporels oubliés.

Afin d'identifier ces problèmes le plus tôt possible et de travailler sur les sous-systèmes existants sans introduire de régression, la simulation peut être utilisée. La prochaine étape dans

l'industrie est la création de jumeaux numériques, qui fournit une représentation numérique du produit tout au long de son cycle de vie. Bien que cela dépasse la portée de notre travail, nous pouvons considérer ce travail comme un pas vers les jumeaux numériques.

Le Tableau 1.1, p. 12, résume les types de simulation.

10.1.2 Processus actuel d'intégration de la simulation

Les composants de simulation peuvent être des modèles, du matériel intégré ou du code logiciel. Les modèles et le logiciel intégré sont considérés comme valides et représentatifs de l'intégration, seule la validation de l'intégration est à considérer à ce stade.

Aujourd'hui, la représentativité de cette intégration est obtenue empiriquement, la première intégration a été comparée à des vols réels, et les intégrations suivantes dépendent d'intégrations antérieures, avec parfois de fortes modifications et comparaisons avec des vols réels.

C'est un moyen efficace et pragmatique de gérer l'intégration des composants, mais trop rigide et impossible à intégrer dans des méthodes et des outils qui implique une forte variation des composants utilisés.

10.1.3 Nouvelles exigences de l'automatisation de l'intégration

L'émergence récente des capacités de calculs et de stockage informatique catalyse l'intégration de logiciels dans les processus industriels. Cette intégration répond a des besoins d'optimisation de cout, à la fois par l'accélération des déploiements, et par l'utilisation plus limitée des autres ressources.

Dans notre société, l'émergence de l'industrie 4.0 est une illustration de cette tendance. Airbus group fait partie des acteurs incontournables de l'évolution des conditions de travail. Le département simulation d'Airbus, EYYS a souhaité partiper à cet effort de modernisation, et à proposer de mettre en place une alternative aux moyens industriels courants de production de simulation.

Le DISC de l'ISAE-SUPAERO a publié par le passé des travaux sur la distribution de simulation, avec une première approche d'un formalisme de simulations distribués. Cette base de travaux est un terreau fertile pour proposer des nouvelles méthodes, et nouveaux outils, afin de maîtriser l'exécution temporelle d'une simulation distribuée.

10.2 Approche

Historiquement, Airbus a fait des investissements considérables dans la modélisation et la génération de systèmes critiques, en particulier pour les ordinateurs de bord, et c'est donc un choix naturel pour Airbus de développer une approche et des outils pour gérer l'intégration des composants de simulation.

Notre approche repose sur l'analyse des raisons empiriques qui ont conduit à l'intégration des simulations existantes, l'analyse des besoins en outillage industriel des acteurs de la simulation et l'analyse des outils de simulation existants, tant chez Airbus qu'à l' ISAE-SUPAERO.

Ces analyses ont permis de prendre des décisions sur l'orientation des travaux de recherche, les formalismes, les méthodes et les outils, et de choisir les travaux de thèse majeurs :

- État de l'art academique.
- Analyse de simulation d'Airbus.
- Définition et vérification de la méthode.
- Définition de l'étude de cas industrielle.
- Conception, mise en œuvre, validation et optimisation des outils.

Les étapes de vérification et de validation ont nécessité l'utilisation d'études de cas, et nous n'aurions pas pu attendre d'avoir l'étude de cas industrielle, ni mettre en œuvre efficacement notre méthode avec une étude de cas qui aurait été trop complexe. Nous avons donc décidé de mettre en place en parallèle de nos travaux la création d'une étude de cas simple, utilisant une simulation open-source de CPS de quelques composants, et un cadre de simulation permettant la diffusion des résultats.

La Figure 1.9, page 17, illustre ces tâches et leurs interdépendances.

10.2.1 Analyse des procédés industriels

La première étape de notre travail a consisté à analyser les méthodes et outils actuels d'Airbus, en particulier la génération des ordonnancements et les cadriciels de simulation existants.

Les ordonnancements actuels des simulations sont obtenus empiriquement, ils satisfont les besoins de simulation pour valider les tests, mais il y a des limitations par rapport à la modernisation des outils, notamment dans l'ajout, la suppression et la modification des modèles, qui impliquent un traitement manuel de la modification de l'ordonnancement. Ce traitement ne permet pas de concrétiser une numérisation totale des processus et des méthodes.

Il était essentiel de comprendre le rôle des intégrateurs de simulation, premiers acteurs en contact avec l'ordonnancement de la simulation, afin d'identifier clairement comment exprimer l'ordonnancement de la simulation, et de comprendre les mécanismes de modification impliqués dans la nouvelle génération d'ordonnancement.

Il était également nécessaire de comprendre comment les modèles sont divisés et regroupés afin de déterminer quelles informations sont facilement accessibles pour générer un ordonnancement, et d'intégrer ces modèles dans un cadre de simulation.

Enfin, il a été nécessaire d'interagir avec les parties prenantes telles que les experts en simulation, les fournisseurs et les clients de simulation pour comprendre leurs besoins du point de vue de la planification de la simulation afin d'orienter nos solutions de projet.

10.2.2 Expression des formalismes, méthode et mise en œuvre des outils

Nous avons fait un état de l'art des formalismes existants liés à la simulation distribuée en temps réel. Nous avons identifié plusieurs concepts précieux dans un certain nombre de forma-

lismes, mais il n'en existe aucun qui soit directement applicable à notre problème particulier, sans un travail important de modélisation de l'architecture d'exécution de simulation.

Nous avons donc décidé de mettre en œuvre notre propre formalisme, qui s'inspire de la littérature scientifique.

10.2.3 Mise en œuvre d'un cas d'utilisation simple et d'outils

Pour démontrer nos concepts, fournir des résultats partiels et évaluer nos outils, nous avons mis en place une étude de cas simple et un cadre de simulation en laboratoire. L'objectif principal est d'investir très tôt dans des outils, et de pouvoir faire des itérations, afin de valider notre travail, et si nécessaire, de l'améliorer progressivement.

10.2.4 Application au cas d'utilisation industriel avec des outils industriels

La dernière partie du projet a consisté à appliquer notre outil à une véritable étude de cas industriel chez Airbus. L'objectif principal est de valider l'évolutivité de notre travail dans une étude de cas complexe.

Cette étape a également permis d'identifier les éléments non précisés dans les documents de conception et les guides d'utilisation des cadres de simulation, et a également conduit au développement du prototype de l'outil industriel.

Chapitre 11

Formalisation des simulations de systèmes cyber-physiques

Sommaire

11.1 Identification du problème 168
11.1.1 Ordonnancement de simulation de CPS 168
11.1.2 Ordonnancement de simulation et ordonnancement de systèmes temps-réels 171
11.2 Caractérisation de l'ordonnancement de modèles 173
11.2.1 Le Simulation Logical Architecture 173
11.2.2 Le Simulation Execution Architecture 174
11.2.3 Allocation de l'architecture logique sur l'architecture d'exécution . . 176
11.2.4 Validation des exigences 177

11.1 Identification du problème

11.1.1 Ordonnancement de simulation de CPS

Dans cette partie, nous présentons notre principale contribution, qui est de générer, *a priori*, un ordonnancement de simulation valide de CPS. Il s'agit de définir ce que nous entendons par ordonnancement de simulation de CPS, et selon quels critères un ordonnancement de simulation peut être considéré comme valide.

Nous considérons que chaque composant de la simulation est suffisamment représentatif, pour cela, nous nous appuyons sur les connaissances existantes en ingénierie de modèles. Nous n'abordons pas le problème de la parallélisation, ou de la distribution des simulations de grands modèles, notre point de départ est un ensemble de composants produits par des experts en ingénierie des modèles et en ingénierie de simulation distribuée.

Ces hypothèses permettent d'exprimer l' ordonnancement de simulation CPS par la composition des composants de simulation existants. Les logiciels utilisés pour intégrer les composants de simulation sont les cadres et les intergiciels de simulation.

Par définition, l'ordonnancement de simulation est une propriété émergente des cadriciels et intergiciels, qui peut être abstraite dans une vue d'un ordonnanceur, ordonnançant les modèles comme indiqué dans la Figure 4.1, p. 58.

Les concepts de cadriciel et d'intergiciels se sont développés sans tenir compte du contrôle de l'ordre d'exécution de leurs composants. Néanmoins, cet ordre existe, il y a toujours un ordre implicite d'exécution. Les simulations sont, *in fine*, des composants logiciels, avec des séquences d'instructions à exécuter dans un ordre précis, tout comme les cadriciels et intergiciels.

L'ensemble des cadriciels et intergiciels de simulation est beaucoup trop varié pour bénéficier d'une rationalisation et d'une modélisation efficace de leurs comportements, cela équivaudrait à répéter et éventuellement renommer un formalisme de description de l'exécution d'un logiciel informatique. Afin de proposer une solution plus simple et plus utilisable, il est nécessaire d'aborder un plus petit groupe de cadriciels et d'intergiciel de simulation, et plus précisément un sous-ensemble de ce groupe suffisamment représentatif. Pragmatiquement, il est nécessaire de se recentrer sur le besoin principal d'identifier les cadres et les intergiciels à traiter, dans notre cas ceux d'Airbus DSS et ASPIC, et un intergiciel de l'ISAE-SUPAERO, CERTI.

11.1.1.1 Distributed Simulation Scheduler

DSS est un cadriciel multiplate-forme d'Airbus pour l'espace utilisateur, utilisant les modèles AP2633. Il est basé sur une architecture à deux niveaux. Un premier niveau, le haut niveau, gère la synchronisation et les échanges dans la simulation. Un deuxième niveau, le bas niveau, met en œuvre l'exécution de la simulation.

La vue de haut niveau de l'ordonnancement de la simulation est illustrée dans la Figure 4.2, p. 60.

Ces niveaux sont mis en œuvre par deux types d'acteurs, illustrés dans fig :dss, p. 61 :

- a **Global scheduler** : un couple CM, CC, calculant des cycles courts et longs à partir des informations données par les ordonnanceurs locaux, gérant la période de synchronisation, et les exécutions des ordonnanceurs locaux.

- **Local schedulers** : un LC, piloté par l'ordonnanceur global, exécute séquentiellement chaque modèle AP2633 fourni en fonction de la configuration.

11.1.1.2 ASPIC

ASPIC est un autre cadriciel Airbus utilisant les modèles AP2633.

ASPIC a une architecture monolithique, intégrée avec le noyau Linux. Les tâches périodiques sont réservées par ASPIC, et les modèles AP2633 sont affectés à ces tâches. Les modèles AP2633 de la même tâche doivent avoir les mêmes périodes d'exécution, égales à la période de la tâche Linux.

Le choix de l'allocation des modèles AP2633 se fait a priori sur les tâches, de la même manière que DSS via les fichiers de configuration, et le noyau Linux programme ensuite ces tâches. La vue de haut niveau de l'ordonnancement de la simulation est illustrée dans la Figure 4.6, p. 63.

L'exécution d'une simulation avec ASPIC est principalement déléguée au noyau Linux. Les tâches Linux en temps réel sont exécutées par un ordonnanceur de noyau, plus précisément le RMS. Il n'y a pas de synchronisation, cette exécution est dite asynchrone.

La communication aussi s'effectue également à l'aide du noyau Linux, en écrivant et en lisant dans des espaces mémoire partagés au moment de l'exécution du modèle.

Cette méthode d'exécution est très efficace par rapport à DSS, mais il est difficile d'obtenir un système déterministe.

11.1.1.3 CERTI : une implémentation open-source de RTI

CERTI [BS02] est une implémentation de la norme de simulation distribuée HLA [II10].

Des fédérés communiquent par le biais d'un RTI, et les fédérations utilisent des mécanismes de publication/abonnement pour échanger des données. Le RTI assure également la gestion du temps entre les fédérés, donc la synchronisation qui peut, comme le DSS, être synchronisée avec l'horloge murale.

Plus spécifiquement dans CERTI, le RTI est distribué entre un RTIG et des RTIAs. Le RTIG, global, gère la création et la destruction des fédérations, la publication/abonnement des données, et la gestion du temps. Le RTIA, local, vise à alléger le travail du RTIG en effectuant les mêmes requêtes que le RTIG, pour lequel il peut être résolu localement sans passer par le réseau.

Cette architecture de simulation est illustrée dans la Figure 4.7, p. 65. Une vue plus détaillée de l'architecture CERTI se trouve dans [NRS09].

Notre première contribution dans ce travail a été de fournir un mécanisme a CERTI pour l'exécution séquentielle des modèles. Plus spécifiquement, un fédéré est un processus avec un seul fil d'exécution, le parallélisme étant largement couvert par la norme HLA. Un processus composé d'un seul *thread* est un ensemble d'instructions à exécuter dans un ordre prédéterminé.

Si, dans le même fédéré, des instructions sont exécutées non pour un seul modèle, mais pour un modèle puis pour un deuxième modèle, c'est comme si les deux modèles avaient été exécutés séquentiellement. Avec cette approche, nous avons mis en place un cadriciel de simulation appelé SEAplanes. Plus spécifiquement, en utilisant un point d'entrée ainsi que DSS pour ordonnancer les modèles, et la machine d'état AP2633 pour l'implémentation des modèles, il est possible de trouver un comportement similaire à DSS avec CERTI.

La vue de haut niveau de l'ordonnancement de la simulation est illustrée dans la Figure 4.8, p. 66 et l'architecture de simulation est illustrée dans la Figure 4.9, p. 67.

Les fédérés sont équivalents aux LCs de DSS, exécutant les modèles séquentiellement, et l'interaction entre les fédérations par RTI fournit un comportement similaire au couple CC / CM.

La principale différence réside dans les échanges de données, qui ne sont pas des échanges par paires entre les fédérés, comme dans DSS mais par le biais du RTI.

11.1.2 Ordonnancement de simulation et ordonnancement de systèmes temps-réels

L'analyse des cadriciels précédents permet de mettre en évidence les tendances récurrentes dans la gestion des ressources des cadres de simulation.

Indépendamment des architectures de simulation, on peut noter qu'il existe deux niveaux d'exécution, un niveau supérieur dans lequel les entités sont exécutées en parallèle, avec ou sans synchronisation, et un niveau inférieur, plus proche de la machine, dans lequel les modèles sont exécutés en séquence, illustré dans la Figure 4.10, p. 72. Cette stratégie d'exécution introduite par A. S. Tanenbaum dans [Tan87] en tant qu'ordonnancement à deux niveaux est connue pour être efficace dans le traitement de l'ordonnancement qui implique un changement de contexte.

Dans l'ordonnancement temps-réel, le composant qui est ordonnancé s'appelle une tâche. Ces tâches ont été définies en arrière-plan, dans sous-section 2.3.1.

Dans l'ordonnancement de simulation, l'entité dont le comportement correspond le mieux à une tâche périodique est le modèle. Les modèles aussi sont exécutés périodiquement, avec un début et une limite. La différence majeure entre une tâche et un modèle est l'exécution d'une instance. Dans une tâche générique, l'instance est souvent considérée comme atomique, mais pour un modèle, plusieurs étapes ont lieu à chaque instance :

1. **Read** les données des autres modèles.
2. **Compute** la sortie de l'instance courante.
3. **Write** les résultats pour les autres modèles.

Cette relation est illustrée dans la Figure 4.11, p. 74.

Dans l'ordonnancement des tâches, l'une des contraintes les plus importantes est la contrainte de précédence. Cette contrainte est définie dans [Lam78] comme une relation entre deux activités i et j, par $i \rightarrow j$, ce qui signifie que j ne peut commencer avant i. *A priori*, la communication entre les composants de simulation peut être convertie en contraintes de précédence. Par exemple, si un composant A produit des données consommées par un composant B, A doit être exécuté avant B.

Cependant dans les CPS, il y a des boucles algébriques. Un composant A peut produire une donnée pour un composant B, et B pour A. Ces boucles algébriques conduisent à deux contraintes contradictoires. La solution est de trouver une contrainte de précédence qui peut être relâchée dans une boucle algébrique et de rompre la boucle. Par exemple, si les données produites par B pour A peuvent être retardées, la contrainte de précédence entre B et A est supprimée.

Dans les systèmes complexes comme les avions, les boucles algébriques sont courantes, et certains systèmes sont dans les boucles multiples. Les flux de données sont beaucoup plus similaires à un maillage qu'à un anneau, comme illustré dans la Figure 4.12, p. 75.

L'assouplissement des contraintes de précédence provient des spécialistes des exigences des systèmes simulés et des modélisateurs d'environnement. Les systèmes, comme l'avionique, sont conçus pour tolérer certains retards lors de l'échange de données, et les modélisateurs d'environnement peuvent concevoir des composants de simulation pour tolérer ces

retards. Plus précisément, les temps de latence minimum et maximum peuvent être associés aux données échangées entre deux composants dans une simulation, ainsi qu'à un long trajet de données entre deux composantes séparées par des composantes multiples. Tant que ces latences sont respectées, des précédences peuvent être établies et la simulation restera représentative.

L'ordonnancement des tâches et des modèles peut être concilié de manière simple, mais il existe de légères différences en termes de concept et d'équivalence d'échelle. Les concepts équivalents entre ordonnancement temps réel et l'ordonnancement de simulation sont listés dans le tableau 4.2, p. 76.

11.1.2.1 Conditions d'admissibilité

L'ordonnancement des modèles étant similaire dans sa définition à l'ordonnancement temps réel, l'acceptabilité d'une solution à un problème d'ordonnancement peut être exprimée de la même manière que celle d'une solution d'ordonnancement temps réel.

Ainsi, une solution à un problème d'ordonnancement de simulation de CPS est acceptable si tous les modèles de simulation peuvent être exécutés, en respectant les contraintes exprimées. Plus exactement, comme les modèles sont périodiques, il est possible de valider l'ordonnancement sur une trame.

La difficulté vient alors du fait que pour choisir un ordonnancement a priori, il faut le modéliser.

11.1.2.2 Tolérance des ordonnancements

Il y a cependant une différence majeure entre l'ordonnancement de la simulation et l'ordonnancement de la simulation.

Un ordonnancement de simulation CPS est utile pour valider par simulation des systèmes, mais ces validations nécessitent de contrôler la fidélité de la simulation, un concept non encore abordé du point de vue de l'ordonnancement. En d'autres termes, l'ordonnancement de simulation peut être une solution valable à un problème d'ordonnancement, mais ne présente aucun intérêt du point de vue d'un utilisateur de simulation.

Ainsi, au-delà de la notion d'acceptabilité de l'ordonnancement, il faut se préoccuper de la validité du résultat produit par l'ordonnancement.

Les modèles de simulation étant considérés comme valides, cette fidélité repose sur l'intégration des modèles, et en particulier sur le respect des contraintes du système simulé décrit dans la simulation.

L'assouplissement de contraintes de précédence dans une simulation CPS provient de la tolérance des communications du CPS simulé. Afin d'obtenir l'ordonnancement le plus utile, il est alors nécessaire d'identifier a priori toutes les contraintes qui peuvent être exprimées sur le système simulé, impactées par l'ordonnancement de la simulation.

11.2 Caractérisation de l'ordonnancement de modèles

Un ordonnancement décrit, à travers un formalisme donné, l'exécution de tâches et l'allocation de ressources dans le temps, afin d'être conforme a des objectifs, tout en respectant des contraintes.

Les éléments de l'ordonnancement de la simulation CPS peuvent être divisés en deux catégories :

- Une catégorie liée à la simulation, telle que tous les modèles qui composent une simulation, les liens qu'ils entretiennent entre eux, et avec les CPS simulés.
- Une catégorie liée à l'exécution de simulation, plus proche de l'informatique, qui couvre l'exécution des modèles et leurs communications physiques, ainsi que leurs interactions avec l'environnement d'exécution.

La formalisation de ces deux catégories ensemble, tout en restant ouverte à l'extension, est une tâche complexe.

L'approche suivie consiste à séparer l'architecture des modèles de simulation de l'architecture d'exécution de la simulation. Chacune de ces architectures est ensuite décrite dans un formalisme adapté à ses spécificités, tandis que les formalismes sont suffisamment proches pour rester compatibles.

Un premier formalisme, le SLA, permet d'exprimer les éléments de la catégorie simulation. DEVS semblait approprié à l'expression de cette catégorie, nous avons repris des éléments de DEVS.

Un second formalisme, le SEA, permet d'exprimer l'architecture informatique qui va exécuter la simulation.

Enfin, la compatibilité entre les deux formalismes est assurée par l'allocation, qui permet d'estimer la programmation des modèles SLA sur le SEA, précisant quelles exigences sont vérifiables par l'allocation.

Cette méthode d'allocation est illustrée dans la Figure 5.1, p. 81.

11.2.1 Le Simulation Logical Architecture

Les simulations de CPS sont composées de deux types de composants, les composants physiques et les cyber-composants.

11.2.1.1 Caractérisation des cyber-composants

Les cyber systèmes modélisés dans les simulations CPS sont, par exemple, les FCS, HMI ou les périphériques réseau (routeurs, commutateurs...). Ce sont des processus exécutés par ordinateur.

Un processus est une entité informatique, souvent appelée tâche. Aujourd'hui, la plupart des OS ne permettent pas la communication directe entre les processus, en particulier, c'est le cas de l'avionique [HJ12]. Dans ces systèmes, les différents processus sont divisés dans l'espace et le temps. Ces notions de partitions sont importantes pour nos composants, elles permettent

de définir des modèles de sous-systèmes totalement isolés, en dehors de la communication explicite. En matière de formalisme, cela nous permet d'utiliser une notion de bus pour la communication entre nos composants, tandis que l'ordonnancement des tâches peut être modélisé par une exécution périodique des modèles.

11.2.1.2 Caractérisation des composants physiques

Selon leurs caractéristiques, les modèles en temps continus doivent être décrits par des équations. Dans notre travail, nous nous concentrerons sur les ODEs. La caractérisation d'ODE pour la simulation distribuée avec des contraintes temps réel est traitée dans [CSSA16]. Nos simulations de CPS étant soumises à des contraintes en temps réel, nous nous concentrerons uniquement sur des méthodes numériques qui peuvent s'adapter à la limitation des ressources de calcul (en espace et en temps).

11.2.1.3 Proposition de modèle atomique générique

Le formalisme suivant est largement inspiré de DEVS, cependant, nous avons introduit la fréquence des composants, et adapté les concepts DEVS à notre contexte. Nous avons introduit des informations opérationnelles sur les CPS simulés, par exemple, les temps de latence minimum et maximum sur les chemins de données, l'expression des périodes simplifie l'estimation du temps pris par un ensemble de composants afin de produire une donnée. De plus, la définition d'un composant dans les premières phases de la conception d'une simulation peut évoluer rapidement, et la modification de la fréquence d'un composant est plus rapide que la redéfinition de la fonction de progression dans le temps. Enfin, le couplage des composants doit être flexible. À l'échelle des composants, les événements d'entrée et de sortie ne sont pas pris en compte, mais les ports. Deux ports connectés par un canal produisent et consomment des données à des fréquences définies par les fréquences des composants connectés. Ainsi, nous introduisons des ensembles de fonctions de transition et de fonctions de sortie, contrairement à DEVS qui considère une fonction interne et une fonction externe.

Nous exprimons le modèle générique des composants c illustrés avec quelques simplifications dans la figure 5.2, p. 85 comme les tuples de l'équation 5.1, p. 83.

Nous proposons de modéliser un SLA complet comme l'ensemble de ses composants, associé aux canaux entre les composants, et à des exigences, comme dans l'équation 5.4, p. 85.

Nous pouvons vérifier certaines propriétés, par exemple, chaque port d'entrée requis par les composants est fourni, où les exigences sont cohérentes. Néanmoins, le SLA ne permet pas de vérifier les propriétés sur l'exécution de la simulation, il faut maintenant modéliser l'architecture d'exécution capable d'exécuter des instances de l'architecture logique de simulation.

11.2.2 Le Simulation Execution Architecture

11.2.2.1 Proposition d'ADL

Dans cette section, nous voulons définir un modèle de SEA. Le SEA peut être instancié de plusieurs façons allant d'un simple programme avec un seul thread à une architecture distribuée complexe. Notre modèle doit donc être suffisamment abstrait pour représenter cette diversité.

Une architecture d'exécution de simulation peut être considérée comme un ensemble non vide de processeurs logiques, les processeurs logiques pouvant exécuter les composants définis dans l'équation 5.1, p. 83, comme des tâches périodiques. Plusieurs modèles peuvent être exécutés par un seul processeur logique, néanmoins, les processeurs logiques respectent la notion de partition dans le temps et dans l'espace. Dans ce travail, nous appellerons le processus de liaison de composants multiples à un processeur logique un *clustering*. Exécuter le SEA implique la distribution et le *clustering* des composants.

Les tâches d'un processeur logique existent dans un domaine séquentiel, tandis que les processeurs logiques existent dans un domaine concurrent. De plus, selon l'implémentation de l'architecture, le domaine concurrent peut être un domaine parallèle, où les processeurs logiques peuvent fonctionner totalement simultanément. Nous avons emprunté ces notions de domaine séquentiel et concurrent au VHDL [BFMR12].

- Communication intra-processeur, directe entre les composants.
- Communication inter-processeur, entre LPs pendant les phases de synchronisation des processeurs logiques.

Les différents types de communication peuvent avoir des natures différentes, *e.g*. mémoire partagée ou communication réseau, impliquant une différence de performances. La distribution des composants sur des processeurs logiques, et les communications créent la notion de ressources nécessaires pour exprimer un ordonnancement, et les composants sont les tâches programmables.

Nous sommes capables d'exprimer le SEA comme un ordonnanceur à deux niveaux :

- Un ordonnanceur global : ordonnance les clusters.
- Des ordonnanceurs locaux : ordonnancent des tâches dans les clusters.

L'ordonnanceur global n'a pas de vue sur les tâches des ordonnanceurs locaux. Il n'y a pas de synchronisation directe entre deux tâches lorsqu'elles sont sur deux processeurs logiques différents.

11.2.2.2 Instanciation des cadriciels

Dans SEAplanes, les modèles sont exécutés en fédérés. Les modèles sont des composants de simulation codés en dur ou importés dans une fédération à partir d'une bibliothèque, l'interface de la bibliothèque étant similaire aux équations 5.2, p. 84 et 5.3, p. 84. L'ordonnanceur global est la synchronisation des fédérations, réalisée par HLA, et en particulier par les RTIG et RTIAs. Ces mêmes composants permettent la communication inter-processeur. Enfin, la communication intra-processeur est la mémoire partagée sur fédéré.

La représentation de l'implémentation de DSS comme SEA est simple, les modèles sont les modèles AP2633, intégrant des implémentations de composants de simulation. Les processeurs logiques et les ordonnanceurs locaux sont le LCs, l'ordonnanceur global, l'ordonnancement du LCs est le CC. La communication intra-processeur est la mémoire partagée.

La communication inter-processeur est la communication P2P par le réseau, sur période de synchronisation définie par l'équation 4.3, p. 62.

L'instanciation d'ASPIC est similaire à celle de DSS, les modèles étant les modèles AP2633 et les processeurs logiques et les ordonnanceurs locaux étant les tâches de l'OS, et l'ordonnanceur global est l'ordonnanceur de l'OS. Néanmoins, les tâches de l'OS ne sont pas capables d'ordonnancer des modèles multi-péridioques, donc quand on représente ASPIC avec un SEA, les LPs n'ordonnancent que les modèles avec une période donnée. La communication entre l'intraprocesseur et l'interprocesseur est la même, réalisée grâce à une mémoire partagée. L'impact sur le SEA en tant que tel est minime, bien que cette rupture dans le partitionnement des communications rende plus complexe l'estimation des temps de communication avec une représentation ASPIC.

Table 5.1, p. 89 résume les instanciation de DSS, ASPIC et SEAplanes comme SEA.

11.2.3 Allocation de l'architecture logique sur l'architecture d'exécution

La distribution et le *clustering* de composants programmables à partir d'un SLA sur une implémentation SEA se fait en deux étapes : partitionnement et projection. Les composants du SLA ont des ports connectés par des canaux. Nous voulons diviser notre ensemble de composants en sous-ensembles, en allouant un processeur logique pour chaque sous-ensemble, et chacun des canaux sera une communication inter-processeur ou intra-processeur lorsqu'il sera porté dans un SEA, selon la partition et la projection. Dans [DFRS92], les notions de partitionnement et de projection sur une architecture parallèle modulaire sont traitées afin de réduire le nombre d'éléments de commutation et de minimiser les temps de communication. Nous adaptons ces notions dans notre travail, avec des contraintes et des objectifs différents. Par exemple, en raison des temps de latence minimum et maximum sur les canaux, nous ne cherchons pas des méthodes pour réduire les temps de communication, mais pour assurer la cohérence entre les contraintes et le partitionnement/projection.

11.2.3.1 Le partitionnement

Le partitionnement de SLA consiste à diviser l'ensemble des composants SLA en sous-ensembles non ordonnés. A partir de ce partitionnement, nous sommes capables d'identifier le type de canaux qui seront instanciés entre composants, soit intra-processeur pour composant dans un même cluster, soit inter-processeur. Au cours de cette étape, si nous avons déjà des informations sur l'implémentation de SEA, nous pouvons éliminer certaines partitions. La Figure 5.4, p. 90 illustre quelques partitions possibles pour un seul ensemble de composants.

11.2.3.2 La projection

L'ensemble des tâches dans un processeur logique est ordonné. Projeter des composants d'une partition à des tâches d'un processeur logique implique la définition d'une séquence.

Les ensembles ordonnés de tâches sont exécutés séquentiellement par les ordonnanceurs locaux des processeurs logiques, tandis que l'ensemble non ordonné des processeurs

logiques est exécuté par l'ordonnanceur global. Pour être plus formel, nous pouvons décrire le partitionnement et la projection comme les définitions de fonctions suivantes :

- Soit C un ensemble des composants du SLA.
- Soit S un ensemble des partitions de C, au sens mathématique.
- Soit O un ensemble ordonné de tâches.
- Soit P soit un ensemble non ordonné de O.

$$partitionnement : C \rightarrow S$$
$$projection : S \rightarrow P$$

Connaissant l'implémentation de SEA, nous sommes capables de vérifier les exigences, telles que les exigences de latence des canaux, et d'adapter notre partitionnement et notre projection. Plus précisément, la décision de regrouper des composants dans le même sous-ensemble est indirectement motivée par les limitations d'implémentation de SEA et les implémentations de composants, puisque le SLA ne considère pas l'exécution.

11.2.4 Validation des exigences

Ces exigences proviennent de l'analyse de simulation des CPS existants.

11.2.4.1 La contrainte de précédence

La contrainte de précédence est l'expression simple de l'exécution d'un modèle avant un autre, comme la contrainte classique de précédence de l'ordonnancement temps réel. Cette contrainte est illustrée dans la Figure 5.5, p. 92.

11.2.4.2 La contrainte de latence

La contrainte de latence a été introduite pour résoudre le problème des boucles dans les simulations. De telles boucles empêchent l'expression de contraintes de simulation basées uniquement sur la précédence. L'introduction d'une contrainte de latence permet d'exprimer un ordre possible, qui peut éventuellement être assoupli. La capacité d'exprimer des contraintes de latence rend artificielles les contraintes de précédence, il est préférable d'exprimer des contraintes de latence qui peuvent être liées à des fenêtres de communication ou d'asynchronisme sur le système, plutôt que de donner des contraintes de précédence qui sont faciles à vérifier, mais rigides. Cette contrainte est illustrée dans la Figure 5.6, p. 93.

11.2.4.3 La contrainte de la coïncidence

Dans le SEA, les communications inter-processeur et intra-processeur peuvent avoir des coûts différents sur les temps de latence logiques. La Figure 5.7, p. 93 illustre comment différentes partitions peuvent conduire à différentes latences de temps logiques. Nous appelons la contrainte d'avoir les mêmes latences sur les chemins de données les contraintes de coïncidence.

11.2.4.4 La contrainte d'affinité

La contrainte d'affinité vient des pratiques de l'industrie. Il peut être préférable de placer des composants dans le même processeur logique, par exemple des composants utilisant la même interface, que vous voulez placer sur un nœud donné. Cette contrainte n'a pas de sens par rapport à la simulation, mais elle permet d'indiquer les besoins provenant de l'utilisation de cette simulation. Cette contrainte est illustrée dans la Figure 5.8, p. 95.

Chapitre 12

Conception, implémentation et application de la solution

Sommaire

12.1 L'outil d'allocation et ses heuristiques 179
12.1.1 Des méthodes aux outils 179
12.1.2 Mise en œuvre du cas d'utilisation de l'allocation 181
12.1.3 Heuristiques d'allocation 183
12.2 Le cas d'étude RROSACE 184
12.2.1 Le cas d'étude original 184
12.2.2 Le cas d'étude RROSACE 184
12.2.3 Mise en œuvre de l'étude de cas 185

12.1 L'outil d'allocation et ses heuristiques

12.1.1 Des méthodes aux outils

Un programme informatique qui lit les sources dans certaines langues, et les traduit dans une autre langue, une langue cible, est appelé un compilateur [ASU86], voir la Figure 6.1, p. 102.

La compilation est un ensemble d'opérations, illustré dans la Figure 6.2, p. 102. De nos jours, les compilateurs sont une technologie modernisée qui peut être multilingue, multi-cible, donc les compilateurs sont généralement divisés en une structure à trois étages, comme illustré dans la Figure 6.4, p. 103.

Notre méthode d'allocation est décrite comme prenant différentes langues en entrée, et en sortant un ordonnancement exécutable par ordinateur. L'allocation est donc similaire à une compilation, ce lien est illustré dans la Figure 6.3, p. 103.

Pour implémenter la solution d'allocation, il est nécessaire de mettre le SLA et le SEA sous la forme de langues qui peuvent être comprises par un ordinateur. Il faut ensuite implémenter les front-ends, jusqu'à la génération intermédiaire d'ordonnancement. Une étape suivante pourrait être l'implémentation du middle-end, l'optimisation de cet ordonnancement intermédiaire, mais il n'est pas nécessaire, et dans un souci pragmatique de gagner du temps en créant un PoC, cela ne sera pas adressé dans ce travail. Enfin, la dernière étape consiste à créer les back-ends, afin de traduire l'ordonnancement intermédiaire en ordonnancement des architectures cibles.

12.1.1.1 Réutiliser l'approche MBSE

Les langages que nous souhaitons implémenter sont inspirés MBSE, une approche pourrait être d'étendre le précédent MBSE pour ajouter les fonctionnalités manquantes.

Adapter DEVS semble compliqué, modifier le SLA pour ne pas prendre en compte l'exécution au moment de l'expression compromet l'utilisation des outils existants, et il n'existe aucune approche évidente pour intégrer le SEA puis l'allocation.

L'AADL [FG13] semble plus permissif. Les composants et les canaux du SLA peuvent être exprimés assez directement sous forme de tâches, et il existe des similitudes entre les processus et LPs, ainsi que sur l'allocation. En outre, certains travaux d'allocation semblent faciliter à l'avenir l'amélioration de l'outil [HZPK07]. Cependant, il est impossible aujourd'hui d'exprimer des contraintes opérationnelles, et donc de couvrir la partie exigences du SLA, il faudrait étendre AADL sur ce point.

D'autres approches par UML, SysML et MARTE/CCSL [RJB04, FMS14, AM08] ont été examinées. Dans l'ensemble, UML n'est clairement pas adaptable aux contraintes de temps réel, et MARTE/CCSL ne permet pas une expression facile de SLA, malgré l'existence de travaux prometteurs dans le domaine des CPS [Mal15]. SysML est un bon candidat couvrant tous les besoins d'expression, mais le travail nécessaire pour adapter un outil semble trop long pour une thèse de doctorat PoC. De plus, SysML ne résout pas le problème d'évolutivité qui existe avec AADL.

En résumé, l'approche d'adaptation MBSE est possible avec AADL et SysML, mais dangereuse, car les temps d'adaptation des outils existants sont susceptibles d'être longs, et il n'existe aucune garantie sur l'évolutivité des outils. Par ailleurs, choisir un outil limite demain la réutilisation du PoC pour l'industrialisation.

12.1.1.2 Définir un langage complet

Une approche différente et plus intéressante consiste à mettre en œuvre nos propres langages informatiques. L'implémentation manuelle des langages informatiques pour SLA et SEA nécessiterait la génération de tous les blocs de la compilation, adapté pour produire une allocation, comme indiqué dans la la Figure 6.5.

Cette implémentation du compilateur moderne est maintenant accessible, car des outils existent pour simplifier cette tâche, c'est le cas de flex et bison [Lev09]. Flex permet la génération d'un analyseur lexical, tandis que bison construit un compilateur.

Bien que cette approche soit moins chronophage que l'adaptation MBSE, elle prend encore beaucoup de temps, puisque les quatres blocs de la la Figure 6.5, p. 105 devraient être implémentés pour chaque langue, en plus de l'allocation elle-même.

12.1.1.3 Approche de l'utilisation du langage de structure hiérarchique

Une troisième approche consiste à mettre en œuvre l'outil d'allocation en utilisant un langage défini, mais avec la réutilisation d'un langage déjà existant. Pour ce faire, de nombreux projets utilisent des langages de formatage.

Une telle approche évite d'avoir à implémenter les premiers blocs du compilateur, comme indiqué dans la Figure 6.6, p. 106. Le gain de temps significatif dans l'implémentation d'un PoC fait que cette solution est celle qui a été retenue.

Dans le cadre de notre implémentation, c'est la popularité de la langue qui nous semble être le critère principal. Le PoC doit être facilement compréhensible, et Airbus doit pouvoir l'industrialiser de manière simple. C'est donc XML [BPSM+08], largement utilisé dans les technologies web, mobile, bureautique et serveur, que nous avons choisi comme langue.

12.1.2 Mise en œuvre du cas d'utilisation de l'allocation

Afin de mettre en œuvre l'outil d'allocation, deux front-ends spécifiques doivent être conçus. Un pour chaque langage.

Puisque nous avons opté pour l'utilisation de langages de formatage pour implémenter les SLA et SEA, l'essentiel du travail consiste à établir des équivalences entre les formalismes et le XML, et à implémenter un module par langage, transformant le XML en objet qui peut être manipulé par l'allocateur.

La structure de base du SLA est directement traduite, les composants sont regroupés, ainsi que les canaux et les exigences. Le Listing 6.2, p. 108 présente l'écriture d'un SLA sous forme XML.

Dans le cadre du PoC, nous avons décidé de limiter techniquement nos choix afin de manipuler une implémentation aussi simple que possible, nous avons donc adapté nos composants. Tout d'abord, dans le contexte du simple, on peut considérer des applications avec seulement une fonction d'avancement et de sortie, et des états initiaux connus. De plus, il est plus facile de considérer que cette fonction se trouve simplement dans le nom du modèle auquel elle appartient, donc nous choisissons de supprimer de notre implémentation des composants SLA les éléments qui appartiennent au modèle, et de les référencer uniquement par le nom du modèle. Les autres éléments explicites sont ceux nécessaires à l'allocateur, à savoir les ports du modèle, sa période, provenant de sa fonction de progression et de sortie, et son budget temps, cette implémentation du composant se présente sous la forme de l' Équation 6.1, p. 108. Un exemple d'écriture est donné dans le Listing 6.3, p. 109.

Le SLA a la forme d'un graphe, par ses canaux. Une limitation de XML est que les données ne peuvent pas être écrites sous forme de graphe, mais sous forme d'arbre. Il est possible de faire de chaque canal une branche de l'arbre, mais il est nécessaire d'ajouter des informations aux canaux. Pragmatiquement, nous avons décidé d'ajouter les noms des composants d'entrée et de sortie, comme dans l' Équation 6.2, p. 109. Un exemple d'écriture est donné dans

le Listing 6.4, p. 109.

Il est intéressant de pouvoir faire la différence entre une exigence importante qui couvre une fonctionnalité que l'on souhaite observer dans la simulation et une exigence mineure. Nous avons donc ajouté un poids à toutes les exigences. Pour cela, nous ajoutons aux éléments d'exigences un attribut `weight`, et un sous-élément correspondant à un élément d'exigence concret, comme illustré dans l' Équation 6.3, p. 110.

Un exemple de contrainte de précédence entre deux composants, illustrée dans la Figure 6.8, p. 110, est donné sous forme dans le Listing 6.5, p. 110.

La contrainte de latence est exprimée sur un chemin de données. Elle consiste à définir sur un parcours donné un délai maximum, et éventuellement un délai minimum. Un exemple de contrainte de latence, illustrée dans la Figure 6.9, p. 111, est donné dans le Listing 6.6, p. 111.

La coïncidence est caractérisée par un ensemble de chemins de données qui doivent arriver de façon synchrone à un certain point. La coïncidence est une forme de contrainte de latence, mais relative. Un exemple de contrainte de coïncidence, illustrée dans la Figure 6.10, p. 111, est donné dans le Listing 6.7, p. 112.

La contrainte d'affinité consiste à exprimer les composants que l'on s'attend à voir alloués ensemble. Un exemple de contrainte d'affinité, illustrée dans la Figure 6.11, p. 112, est donné dans le Listing 6.8, p. 113.

Le SEA est plus facile à écrire que le SLA. Il ne s'agit pas de la description d'un réseau de composants, mais d'une architecture, dont nous savons déjà qu'elle possède un ordonnanceur à deux niveaux. Le SEA est délibérément peu expressif afin d'ajouter facilement de nouveaux cadriciels de simulation, tout en permettant l'expression concrète de comportements synchrones ou non synchrones, périodiques ou non, afin de donner une indication suffisante à l'outil d'allocation pour faire ses calculs d'ordonnancement et de temps de latence. Le principal travail d'ajout de nouveaux comportements se situe au niveau du module d'allocation dans lequel il est nécessaire d'exprimer clairement comment faire les calculs.

Le Listing 6.9, p. 113, illustre l'écriture du SEA de SEAplanes.

12.1.2.1 Adaptateurs back-end

Actuellement, la génération de LP via l'adaptateur SEAplanes se fait de manière semi-automatique. Les modèles associés à LPs peuvent être développés manuellement, générés automatiquement, par exemple à partir de Matlab, ou reciblés à partir de simulations réelles ou anciennes. Le schéma fonctionnel de la Figure 6.13, p. 115, illustre l'association des modèles de simulation dans les tâches des LPs. La Figure 6.14, p. 116, illustre une allocation d'un SLA sur SEAplanes. L'intégration de ces modèles se fait alors en créant une mémoire partagée pour les échanges internes. Pour les échanges entre LPs, les mécanismes de publication et subscription. Enfin, en étendant les méthodes d'initialisation, de finalisation et d'instance de boucle de simulation, avec respectivement les fonctions d'initialisation, de finalisation et d'avancement. Les fonctions d'avancement sont celles d'avancement et de sortie décrites par l'Équation 5.2, p. 84, et l'Équation 5.3, p. 84. Les fonctions d'initialisation permettent d'allouer de la mémoire pour la gestion des états du modèle, tout en initialisant cette mémoire aux états initiaux de l' Équation 5.1, p. 83. Les fonctions de finalisation détruisent simplement l'espace mémoire réservé à l'initialisation.

Les adaptateurs utilisés chez Airbus ne génèrent pas de code, mais des configurations pour les cadres de simulation DSS et ASPIC. L'exécution des ordonnancements étant assurée par DSS et ASPIC, il n'y a aucune complexité dans les adaptateurs qui ne font que traduire les ordonnancements intermédiaires. Les opérations internes de ces adaptateurs sont spécifiques à Airbus.

12.1.3 Heuristiques d'allocation

Notre problème d'allocation est NP-complet, et similaire au MKP, avec quelques spécificités spécifiques aux simulations de CPS telles que le choix d'un nombre limité ou illimité de LPs, le choix des périodes pour certains composants, et les exigences.

Il n'y a pas de solution directe aux problèmes NP-Complet, mais il est possible de vérifier une solution en temps polynomial. L'approche classique est d'utiliser des heuristiques. L'heuristique est une méthode de calcul, elle donne rapidement un résultat, pas forcément optimal.

12.1.3.1 Heuristiques gloutonnes

Les heuristiques gloutonnes sont des variations des heuristiques les plus connues utilisées pour résoudre les MKP. L'algorithme pour implémenter ces heuristiques est donné dans 1, p. 118. Il est possible d'utiliser le même algorithme pour toutes les heuristiques gloutonnes en adaptant le type de tous les LPs. Le tableau 6.1, p. 117, résume la possibilité d'utiliser un type pour obtenir un comportement de $*$-fit.

12.1.3.2 Recuit simulé

Une des limites de l'heuristique précédente est la forte dépendance entre l'ordre des entrées et la qualité de la sortie. En effet, les heuristiques gloutonnes ne savent pas comment déplacer une composante déjà allouée, même lorsqu'il est évident qu'un déplacement peut améliorer l'allocation produite. Nous avons donc travaillé sur l'implémentation d'autres heuristiques, qui ne connaissent pas ce problème.

Le recuit simulé est l'une des métaheuristiques les plus courantes, dont l'une des spécificités est de faire une recherche dans un voisinage. Le recuit simulé s'inspire de la métallurgie : les cycles de chauffage et de refroidissement (recuit) rendent le métal plus résistant. Il permet de rechercher un maximum ou un minimum global, tout en évitant d'être piégé dans un maximum ou un minimum local. Une énergie est définie, puis est réduite au fur et à mesure que le recuit simulé avance.

L'algorithme utilisé, 2, p. 119, nécessite moins d'adaptations que l'heuristique gloutonne, en partie parce qu'il est conçu en prenant en compte un certain nombre de paramètres à choisir. Les choix que nous pouvons faire pour ces paramètres sont présentés dans le tableau 6.2, p. 119.

12.2 Le cas d'étude RROSACE

Afin de valider notre méthode lors de son développement, et d'effectuer des tests lors de l'implémentation de l'outil, nous avons décidé de mettre en place une étude de cas simple et précise.

Nous avons décidé de capitaliser sur nos expériences en construisant notre étude de cas sur un modèle open-source identifié dans la littérature scientifique, et avons implémenté notre solution de manière modulaire et réutilisable pour pouvoir réaliser des tests avec différents cadriciels.

12.2.1 Le cas d'étude original

ROSACE est une étude de cas couvrant différentes étapes de la conception à l'implémentation d'une boucle de vol longitudinal. A l'origine, l'étude de cas de ROSACE a commencé avec le contrôleur développé en Matlab/SIMULINK, et s'est terminée avec un contrôleur multipériodique s'exécutant sur une cible multi/manycore [PSG+14]. Un défi majeur dans la conception d'un contrôleur ROSACE est le besoin d'interactions entre les ingénieurs en contrôle-commande et les ingénieurs logiciels. L'ingénierie de contrôle et l'informatique ne considèrent pas les mêmes problèmes en design, car ces deux disciplines sont techniquement et culturellement séparées. Par exemple, l'informatique ne tient pas compte des exigences des systèmes physiques, comme la stabilité, tandis que l'ingénierie de contrôle ignore l'importance de l'informatique comme l'ordonnancement des tâches et les ressources réseau. Cette question est particulièrement fréquente lors de la conception de CPS [HE11], ce qui témoigne de notre volonté de notre étude se base sur l'étude de cas de ROSACE.

L'objectif de l'étude de cas ROSACE est de valider l'aspect temps réel de l'implémentation du contrôleur.

12.2.2 Le cas d'étude RROSACE

Le cas d'étude original ROSACE a été étendu en ajoutant des contrôleurs redondants, ce qui nous permet de jouer avec les erreurs d'ordonnancement.

12.2.2.1 Ajout de la redondance

Trois contrôleurs existent dans le cas d'étude original, sur v_z, v_a et h. Nous les regroupons en un seul composant, appelé FCC, et ajoutons une logique de surveillance, comme décrite dans la Figure 7.1, p. 124. Lors de l'utilisation de plusieurs FCC, il faut déterminer lesquels fourniront les commandes aux actionneurs. D'après l'expérience d'Airbus, nous savons que nous disposons de suffisamment d'informations pour déterminer la commande, mais nous devons introduire un autre composant, un modèle de câblage contenant des commutateurs. Ce composant simule la logique de sélection des commandes à l'aide des relais [BAB+07].

Dans RROSACE, nous allons considérer un couple de FCCs, chacun étant composé d'une commande, et d'un moniteur. Nous ajouterons également le modèle de câblage entre ce couple de FCCs, et les actionneurs.

La Figure 7.4, p. 126, est le graphique des composants de l'étude de cas RROSACE. De plus, la nature de chaque composante est mise en évidence (continue ou discrète).

12.2.3 Mise en œuvre de l'étude de cas

La première étape de l'implémentation de l'étude de cas a été l'analyse de l'étude de cas préexistante, afin d'identifier clairement la couverture fonctionnelle par les composants de simulation, et de perfectionner la modularité de notre solution.

À partir de l'analyse fonctionnelle, nous avons identifié les composants et leurs blocs de simulation. Néanmoins, pour utiliser DSS et SEAplanes pour notre simulation, nous avons besoin d'une base de source commune.

- La bibliothèque doit être facilement lisible.
- La bibliothèque doit être facilement extensible.
- La bibliothèque doit être portable.
- La bibliothèque doit être déterministe.
- La bibliothèque doit avoir le moins d'impact possible sur le temps d'exécution de la simulation.

Pour ce faire, nous utilisons le langage ANSI C avec uniquement des bibliothèques standards [Rit78]. Chaque composant de notre analyse est une instance de classe de bibliothèque de modèles.

Les bibliothèques de modèles contiennent des contextes et des interfaces de modèles. Nous avons suivi le paradigme de l'OOP pour garder une approche simple et compréhensible.

- Un modèle est un contexte avec des fonctions C utilisées pour le manipuler.
- Un modèle est totalement opaque pour l'utilisateur de la bibliothèque.
- Le contexte est une structure opaque.
- La bibliothèque fournit des fonctions pour allouer et libérer le contexte.
- La bibliothèque fournit une fonction pour faire une itération du modèle avec un contexte.

Afin de présenter cette implémentation, nous présentons deux listes, correspondant à l'implémentation de la gouverne :

- Listing 7.1, p. 128 — implémentation de l'interface de la gouverne.
- Listing 7.2, p. 129 — implémentation du corps de la gouverne.

Les autres modèles ont été implémentés de la même manière.

12.2.3.1 Simulation de RROSACE

SEAplanes est basé sur HLA, donc nous avons généré une FOM contenant les attributs et objets de la fédération, Listing 7.3, p. 133. Pour créer une fédération, tous les modèles présents dans ce FOM doivent être simulés. La déclaration des instances d'objets HLA dépend de la répartition de l'allocation. Le Listing 7.4, p. 135, et le Listing 7.5, p. 137, illustrent la création d'un LP, ici pour un LP ordonnançant uniquement la gouverne.

DSS ordonnance les modèles AP2633, en utilisant les fichiers de configuration Les modèles AP2633 contiennent une seule et unique de nos instances de modèles. Chaque modèle AP2633 est construit dans une bibliothèque dynamique avec un principal programmable. Les résultats obtenus avec les simulations DSS sont équivalents aux simulations SEAplanes.

Une fois l'étude de cas académique validée avec SEAplanes et DSS, il a été possible de l'utiliser pour valider la méthode et l'outil.

12.2.3.2 Validation académique des concepts

L'étude de cas est assez simple pour ne pas avoir beaucoup de contraintes, il était donc facile de manipuler l'ordonnancement pour exposer les problèmes de coïncidence sur la redondance des FCCs. Pour d'autres contraintes, comme les contraintes de latence, elles n'existent pas dans RROSACE. Les pires latences que nous avons obtenues avec nos distributions n'ont pas révélé d'impact négatif, nous avons donc ajouté des latences artificielles pour nos tests de validation dans ce cas.

Le Listing 7.6, p. 142, est le résultat de l'allocation avec l'heuristique first-fit.

Le Listing 7.7, p. 143, est le résultat de l'allocation avec l'heuristique best-fit, l'allocateur essaie toujours d'utiliser un LP aussi peu chargé que possible. N'ayant pas limité le nombre de LPs, l'allocateur crée un LP vide pour chaque composant.

Le Listing 7.8, p. 143, est le résultat, compressé, de l'allocation avec l'heuristique best-fit avec une contrainte d'affinité artificielle supplémentaire entre les filtres a_z et v_z. L'heuristique a adapté son comportement au placement du filtre pour respecter la contrainte d'affinité.

Le Listing 7.9, p. 144, est le message d'erreur d'allocation avec l'heuristique best-fit lorsqu'on limite le nombre de LPs à deux, créant des contraintes non vérifiables. L'heuristique alterne l'allocation des composants afin de remplir les LPs de manière équivalente, jusqu'à ce que le FCCs soit traité, qui sont ensuite alloués afin de ne pas briser les contraintes de coïncidence entre COM et MON. L'affectation ne parvient pas à placer le câblage, qui ne peut pas recevoir les données de l'FCCs en même temps. Cependant, il est facile de changer l'ordre des composants d'entrée, et de forcer l'allocateur à réussir son allocation, comme présenté dans le Listing 7.10, p. 144.

RROSACE nous a permis de valider notre méthode telle qu'elle a été développée, et nous a permis de mettre en œuvre plus facilement l'outil d'allocation en faisant des tests rapides, mais cela ne nous a pas permis d'obtenir beaucoup d'informations sur la qualité de nos solutions, telles que la vitesse de convergence du recuit simulé, par exemple. L'existence de peu de contraintes fait converger rapidement le recuit simulé vers une solution avec l'énergie maximale du système.

12.2.3.3 Retour d'expérience industrielle

Une fois l'outil d'allocation implémenté et validé avec RROSACE, nous l'avons testé chez Airbus sur un cas industriel. L'analyse et les résultats ne sont accessibles qu'à Airbus. Le retour d'expérience que nous pouvons fournir est que nous voulions voir le comportement de notre outil sur une solution avec des centaines de modèles et des dizaines de contraintes.

La génération de l'ordonnancement par simple heuristique était trop soumise à l'ordre de saisie des modèles, peu d'exigences pouvaient être validées la plupart du temps. L'application du recuit simulé a été enrichissante, et nous avons trouvé que la qualité de la solution dépendait principalement du choix du temps budgétaire et de l'estimation du SEAs. Cependant, nous n'avons pas été en mesure d'exécuter un ordonnancement généré sur un SEA, nos tentatives étaient infructueuses en raison de problèmes de sortie des adaptateurs, tels que des trames majeures trop grandes pour être ordonnancées comme contraintes spécifiques supplémentaires, ou des contraintes d'allocation spécifiques que nous n'avions pas identifiées *a priori*. Cela n'a pas invalidé notre méthode, mais il y a un manque de temps et d'efforts pour générer des adaptateurs fonctionnels, et pour prendre en compte des contraintes d'exécution supplémentaires, spécifiques aux simulateurs Airbus.

Chapitre 13

Conclusion

Sommaire

13.1 Résumé des contributions 188

13.1.1 Ordonnancement de simulation de CPS 188

13.1.2 RROSACE et SEAplanes 189

13.1.3 Le SLA, le SEA et la méthode d'allocation 190

13.1.4 L'outil d'allocation 190

13.1.5 Application industrielle 191

13.2 Perspectives 191

13.2.1 Perspectives académiques 192

13.2.2 Perspectives industrielles 193

13.1 Résumé des contributions

L'objectif de cette section est de faire un rappel sur l'ensemble des contributions faites dans le cadre de cette thèse de doctorat. Une liste des publications référencées dans cette section se trouve dans chapitre .

13.1.1 Ordonnancement de simulation de CPS

La première partie de notre travail a consisté à définir l'ordonnancement de simulation de système cyber-physique.

13.1.1.1 Définition du problème

Partant du constat de la complexité actuelle des simulations de CPS, de l'impact que peut avoir l'ordre d'exécution de modèles, et du manque d'outils permettant de manipuler celui-ci, Airbus et l'ISAE on mit en place des collaborations, dont cette thèse de doctorat.

Avant de traiter le problème de l'ordonnancement, nous avons montré que ces ordres d'exécutions de modèles sont un ordonnancement, et nous avons défini le périmètre de notre travail.

Nous avons clairement identifié des besoins d'Airbus les hypothèses sur lesquelles partir, en particulier sur le découpage des modèles de simulation, leurs interactions, et la confiance qu'il est possible de leur donner.

Nous avons présenté ce travail par le biais d'un poster à l'école d'été des CPS [Des16b].

13.1.1.2 Concepts de l'ordonnancement de simulation de CPS

Nous avons adapté des concepts du temps réel aux simulations de CPS qui nous étaient accessibles, celle de l'ISAE et d'Airbus, et avons défini les tâches de simulations, leurs instances, et leurs exécutions de manière parallèle et séquentielle.

Nous avons ensuite cherché à exprimer les contraintes de simulation sous forme de contraintes d'ordonnancement, et avons exprimé un jeu le plus complet possible dans la limite des exemples de simulations disponibles.

13.1.2 RROSACE et SEAplanes

Avant de mettre en place les méthodes et outils de générations d'ordonnancement de simulation, nous avons implémenté un cas d'étude et un cadriciel de simulation.

L'objectif était d'avoir à disposition des outils de communication et de test, présentables sans problèmes de propriété intellectuelle. Les critères principaux d'implémantation étaient la simplicité et l'efficacité.

13.1.2.1 Le cas d'étude RROSACE

Nous avons adapté notre cas d'étude, RROSACE, d'un cas d'étude déjà existant, ROSACE, une boucle de vol longitudinale. Nous avons fait un travail d'analyse et d'extension de cette boucle de vol pour pouvoir faire apparaître les contraintes de simulation que nous avions identifiées. Nous avons ensuite implémenté des modèles de simulations utilisables à l'ISAE et chez Airbus. Nous avons fait un retour à l'équipe scientifique ayant travaillé sur le ROSACE originel, et avons publié, sur leur site, nos modèles [Des16a] et un papier électronique [DCCS17a].

13.1.2.2 Le cadriciel SEAplanes

Nous avons ensuite implémenté un cadriciel de simulation basé sur une implémentation d'HLA, CERTI, pour l'interopérabilité au sein de l'ISAE, dont le comportement était similaire à un cadriciel de simulation d'Airbus. Ce cadriciel de simulation, appelé SEAplanes, permet

d'implémenter facilement une simulation en utilisant des modèles, tout en maîtrisant complètement l'aspect d'ordonnancement [Des17]. Par la suite, et pour simplifier les tests et présentations, nous avons implémenté une extension Qt permettant de sonder, exporter, et afficher les données échangées [Des18c].

En parallèle, nous avons manipulé ce cadriciel à l'ISAE, et par le biais de cas d'étude propres, montrer au cours de stages et projets l'interopérabilité de ce cadriciel avec ptolemy-HLA et Matlab et la HLA toolbox, ainsi que l'intégration de code généré par Matlab.

Nous avons lié ces deux contributions et exécuté RROSACE sur SEAplanes, avec de nombreux ordonnancements.

13.1.3 Le SLA, le SEA et la méthode d'allocation

Notre principal travail de recherche a consisté à mettre en place un formalisme pour exprimer l'ordonnancement de simulation, et une méthode pour manipuler cet ordonnancement.

13.1.3.1 Formalisation de l'ordonnancement de simulation de CPS

Afin d'exprimer l'ordonnancement de simulations de CPS, nous avons étudié la littérature scientifique pour adapter un langage déjà existant. Nous avons constaté que les différences entre la simulation en tant que telle, son exécution, et les besoins à vérifier sur une simulation, font qu'aujourd'hui il n'existe pas de langage pour tout exprimer directement.

Nous avons choisi de nous inspirer de certains des langages que nous avons étudiés tel que DEVS et AADL afin de proposer deux formalismes, un premier, le SLA, permettant d'exprimer la composition d'une simulation, les liens qu'entretiennent les composants, et des propriétés que l'on souhaite vérifier, en les exprimant sous forme d'exigences de simulation. Le second formalisme, SEA, permet de décrire l'architecture d'exécution de simulation.

Nous avons présenté le travail de création des langages et de vérification de contraintes au cours de conférences internationales, respectivement [DCCS17b] et [DCCS18a], présenté une seconde fois[Des18b]

13.1.3.2 Manipulation de l'ordonnancement de simulation de CPS

À partir des formalismes mis en place précédemment, nous avons défini une méthode de manipulation d'ordonnancement. En liant ces deux formalismes, il est possible d'estimer l'exécution des composants du SLA sur l'architecture du SEA, et de vérifier les exigences du SLA, le problème d'ordonnancent est alors réduit à un problème d'allocation. Les deux fonctions principales de l'allocation sont le partitionnement, et la projection, respectivement créant des clusters de modèles non ordonnés, et ordonnent des modèles dans un cluster.

13.1.4 L'outil d'allocation

La méthode précédente pouvant être appliquée sur des simulations possédant un nombre important de composants et de contraintes, nous avons décidé de mettre en place un outil permettant l'allocation automatique [Des18a].

13.1.4.1 Implémentation de l'outil d'allocation

Afin d'implémenter un outil d'allocation, nous avons adapté nos formalismes pour les rendre lisibles par un ordinateur. Nous avons implémenté des modules permettant de transformer un formalisme en entrée en objet manipulable par un outil informatique, modélisé les comportements d'exécution des architectures et implémenté des fonctions permettant de vérifier, sur une allocation, si une exigence est vérifiée. Des solutions directes au problème d'allocation n'existant pas, nous avons implémenté des heuristiques, dont le fonctionnement consiste à prendre les composants en entrée, essayer de les allouer et vérifier à chaque étape les exigences pour en valider le plus possible.

Nous avons présenté ce travail au cours d'une conférence internationale [DCCS18b].

13.1.4.2 Optimisation de l'ordonnancement

Les heuristiques d'allocations permettent d'obtenir des résultats, mais ceux-ci sont parfois de très mauvaise qualité. Plus exactement certaines limitations, telles que l'impossibilité de déplacer un composant déjà alloué au cours d'une allocation rendent les allocations très dépendantes des entrées, cela nous a amené à chercher des alternatives. Afin de ne plus dépendre de l'ordre des entrées, et de ne pas être piégés dans des maxima locaux, nous avons décidé d'étudier la piste des métaheuristiques, et en particulier celle du recuit simulé. Nous avons réussi à adapter le recuit simulé à notre problème d'allocation, en particulier en liant la notion de voisinage aux partitions et projections, et la notion d'énergie aux poids des contraintes.

13.1.5 Application industrielle

Enfin, nous avons utilisé notre outil dans un cadre industriel. L'outil, démonstrateur de faisabilité, manque de maturité et n'a pas permis d'obtenir de meilleurs résultats que les ordonnancements manuels, en grande partie à cause des zones floues sur les configurations de simulations qui n'ont pas permis d'obtenir d'adapteur complet. Les expériences nous ont tout de même apporté un retour qui nous a permis d'améliorer localement l'outil, et les résultats de l'application avec le cas d'étude académique nous amènent à penser que le problème de l'adapter n'est qu'une question de temps. Les éléments de ce travail sont propres à Airbus.

13.2 Perspectives

Le travail fourni dans cette thèse de doctorat est à la croisée entre les mondes académiques et industriels, des perspectives existent à la fois pour le monde académique et pour le monde industriel.

13.2.1 Perspectives académiques

13.2.1.1 Extension du domaine d'application

Dans le cadre du financement des travaux de cette thèse de doctorat, il nous a semblé plus pertinent de limiter notre domaine d'application aux simulations d'Airbus. Une perspective académique conséquente est l'extension à de nouveaux domaines. Certains travaux peuvent avoir un impact limité sur la méthode et les outils que nous avons développés, tel que l'ajout de nouvelles exigences, ou d'attributs à nos composants. D'autres travaux peuvent avoir une prétention plus significative, tel que la considération de l'impact de modèles peu fiables dans l'ordonnancement.

C'est particulièrement vrai aujourd'hui pour le monde de l'automobile, qui gagnerait à utiliser notre méthode pour accélérer les simulations, afin de valider *a priori* les comportements émergents de véhicules autonomes capables d'apprendre, alors que les modèles historiquement utilisés pour les simulations automobiles sont moins précis que ceux utilisés en aéronautique.

Sur le long terme, l'aéronautique pourra également développer des compétences de simulation du comportement émergent des véhicules autonomes à partir du retour d'expérience automobile.

13.2.1.2 Évaluation des performances

L'objectif de notre travail ayant été d'obtenir une méthode, relativement peu de temps a pu être consacré à l'implémentation d'outils efficaces. Un travail académique intéressant pourrait être d'implémenter une version ouverte de l'outil d'allocation et d'étudier en détail l'impact de tous les paramètres de la simulation et de son expression à la fois sur la qualité des résultats de l'outil, mais aussi sur l'efficacité de l'outil en lui-même.

Ce travail pourrait aussi être l'occasion d'adresser plus de métaheuristique, et éventuellement d'en concevoir une spécifiquement à l'allocation de modèles. Par exemple, concevoir une heuristique pouvant donner de l'importance aux modèles concernés par des exigences fortes, et explorant des solutions en priorisant l'allocation de ces modèles.

13.2.1.3 Maîtrise de l'ordonnancement

Une limitation dans nos résultats, qui n'est pas inhérente à notre méthode, est la maîtrise des temps d'exécution exacte des instances de modèles. L'ajout de budgets temporels pour estimer les temps d'exécution est une solution commune dans les langages décrivant une exécution tels qu'AADL, et nous n'avons pas cherché, dans le cadre de cette thèse, a trouvé une solution plus efficace. Nous avons en revanche identifié des travaux de recherche qui adresse l'estimation exacte des temps d'exécution de programmes informatiques sur des architectures [BCRS10], et nous pensons que la prise en considération de ces travaux dans notre solution pourrait avoir un impact extrêmement positif, à la hauteur des efforts qu'il faudrait consacrer.

13.2.1.4 Amélioration de SEAplanes

L'utilisation actuelle de SEAplanes est le déploiement rapide de la simulation distribuée.

L'analyse des performances devra être effectuée. Cette analyse pourrait nous fournir une remontée d'information importante qui nous permettrait de mieux estimer les performances des communications et nous aiderait à travailler sur les problèmes d'optimisation durant l'exécution. Par exemple, en identifiant les modèles qui communiquent le plus entre eux, pour les rapprocher, et ainsi limiter les communications entre LPs, ce qui est généralement plus coûteux.

De plus, SEAplanes est maintenant limité à la planification statique. Il s'agit d'une démarche entièrement volontaire et très courante dans l'industrie aéronautique. Cependant, sans aller jusqu'à l'utilisation de la planification dynamique, certaines améliorations pourraient avoir un impact intéressant sur l'utilisation des ressources. L'une de ces améliorations pourrait être le déplacement des composants. Selon les phases de la simulation, les composants ne seront pas sollicités de la même manière. On peut le voir sur les simulateurs de vol grand public sur un ordinateur limité, c'est souvent lors des phases de décollage et d'atterrissage que l'on observe le plus de lenteurs. L'intérêt avec SEAplanes serait de pouvoir relocaliser les composants pendant l'exécution en utilisant plus de LPs afin de pouvoir accéder à plus de ressources quand il en a besoin, et de se relocaliser vers moins de ressources quand il n'en a plus besoin afin qu'elles soient économisées et finalement accessibles pour d'autres SEAplanes.

En outre, le transfert pourrait faire en sorte que les fonctionnalités ne soient vérifiées que lorsque cela est approprié. Par exemple, les contraintes peuvent être relâchées et le placement optimal peut être facilité pour une grande partie de la simulation si ce que vous voulez vérifier ne se produit que pendant quelques secondes à l'atterrissage. Cela pourrait être intéressant si deux caractéristiques dont les contraintes ne peuvent pas être validées en même temps sur un ordonnancement statique sont en fait vérifiées pendant deux phases de simulation différentes.

13.2.2 Perspectives industrielles

13.2.2.1 Affinement de l'outil d'allocation

L'outil d'allocation présenté dans ce travail est un démonstrateur de faisabilité, et est a considéré en tant que tel. Ainsi l'outil a été développé dans un temps limité, avec un minimum de tests s'assurant des fonctionnalités, et de nombreux retours d'utilisation pour identifier les ajouts entre plusieurs révisions. L'outil, aussi important fut-il pour la validation de concepts de l'allocation, n'est pas aujourd'hui assez mature pour être intégré dans un processus industriel conséquent, et l'étape industrielle importante suivante pour son utilisation est l'affinement dans un outil plus propre.

13.2.2.2 Outil pour le déploiement guidé

Une fois l'outil affiné, la principale perspective industrielle est sa mise en place dans des processus de conception et maintenance. Ceci est d'autant plus vrai que dans une approche

MBSE avec des cycles de révisions rapides, l'accès à un outil générant de manière invisible à l'utilisateur un ordonnancement de ses modèles, en soulignant subtilement l'ensemble des exigences de simulation non vérifiable pourrait être un plus non négligeable.

13.2.2.3 Catalysation de parallélisme, et optimisation des coûts

De manière orthogonale à l'assistance d'ingénieur, un des défis industriels aujourd'hui est l'adéquation des ressources. Notre méthode implémentée dans un outil industriel propre permettrait de facilement déployer sur une architecture éphémère une simulation, le temps de ladite simulation. Bien que cela puisse s'apparenter à des buzz words, nous considérons tout de même qu'il ne faille pas négliger la possibilité d'utiliser nos travaux pour aider les industriels à paralléliser leurs simulations, limiter les communications et temps d'attente, et déployer sur des architectures éphémères voir louer comme pourrait l'être l'informatique dématérialisée, mais aussi de manière plus prétentieuse en imaginant un déploiement sur des ressources informatiques non utilisées dans une entreprise comme le serait des serveurs ou ordinateur sous-utilisés, voir dans une entreprise 4.0 en utilisant la puissance de calcul gaspillée le l'ensemble des petits systèmes informatiques non critiques souvent en attente.

Part VII

Appendices

Table of Contents

A RROSACE resources **199**

A.1 Complete RROSACE SLA 199

A.2 RROSACE allocations on SEAplanes 210

Appendix A

RROSACE resources

Contents

A.1 Complete RROSACE SLA 199

A.2 RROSACE allocations on SEAplanes 210

A.2.1 Greedy heuristics 210

A.2.2 Specific allocations 213

A.1 Complete RROSACE SLA

Listing A.1 – rrosace.sla

```
<?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>
      <component name="elevator" period="50ms" time_budget="1ms">
         <ports_in> <port label="delta_e_c"/> </ports_in>
         <ports_out> <port label="delta_e"/> </ports_out>
      </component>
      <component name="flight_dynamics" period="50ms" time_budget="1ms
          ">
         <ports_in>
            <port label="delta_e"/>
            <port label="T"/>
```

```
16            </ports_in>
17            <ports_out>
18                <port label="q"/>
19                <port label="Va"/>
20                <port label="Vz"/>
21                <port label="az"/>
22                <port label="h"/>
23            </ports_out>
24        </component>
25        <component name="h_filter" period="100ms" time_budget="1ms">
26            <ports_in> <port label="h"/> </ports_in>
27            <ports_out> <port label="h_meas"/> </ports_out>
28        </component>
29        <component name="az_filter" period="100ms" time_budget="1ms">
30            <ports_in> <port label="az"/> </ports_in>
31            <ports_out> <port label="az_meas"/> </ports_out>
32        </component>
33        <component name="Vz_filter" period="100ms" time_budget="1ms">
34            <ports_in> <port label="Vz"/> </ports_in>
35            <ports_out> <port label="Vz_meas"/> </ports_out>
36        </component>
37        <component name="q_filter" period="100ms" time_budget="1ms">
38            <ports_in> <port label="q"/> </ports_in>
39            <ports_out> <port label="q_meas"/> </ports_out>
40        </component>
41        <component name="Va_filter" period="100ms" time_budget="1ms">
42            <ports_in> <port label="Va"/> </ports_in>
43            <ports_out> <port label="Va_meas"/> </ports_out>
44        </component>
45        <component name="fcu" period="200ms" time_budget="1ms">
46            <ports_out>
47                <port label="Vz_c"/>
48                <port label="h_c"/>
49                <port label="Va_c"/>
50            </ports_out>
51        </component>
52        <component name="flight_mode" period="200ms" time_budget="1ms">
53            <ports_out> <port label="flight_mode"/> </ports_out>
54        </component>
55        <component name="fcc_1a" period="200ms" time_budget="1ms">
56            <ports_in>
57                <port label="Vz_c"/>
58                <port label="q_meas"/>
59                <port label="Va_c"/>
60                <port label="h_meas"/>
61                <port label="h_c"/>
62                <port label="Vz_meas"/>
63                <port label="flight_mode"/>
64                <port label="az_meas"/>
65                <port label="Va_meas"/>
66            </ports_in>
67            <ports_out>
68                <port label="delta_x_c"/>
```

```
            <port label="delta_e_c"/>
        </ports_out>
    </component>
    <component name="fcc_1b" period="200ms" time_budget="1ms">
        <ports_in>
            <port label="Vz_c"/>
            <port label="q_meas"/>
            <port label="Va_c"/>
            <port label="h_meas"/>
            <port label="delta_e_c_com"/>
            <port label="h_c"/>
            <port label="Vz_meas"/>
            <port label="flight_mode"/>
            <port label="az_meas"/>
            <port label="delta_x_c_com"/>
            <port label="Va_meas"/>
        </ports_in>
        <ports_out>
            <port label="relay_delta_x_c"/>
            <port label="relay_delta_e_c"/>
        </ports_out>
    </component>
    <component name="fcc_2a" period="200ms" time_budget="1ms">
        <ports_in>
            <port label="Vz_c"/>
            <port label="q_meas"/>
            <port label="Va_c"/>
            <port label="h_meas"/>
            <port label="h_c"/>
            <port label="Vz_meas"/>
            <port label="flight_mode"/>
            <port label="az_meas"/>
            <port label="Va_meas"/>
        </ports_in>
        <ports_out>
            <port label="delta_x_c"/>
            <port label="delta_e_c"/>
        </ports_out>
    </component>
    <component name="fcc_2b" period="200ms" time_budget="1ms">
        <ports_in>
            <port label="Vz_c"/>
            <port label="q_meas"/>
            <port label="Va_c"/>
            <port label="h_meas"/>
            <port label="delta_e_c_com"/>
            <port label="h_c"/>
            <port label="Vz_meas"/>
            <port label="flight_mode"/>
            <port label="az_meas"/>
            <port label="delta_x_c_com"/>
            <port label="Va_meas"/>
        </ports_in>
```

```
            <ports_out>
                <port label="relay_delta_x_c"/>
                <port label="relay_delta_e_c"/>
            </ports_out>
        </component>
        <component name="wiring" period="50ms" time_budget="1ms">
            <ports_in>
                <port label="relay_delta_e_c_2"/>
                <port label="relay_delta_e_c_1"/>
                <port label="delta_e_c_2"/>
                <port label="delta_e_c_1"/>
                <port label="delta_x_c_2"/>
                <port label="delta_x_c_1"/>
                <port label="relay_delta_x_c_2"/>
                <port label="relay_delta_x_c_1"/>
            </ports_in>
            <ports_out>
                <port label="delta_x_c"/>
                <port label="delta_e_c"/>
            </ports_out>
        </component>
    </components>
    <channels>
        <channel>
            <from component="engine" port="T"/>
            <to component="flight_dynamics" port="T"/>
        </channel>
        <channel>
            <from component="elevator" port="delta_e"/>
            <to component="flight_dynamics" port="delta_e"/>
        </channel>
        <channel>
            <from component="flight_dynamics" port="h"/>
            <to component="h_filter" port="h"/>
        </channel>
        <channel>
            <from component="flight_dynamics" port="az"/>
            <to component="az_filter" port="az"/>
        </channel>
        <channel>
            <from component="flight_dynamics" port="Vz"/>
            <to component="Vz_filter" port="Vz"/>
        </channel>
        <channel>
            <from component="flight_dynamics" port="q"/>
            <to component="q_filter" port="q"/>
        </channel>
        <channel>
            <from component="flight_dynamics" port="Va"/>
            <to component="Va_filter" port="Va"/>
        </channel>
        <channel>
            <from component="fcu" port="h_c"/>
```

```
            <to component="fcc_1a" port="h_c"/>
        </channel>
        <channel>
            <from component="fcu" port="Va_c"/>
            <to component="fcc_1a" port="Va_c"/>
        </channel>
        <channel>
            <from component="fcu" port="Vz_c"/>
            <to component="fcc_1a" port="Vz_c"/>
        </channel>
        <channel>
            <from component="fcu" port="h_c"/>
            <to component="fcc_1b" port="h_c"/>
        </channel>
        <channel>
            <from component="fcu" port="Va_c"/>
            <to component="fcc_1b" port="Va_c"/>
        </channel>
        <channel>
            <from component="fcu" port="Vz_c"/>
            <to component="fcc_1b" port="Vz_c"/>
        </channel>
        <channel>
            <from component="fcu" port="h_c"/>
            <to component="fcc_2a" port="h_c"/>
        </channel>
        <channel>
            <from component="fcu" port="Va_c"/>
            <to component="fcc_2a" port="Va_c"/>
        </channel>
        <channel>
            <from component="fcu" port="Vz_c"/>
            <to component="fcc_2a" port="Vz_c"/>
        </channel>
        <channel>
            <from component="fcu" port="h_c"/>
            <to component="fcc_2b" port="h_c"/>
        </channel>
        <channel>
            <from component="fcu" port="Va_c"/>
            <to component="fcc_2b" port="Va_c"/>
        </channel>
        <channel>
            <from component="fcu" port="Vz_c"/>
            <to component="fcc_2b" port="Vz_c"/>
        </channel>
        <channel>
            <from component="h_filter" port="h_meas"/>
            <to component="fcc_1a" port="h_meas"/>
        </channel>
        <channel>
            <from component="h_filter" port="h_meas"/>
            <to component="fcc_1b" port="h_meas"/>
```

```
228         </channel>
229         <channel>
230             <from component="h_filter" port="h_meas"/>
231             <to component="fcc_2a" port="h_meas"/>
232         </channel>
233         <channel>
234             <from component="h_filter" port="h_meas"/>
235             <to component="fcc_2b" port="h_meas"/>
236         </channel>
237         <channel>
238             <from component="az_filter" port="az_meas"/>
239             <to component="fcc_1a" port="az_meas"/>
240         </channel>
241         <channel>
242             <from component="az_filter" port="az_meas"/>
243             <to component="fcc_1b" port="az_meas"/>
244         </channel>
245         <channel>
246             <from component="az_filter" port="az_meas"/>
247             <to component="fcc_2a" port="az_meas"/>
248         </channel>
249         <channel>
250             <from component="az_filter" port="az_meas"/>
251             <to component="fcc_2b" port="az_meas"/>
252         </channel>
253         <channel>
254             <from component="Vz_filter" port="Vz_meas"/>
255             <to component="fcc_1a" port="Vz_meas"/>
256         </channel>
257         <channel>
258             <from component="Vz_filter" port="Vz_meas"/>
259             <to component="fcc_1b" port="Vz_meas"/>
260         </channel>
261         <channel>
262             <from component="Vz_filter" port="Vz_meas"/>
263             <to component="fcc_2a" port="Vz_meas"/>
264         </channel>
265         <channel>
266             <from component="Vz_filter" port="Vz_meas"/>
267             <to component="fcc_2b" port="Vz_meas"/>
268         </channel>
269         <channel>
270             <from component="q_filter" port="q_meas"/>
271             <to component="fcc_1a" port="q_meas"/>
272         </channel>
273         <channel>
274             <from component="q_filter" port="q_meas"/>
275             <to component="fcc_1b" port="q_meas"/>
276         </channel>
277         <channel>
278             <from component="q_filter" port="q_meas"/>
279             <to component="fcc_2a" port="q_meas"/>
280         </channel>
```

```
        <channel>
            <from component="q_filter" port="q_meas"/>
            <to component="fcc_2b" port="q_meas"/>
        </channel>
        <channel>
            <from component="Va_filter" port="Va_meas"/>
            <to component="fcc_1a" port="Va_meas"/>
        </channel>
        <channel>
            <from component="Va_filter" port="Va_meas"/>
            <to component="fcc_1b" port="Va_meas"/>
        </channel>
        <channel>
            <from component="Va_filter" port="Va_meas"/>
            <to component="fcc_2a" port="Va_meas"/>
        </channel>
        <channel>
            <from component="Va_filter" port="Va_meas"/>
            <to component="fcc_2b" port="Va_meas"/>
        </channel>
        <channel>
            <from component="flight_mode" port="flight_mode"/>
            <to component="fcc_1a" port="flight_mode"/>
        </channel>
        <channel>
            <from component="flight_mode" port="flight_mode"/>
            <to component="fcc_1b" port="flight_mode"/>
        </channel>
        <channel>
            <from component="flight_mode" port="flight_mode"/>
            <to component="fcc_2a" port="flight_mode"/>
        </channel>
        <channel>
            <from component="flight_mode" port="flight_mode"/>
            <to component="fcc_2b" port="flight_mode"/>
        </channel>
        <channel>
            <from component="fcc_1a" port="delta_e_c"/>
            <to component="fcc_1b" port="delta_e_c_com"/>
        </channel>
        <channel>
            <from component="fcc_1a" port="delta_x_c"/>
            <to component="fcc_1b" port="delta_x_c_com"/>
        </channel>
        <channel>
            <from component="fcc_2a" port="delta_e_c"/>
            <to component="fcc_2b" port="delta_e_c_com"/>
        </channel>
        <channel>
            <from component="fcc_2a" port="delta_x_c"/>
            <to component="fcc_2b" port="delta_x_c_com"/>
        </channel>
        <channel>
```

```
            <from component="fcc_1a" port="delta_e_c"/>
            <to component="wiring" port="delta_e_c_1"/>
        </channel>
        <channel>
            <from component="fcc_1a" port="delta_x_c"/>
            <to component="wiring" port="delta_x_c_1"/>
        </channel>
        <channel>
            <from component="fcc_1b" port="relay_delta_e_c"/>
            <to component="wiring" port="relay_delta_e_c_1"/>
        </channel>
        <channel>
            <from component="fcc_1b" port="relay_delta_x_c"/>
            <to component="wiring" port="relay_delta_x_c_1"/>
        </channel>
        <channel>
            <from component="fcc_2a" port="delta_e_c"/>
            <to component="wiring" port="delta_e_c_2"/>
        </channel>
        <channel>
            <from component="fcc_2a" port="delta_x_c"/>
            <to component="wiring" port="delta_x_c_2"/>
        </channel>
        <channel>
            <from component="fcc_2b" port="relay_delta_e_c"/>
            <to component="wiring" port="relay_delta_e_c_2"/>
        </channel>
        <channel>
            <from component="fcc_2b" port="relay_delta_x_c"/>
            <to component="wiring" port="relay_delta_x_c_2"/>
        </channel>
        <channel>
            <from component="wiring" port="delta_x_c"/>
            <to component="engine" port="delta_x_c"/>
        </channel>
        <channel>
            <from component="wiring" port="delta_e_c"/>
            <to component="elevator" port="delta_e_c"/>
        </channel>
    </channels>
    <requirements>
        <requirement weight="100">
            <coincidence>
                <path>
                    <ord index="0">
                        <channel>
                            <from component="fcc_1a" port="delta_e_c"/>
                            <to component="wiring" port="delta_e_c_1"/>
                        </channel>
                    </ord>
                </path>
                <path>
                    <ord index="0">
```

```
                    <channel>
                        <from component="fcc_1a" port="delta_e_c"/>
                        <to component="fcc_1b" port="delta_e_c_com"/>
                    </channel>
                </ord>
                <ord index="1">
                    <channel>
                        <from component="fcc_1b" port="relay_delta_e_c
                            "/>
                        <to component="wiring" port="relay_delta_e_c_1
                            "/>
                    </channel>
                </ord>
            </path>
        </coincidence>
    </requirement>
    <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>
    <requirement weight="100">
        <coincidence>
            <path>
                <ord index="0">
                    <channel>
                        <from component="fcc_2a" port="delta_e_c"/>
                        <to component="wiring" port="delta_e_c_2"/>
                    </channel>
                </ord>
```

```
436             </path>
437             <path>
438                 <ord index="0">
439                     <channel>
440                         <from component="fcc_2a" port="delta_e_c"/>
441                         <to component="fcc_2b" port="delta_e_c_com"/>
442                     </channel>
443                 </ord>
444                 <ord index="1">
445                     <channel>
446                         <from component="fcc_2b" port="relay_delta_e_c
                                "/>
447                         <to component="wiring" port="relay_delta_e_c_2
                                "/>
448                     </channel>
449                 </ord>
450             </path>
451         </coincidence>
452     </requirement>
453     <requirement weight="100">
454         <coincidence>
455             <path>
456                 <ord index="0">
457                     <channel>
458                         <from component="fcc_2a" port="delta_x_c"/>
459                         <to component="wiring" port="delta_x_c_2"/>
460                     </channel>
461                 </ord>
462             </path>
463             <path>
464                 <ord index="0">
465                     <channel>
466                         <from component="fcc_2a" port="delta_x_c"/>
467                         <to component="fcc_2b" port="delta_x_c_com"/>
468                     </channel>
469                 </ord>
470                 <ord index="1">
471                     <channel>
472                         <from component="fcc_2b" port="relay_delta_x_c
                                "/>
473                         <to component="wiring" port="relay_delta_x_c_2
                                "/>
474                     </channel>
475                 </ord>
476             </path>
477         </coincidence>
478     </requirement>
479     <requirement weight="100">
480         <coincidence>
481             <path>
482                 <ord index="0">
483                     <channel>
484                         <from component="fcc_1a" port="delta_e_c"/>
```

```
                        <to component="wiring" port="delta_e_c_1"/>
                    </channel>
                </ord>
            </path>
            <path>
                <ord index="0">
                    <channel>
                        <from component="fcc_1b" port="relay_delta_e_c
                            "/>
                        <to component="wiring" port="relay_delta_e_c_1
                            "/>
                    </channel>
                </ord>
            </path>
            <path>
                <ord index="0">
                    <channel>
                        <from component="fcc_2a" port="delta_e_c"/>
                        <to component="wiring" port="delta_e_c_2"/>
                    </channel>
                </ord>
            </path>
            <path>
                <ord index="0">
                    <channel>
                        <from component="fcc_2b" port="relay_delta_e_c
                            "/>
                        <to component="wiring" port="relay_delta_e_c_2
                            "/>
                    </channel>
                </ord>
            </path>
        </coincidence>
    </requirement>
    <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_1b" port="relay_delta_x_c
                            "/>
                        <to component="wiring" port="relay_delta_x_c_1
                            "/>
                    </channel>
                </ord>
```

```
532                 </path>
533                 <path>
534                     <ord index="0">
535                         <channel>
536                             <from component="fcc_2a" port="delta_x_c"/>
537                             <to component="wiring" port="delta_x_c_2"/>
538                         </channel>
539                     </ord>
540                 </path>
541                 <path>
542                     <ord index="0">
543                         <channel>
544                             <from component="fcc_2b" port="relay_delta_x_c
                                    "/>
545                             <to component="wiring" port="relay_delta_x_c_2
                                    "/>
546                         </channel>
547                     </ord>
548                 </path>
549             </coincidence>
550         </requirement>
551     </requirements>
552 </sla>
```

A.2 RROSACE allocations on SEAplanes

A.2.1 Greedy heuristics

A.2.1.1 First-fit heuristic

Listing A.2 – rrosace_first_fit.is

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

```
20 </allocation>
```

A.2.1.2 Next-fit heuristic

Listing A.3 – rrosace_next_fit.is

```
 1 <?xml version="1.0" ?>
 2 <allocation name="alloc" xmlns="">
 3     <logical_processor>
 4         <task name="engine" ord="0" period="50ms"/>
 5     </logical_processor>
 6     <logical_processor>
 7         <task name="elevator" ord="0" period="50ms"/>
 8     </logical_processor>
 9     <logical_processor>
10         <task name="flight_dynamics" ord="0" period="50ms"/>
11     </logical_processor>
12     <logical_processor>
13         <task name="h_filter" ord="0" period="100ms"/>
14     </logical_processor>
15     <logical_processor>
16         <task name="az_filter" ord="0" period="100ms"/>
17     </logical_processor>
18     <logical_processor>
19         <task name="Vz_filter" ord="0" period="100ms"/>
20     </logical_processor>
21     <logical_processor>
22         <task name="q_filter" ord="0" period="100ms"/>
23     </logical_processor>
24     <logical_processor>
25         <task name="Va_filter" ord="0" period="100ms"/>
26     </logical_processor>
27     <logical_processor>
28         <task name="fcu" ord="0" period="200ms"/>
29     </logical_processor>
30     <logical_processor>
31         <task name="flight_mode" ord="0" period="200ms"/>
32     </logical_processor>
33     <logical_processor>
34         <task name="fcc_1a" ord="0" period="200ms"/>
35     </logical_processor>
36     <logical_processor>
37         <task name="fcc_1b" ord="0" period="200ms"/>
38     </logical_processor>
39     <logical_processor>
40         <task name="fcc_2a" ord="0" period="200ms"/>
41     </logical_processor>
42     <logical_processor>
43         <task name="fcc_2b" ord="0" period="200ms"/>
44     </logical_processor>
45     <logical_processor>
46         <task name="wiring" ord="0" period="50ms"/>
```

```
47     </logical_processor>
48 </allocation>
```

A.2.1.3 Best-fit heuristic

Listing A.4 – rrosace_best_fit.is

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

A.2.1.4 Worst-fit heuristic

Listing A.5 – rrosace_worst_fit.is

```
1 <?xml version="1.0" ?>
2 <allocation name="alloc" xmlns="">
3     <logical_processor>
4         <task name="engine" ord="0" period="50ms"/>
5         <task name="elevator" ord="1" period="50ms"/>
```

```
6         <task name="flight_dynamics" ord="2" period="50ms"/>
7         <task name="h_filter" ord="3" period="100ms"/>
8         <task name="az_filter" ord="4" period="100ms"/>
9         <task name="Vz_filter" ord="5" period="100ms"/>
10        <task name="q_filter" ord="6" period="100ms"/>
11        <task name="Va_filter" ord="7" period="100ms"/>
12        <task name="fcu" ord="8" period="200ms"/>
13        <task name="flight_mode" ord="9" period="200ms"/>
14        <task name="fcc_1a" ord="10" period="200ms"/>
15        <task name="fcc_1b" ord="11" period="200ms"/>
16        <task name="fcc_2a" ord="12" period="200ms"/>
17        <task name="fcc_2b" ord="13" period="200ms"/>
18        <task name="wiring" ord="14" period="50ms"/>
19    </logical_processor>
20 </allocation>
```

A.2.2 Specific allocations

A.2.2.1 Adding an affinity constraint

Proof of the impact of SLA requirements on allocation, with a an affinity requirements set between a_z and v_z filters:

Listing A.6 – rrosace_with_affinity_req.sla

```
1 <?xml version="1.0" ?>
2 <sla name="rrosace" xmlns="">
3     <!-- ... -->
4     <requirements>
5         <!-- ... -->
6         <requirement weight="100">
7             <affinity>
8                 <component name="az_filter"/>
9                 <component name="Vz_filter"/>
10            </affinity>
11        </requirement>
12    </requirements>
13 </sla>
```

Listing A.7 – rrosace_best_fit_with_affinity.is

```
1 <?xml version="1.0" ?>
2 <allocation name="alloc" xmlns="">
3     <logical_processor> <task name="engine" ord="0" period="50ms"/> </
          logical_processor>
4     <logical_processor> <task name="elevator" ord="0" period="50ms"/> </
          logical_processor>
5     <logical_processor> <task name="flight_dynamics" ord="0" period="50
          ms"/> </logical_processor>
6     <logical_processor> <task name="h_filter" ord="0" period="100ms"/> <
          /logical_processor>
7     <logical_processor>
```

```
8         <task name="az_filter" ord="0" period="100ms"/>
9         <task name="Vz_filter" ord="1" period="100ms"/>
10    </logical_processor>
11    <logical_processor> <task name="q_filter" ord="0" period="100ms"/> <
          /logical_processor>
12    <logical_processor> <task name="Va_filter" ord="0" period="100ms"/>
          </logical_processor>
13    <logical_processor> <task name="fcu" ord="0" period="200ms"/> </
          logical_processor>
14    <logical_processor> <task name="flight_mode" ord="0" period="200ms"/
          > </logical_processor>
15    <logical_processor> <task name="fcc_1a" ord="0" period="200ms"/> </
          logical_processor>
16    <logical_processor> <task name="fcc_1b" ord="0" period="200ms"/> </
          logical_processor>
17    <logical_processor> <task name="fcc_2a" ord="0" period="200ms"/> </
          logical_processor>
18    <logical_processor> <task name="fcc_2b" ord="0" period="200ms"/> </
          logical_processor>
19    <logical_processor> <task name="wiring" ord="0" period="50ms"/> </
          logical_processor>
20 </allocation>
```

A.2.2.2 Limitation of greedy heuristics

Trying to use Best-fit to allocate on 2 LPs leads to allocation failure.

Listing A.8 – RROSACE allocation problem

```
1 No candidate logical processor found during allocation of component
      wiring. Try another heuristic or modify SLA or SEA. Current
      allocation:
2 Allocation alloc:
3     logical_processors:
4         logical processor:
5             tasks:
6                 task name=engine period=50ms
7                 task name=flight_dynamics period=50ms
8                 task name=az_filter period=100ms
9                 task name=q_filter period=100ms
10                task name=fcu period=200ms
11                task name=fcc_1a period=200ms
12                task name=fcc_2a period=200ms
13        logical processor:
14            tasks:
15                task name=elevator period=50ms
16                task name=h_filter period=100ms
17                task name=Vz_filter period=100ms
18                task name=Va_filter period=100ms
19                task name=flight_mode period=200ms
20                task name=fcc_1b period=200ms
21                task name=fcc_2b period=200ms
```

Changing the order of the component in the input allows to do the allocation.

Listing A.9 – rrosace_best_fit_2_lps.is

```
<?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="wiring" ord="4" period="50ms"/>
        <task name="Vz_filter" ord="5" period="100ms"/>
        <task name="q_filter" ord="6" period="100ms"/>
        <task name="fcu" ord="7" period="200ms"/>
    </logical_processor>
    <logical_processor>
        <task name="fcc_1a" ord="0" period="200ms"/>
        <task name="fcc_1b" ord="1" period="200ms"/>
        <task name="fcc_2a" ord="2" period="200ms"/>
        <task name="fcc_2b" ord="3" period="200ms"/>
        <task name="az_filter" ord="4" period="100ms"/>
        <task name="Va_filter" ord="5" period="100ms"/>
        <task name="flight_mode" ord="6" period="200ms"/>
    </logical_processor>
</allocation>
```

This page was intentionally left blank.

Part VIII

References

References

[Ali14] Cesare Alippi. *Intelligence for embedded systems: a methodological approach*. Springer, Cham, 2014. OCLC: 879603636. (Cited on page 21.)

[Alu15] Rajeev Alur. *Principles of cyber-physical systems*. The MIT Press, Cambridge, Massachusetts, 2015. (Cited on pages xxxiii and 19.)

[AM08] Charles André and Frédéric Mallet. Clock Constraints in UML/MARTE CCSL. report, 2008. (Cited on pages 104 and 180.)

[APS98] U. M Ascher, Linda Ruth Petzold, and Society for Industrial and Applied Mathematics. *Computer methods for ordinary differential equations and differential-algebraic equations*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), Philadelphia, Pa., 1998. OCLC: 722506655. (Cited on page 26.)

[ASC10] Martin Adelantado, Pierre Siron, and Jean-Baptiste Chaudron. Towards an HLA Run-time Infrastructure with Hard Real-time Capabilities. 2010. (Cited on page 51.)

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers, principles, techniques, and tools*. Addison-Wesley Pub. Co, Reading, Mass, 1986. (Cited on pages 101 and 179.)

[ASY82] D. M. Auslander, R. C. Spear, and G. E. Young. A Simulation-Based Approach to the Design of Control Systems With Uncertain Parameters. *J. Dyn. Sys., Meas., Control*, 104(1):20–26, March 1982. ISSN 0022-0434. doi: 10.1115/1.3149626. (Cited on pages 11 and 164.)

[AT17] Anastasia Anagnostou and Simon J. E. Taylor. A distributed simulation methodological framework for OR/MS applications. *Simulation Modelling Practice and Theory*, 70:101–119, January 2017. ISSN 1569-190X. doi: 10.1016/j.simpat.2016.10.007. (Cited on page 49.)

[BAB+07] Romain Bernard, Jean-Jacques Aubert, Pierre Bieber, Christophe Merlini, and Sylvain Metge. Experiments in model based safety analysis: Flight controls. *IFAC Proceedings Volumes*, 40(6):43–48, 2007. ISSN 14746670. doi: 10.3182/20070613-3-FR-4909.00010. (Cited on pages 126 and 184.)

[Ban10] Jerry Banks, editor. *Discrete-event system simulation*. Prentice Hall, Upper Saddle River, 5th ed edition, 2010. (Cited on pages xxxvii and 24.)

[Bau13] V. Bauer. Facts and Fallacies of Reuse in Practice. pages 431–434. IEEE, March 2013. doi: 10.1109/CSMR.2013.65. (Cited on page 49.)

[BCRS10] Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat. OTAWA: An Open Toolbox for Adaptive WCET Analysis. In Sang Lyul Min, Robert Pettit, Peter Puschner, and Theo Ungerer, editors, *Software Technologies for Embedded and Ubiquitous Systems*, Lecture Notes in Computer Science, pages 35–46. Springer Berlin Heidelberg, 2010. (Cited on pages 156 and 192.)

[BCSZ98] Pierre Bieber, Jacques Cazin, Pierre Siron, and Guy Zanon. Security Extensions to ONERA HLA RTI Prototype. In *In Proceedings of the 1998 Fall Simulation Interoperability Workshop. Paper number 98F-SIW-086. Available online via doclib/doclib.cfm?SISO_FID_1709> [accessed*, pages 511–516, 1998. (Cited on page 50.)

[BD15] Paolo Bocciarelli and Andrea D'Ambrogio. Chapter 14 - A model-driven method for the design-time performance analysis of service-oriented software systems. In Mohammad S. Obaidat, Petros Nicopolitidis, and Faouzi Zarai, editors, *Modeling and Simulation of Computer Networks and Systems*, pages 425–450. Morgan Kaufmann, Boston, January 2015. doi: 10.1016/B978-0-12-800887-4.00014-6. (Cited on page 49.)

[BEHK14] Veronika Bauer, Jonas Eckhardt, Benedikt Hauptmann, and Manuel Klimek. An Exploratory Study on Reuse at Google. In *Proceedings of the 1st International Workshop on Software Engineering Research and Industrial Practices*, SER&IPs 2014, pages 14–23, New York, NY, USA, 2014. ACM. doi: 10.1145/2593850.2593854. event-place: Hyderabad, India. (Cited on page 49.)

[BFMR12] Jean-Michel Bergé, Alain Fonkoua, Serge Maginot, and Jacques Rouillard. *VHDL Designer's Reference*. Springer Science & Business Media, December 2012. Google-Books-ID: aZvqBwAAQBAJ. (Cited on pages 86 and 175.)

[BK] Peter Brucker and Sigrid Knust. Complexity results for scheduling problems. url: "`http://www2.informatik.uni-osnabrueck.de/knust/class/`". (Cited on page 37.)

[BK06] Peter Brucker and Sigrid Knust. *Complex scheduling*. GOR-Publications. Springer, Berlin, 2006. OCLC: ocm64399391. (Cited on page 37.)

[BKB] Marco Brassé, Arjan J F Kok, and Ernst-Wichard Budde. System Engineering and Integration Aspects of a Multi-National HLA Federation Development. page 8. (Cited on pages 49 and 50.)

[BKK02] Andrei Borshchev, Yuri Karpov, and Vladimir Kharitonov. Distributed simulation of hybrid systems with AnyLogic and HLA. *Future Generation Computer Systems*, 18(6):829–839, May 2002. ISSN 0167-739X. doi: 10.1016/S0167-739X(02)00055-9. (Cited on page 50.)

[BPSM+08] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition) – Review Version, February 2008. url: "`https://www.w3.org/TR/2008/PER-xml-20080205/PER-xml-20080205.xml`". (Cited on pages 106 and 181.)

[BR16] Stefan Boschert and Roland Rosen. Digital Twin—The Simulation Aspect. In Peter Hehenberger and David Bradley, editors, *Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and their Designers*, pages 59–74. Springer International Publishing, Cham, 2016. doi: 10.1007/978-3-319-32156-1_5. (Cited on page 30.)

[Bre09] Clay Breshears. *The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications*. "O'Reilly Media, Inc.", May 2009. Google-Books-ID: rU68SYVS7S8C. (Cited on page 33.)

[BRH90] Sanjoy K. Baruah, Louis E. Rosier, and Rodney R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Syst*, 2(4):301–324, November 1990. ISSN 1573-1383. doi: 10.1007/BF01995675. (Cited on pages xlii and 36.)

[BS02] Benoit Bréholée and Pierre Siron. Certi: Evolutions of the onera rti prototype. In *Fall Simulation Interoperability Workshop*, 2002. (Cited on pages 64 and 170.)

[BY] Y Bar-Yam. General Features of Complex Systems. *KNOWLEDGE MANAGEMENT*, page 10. (Cited on page 22.)

[CBP02] Christopher D. Carothers, David Bauer, and Shawn Pearce. ROSS: A high-performance, low-memory, modular Time Warp system. *Journal of Parallel and Distributed Computing*, 62(11):1648–1669, November 2002. ISSN 0743-7315. doi: 10.1016/S0743-7315(02)00004-7. (Cited on pages 49 and 51.)

[CFH+04] John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson, and Sanjoy Baruah. A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms. page 30, 2004. (Cited on page 71.)

[CGG+14] Francis Cottet, Emmanuel Grolleau, Sébastien Gérard, Jérôme Hugues, Yassine Ouhamou, and Sarah Tucci. *Systèmes temps réel embarqués : spécification, conception, implémentation et validation temporelle*. Dunod, Paris, 2014. (Cited on pages xl and 30.)

[Cha12] Jean-Baptiste Chaudron. *Architecture de simulation distribuée temps-réel*. Toulouse, ISAE, January 2012. (Cited on page 51.)

[CK06] François E. Cellier and Ernesto Kofman. *Continuous system simulation*. Springer, New York, 2006. (Cited on pages xxxvi, 21, and 25.)

[CM79] K. M. Chandy and J. Misra. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, September 1979. ISSN 0098-5589. doi: 10.1109/TSE.1979.230182. (Cited on pages 44, 50, and 51.)

[CM81] K. M. Chandy and J. Misra. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Commun. ACM*, 24(4):198–206, April 1981. ISSN 0001-0782. doi: 10.1145/358598.358613. (Cited on page 44.)

[COM15] Nicholson Collier, Jonathan Ozik, and Charles M. Macal. Large-Scale Agent-Based Modeling with Repast HPC: A Case Study in Parallelizing an Agent-Based Model. In Sascha Hunold, Alexandru Costan, Domingo Giménez, Alexandru Iosup, Laura Ricci, María Engracia Gómez Requena, Vittorio Scarano, Ana Lucia Varbanescu, Stephen L. Scott, Stefan Lankes, Josef Weidendorfer, and Michael Alexander, editors, *Euro-Par 2015: Parallel Processing Workshops*, Lecture Notes in Computer Science, pages 454–465. Springer International Publishing, 2015. (Cited on pages 47 and 49.)

[CR09] J. Casteres and T. Ramaherirariny. Aircraft integration real-time simulator modeling with AADL for architecture tradeoffs. In *Automation Test in Europe Conference Exhibition 2009 Design*, pages 346–351, April 2009. doi: 10.1109/DATE.2009.5090686. (Cited on page 48.)

[CS11] Eugene Chemeritskiy and Konstantin Savenkov. Towards a real-time simulation environment on the edge of current trends. In *Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering*, 2011. (Cited on page 48.)

[CSSA11] Jean-Baptiste Chaudron, David Saussié, Pierre Siron, and Martin Adelantado. Real-time aircraft simulation using HLA standard. pages 1–28, 2011. (Cited on page 51.)

[CSSA16] Jean-Baptiste Chaudron, David Saussié, Pierre Siron, and Martin Adelantado. How to solve ODEs in real-time HLA distributed simulation. pages 1–12, 2016. (Cited on pages 82 and 174.)

[DCCS17a] Henrick Deschamps, Gerlando Cappello, Janette Cardoso, and Pierre Siron. R-ROSACE: adding redundancy to the ROSACE case study. March 2017. (Cited on pages xiii, 152, and 189.)

[DCCS17b] Henrick Deschamps, Gerlando Cappello, Janette Cardoso, and Pierre Siron. 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*, Rome, Italy, October 2017. IEEE Computer Society. (Cited on pages xiii, 153, and 190.)

[DCCS18a] Henrick Deschamps, Gerlando Cappello, Janette Cardoso, and Pierre Siron. 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, February 2018. (Cited on pages xiii, 153, and 190.)

[DCCS18b] Henrick Deschamps, Gerlando Cappello, Janette Cardoso, and Pierre Siron. Implementation of a Cyber-Physical Systems simulation components allocation tool. In *Proceedings of the 2018 32nd European Simulation and Modelling Conference*, pages 112–119, 2018. (Cited on pages xiii, 154, and 191.)

[DEL] Sabine DELPECH. Université Toulouse III - Paul Sabatier - Accueil général FR. url: "`http://www.univ-tlse3.fr/`". (Cited on page 9.)

[Den75] Jack B. Dennis. Packet Communication Architecture,. Technical Report MAC-CSG-M-130, MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC, August 1975. (Cited on page 44.)

[Des16a] Henrick Deschamps. Redundant ROSACE, 2016. url: "`https://svn.onera.fr/schedmcore/branches/ROSACE_CaseStudy/redundant/`". (Cited on pages xiv, 123, 152, and 189.)

[Des16b] Henrick Deschamps. Scheduling of a cyber-physical system simulation, July 2016. url: "`https://cps2016.sciencesconf.org/resource/page/id/10`". (Cited on pages xiv, 152, and 189.)

[Des17] Henrick Deschamps. SEAplanes, 2017. url: "`https://openforge.isae.fr/svn/ordo-simu-dist/code/seaplanes`". (Cited on pages xiv, 152, and 190.)

[Des18a] Henrick Deschamps. Allocation tool, 2018. url: "`https://openforge.isae.fr/svn/ordo-simu-dist/code/allocation_tool`". (Cited on pages xiv, 153, and 190.)

[Des18b] Henrick Deschamps. Presentation of the Coincidence Problem in CPS Simulations: the R-ROSACE Case Study, April 2018. url: "`http://projects.laas.fr/IFSE/FAC/program/`". (Cited on pages xiii, 153, and 190.)

[Des18c] Henrick Deschamps. SEAplanes scope, 2018. url: "`https://openforge.isae.fr/projects/ordo-simu-dist/repository/show/code/seaplanes_scope`". (Cited on pages 152 and 190.)

[DFRS92] V. David, Ch Fraboul, J. Y. Rousselot, and Pierre Siron. Partitioning and mapping communication graphs on a modular reconfigurable parallel architecture. In *Parallel Processing: CONPAR 92—VAPP V*, pages 43–48. 1992. (Cited on pages 89 and 176.)

[DGST09] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, May 2009. ISSN 0167-739X. doi: 10.1016/j.future.2008.06.012. (Cited on page 47.)

[DL78] Sudarshan K. Dhall and C. L. Liu. On a Real-Time Scheduling Problem. *Operations Research*, 26(1,):127–140, 1978. (Cited on page 114.)

[DM14] Gabriele D'Angelo and Moreno Marzolla. New trends in parallel and distributed simulation: From many-cores to Cloud Computing. *Simulation Modelling Practice and Theory*, 49:320–335, December 2014. ISSN 1569-190X. doi: 10.1016/j.simpat.2014.06.007. (Cited on page 48.)

[DTCS17] Henrick Deschamps, Bastien Tauran, Janette Cardoso, and Pierre Siron. Distributing Cyber-Physical Systems Simulation: The Satellite Constellation Case. In *Proceedings of the 2017 5th International Federated and Fractionated Satellite Systems Workshop*, Toulouse, France, November 2017. (Cited on pages xiii and 10.)

[FBB+17] R. M. Fujimoto, R. Bagrodia, R. E. Bryant, K. M. Chandy, D. Jefferson, J. Misra, D. Nicol, and B. Unger. Parallel discrete event simulation: The making of a field. In *2017 Winter Simulation Conference (WSC)*, pages 262–291, December 2017. doi: 10.1109/WSC.2017.8247793. (Cited on page 43.)

[FCZ] E Fernandez-Cara and E Zuazua. Control Theory: History, Mathematical Achievements and Perspectives. page 62. (Cited on page 20.)

[FDMPR17] Massimo Ficco, Beniamino Di Martino, Roberto Pietrantuono, and Stefano Russo. Optimized task allocation on private cloud for hybrid simulation of large-scale critical systems. *Future Generation Computer Systems*, 74:104–118, September 2017. ISSN 0167-739X. doi: 10.1016/j.future.2016.01.022. (Cited on page 50.)

[FG13] Peter H. Feiler and David P. Gluch. *Model-based engineering with AADL: an introduction to the SAE Architecture Analysis & Design Language*. The SEI series in software engineering. Addison-Wesley, Upper Saddle River, N.J, 2013. OCLC: 820515268. (Cited on pages 104 and 180.)

[FH18] Roman Frigg and Stephan Hartmann. Models in Science. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2018 edition, 2018. (Cited on page 20.)

[FMS14] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, October 2014. Google-Books-ID: Ze60AwAAQBAJ. (Cited on pages 104 and 180.)

[Fuj90] Richard M. Fujimoto. Parallel Discrete Event Simulation. *Commun. ACM*, 33(10):30–53, October 1990. ISSN 0001-0782. doi: 10.1145/84537.84545. (Cited on pages 43, 46, and 51.)

[Fuj98] Richard M. Fujimoto. Time Management in The High Level Architecture. *SIMULATION*, 71(6):388–400, December 1998. ISSN 0037-5497. doi: 10.1177/003754979807100604. (Cited on page 50.)

[Fuj00] Richard M. Fujimoto. *Parallel and distribution simulation systems*. Wiley, New York, 2000. (Cited on pages xxxvi, xxxvii, 27, 43, and 51.)

[Fuj01] Richard M. Fujimoto. Parallel simulation: parallel and distributed simulation systems. In *Proceedings of the 33nd conference on Winter simulation*, pages 147–157. IEEE Computer Society, 2001. (Cited on page 43.)

[Fuj16] Richard M. Fujimoto. Research Challenges in Parallel and Distributed Simulation. *ACM Trans. Model. Comput. Simul.*, 26(4):22:1–22:29, May 2016. ISSN 1049-3301. doi: 10.1145/2866577. (Cited on page 43.)

[GGB15] S. Guan, R. E. De Grande, and A. Boukerche. Enabling HLA-based Simulations on the Cloud. In *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 112–119, October 2015. doi: 10.1109/DS-RT.2015.36. (Cited on page 48.)

[Gra03] Len Granowetter. RTI Interoperability Issues - API Standards, Wire Standards, and RTI Bridges. *Proceedings of the 2003 European Simulation Interoperability Workshop*, (03S-SIW):23, 2003. (Cited on page 49.)

[HE11] Dan Henriksson and Hilding Elmqvist. Cyber-Physical Systems Modeling and Simulation with Modelica. pages 502–509, June 2011. doi: 10.3384/ecp11063502. (Cited on pages 122 and 184.)

[Hea02] Steve Heath. *Embedded Systems Design*. Elsevier, October 2002. Google-Books-ID: BjNZXwH7HlkC. (Cited on page 21.)

[Hil18] Yannick Hildenbrand. ED-247 (VISTAS) Gateway for Hybrid Test Systems. SAE Technical Paper 2018-01-1949, SAE International, Warrendale, PA, October 2018. (Cited on page 48.)

[HJ12] Sanghyun Han and Hyun-Wook Jin. Kernel-level ARINC 653 Partitioning for Linux. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1632–1637, New York, NY, USA, 2012. ACM. doi: 10.1145/2245276.2232037. (Cited on pages 81 and 173.)

[HLV96] J. A. Hoogeveen, J. K. Lenstra, and B. Veltman. Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. *European Journal of Operational Research*, 89(1):172–175, February 1996. ISSN 0377-2217. doi: 10.1016/S0377-2217(96)90070-3. (Cited on pages xlii and 36.)

[HS92] Wolfgang A. Halang and Krzysztof M. Sacha. *Real-time systems: implementation of industrial computerised process automation*. World Scientific, Singapore ; River Edge, NJ, 1992. (Cited on pages xxxviii and 33.)

[HSVS14] M. Hanai, T. Suzumura, A. Ventresque, and K. Shudo. An Adaptive VM Provisioning Method for Large-Scale Agent-Based Traffic Simulations on the Cloud. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 130–137, December 2014. doi: 10.1109/CloudCom.2014.164. (Cited on page 48.)

[HZPK07] Jérôme Hugues, Béchir Zalila, Laurent Pautet, and Fabrice Kordon. Rapid Prototyping of Distributed Real-Time Embedded Systems Using the AADL and Ocarina. In *18th International Workshop on Rapid System Prototyping (RSP)*, pages 106–112, Porto Alegre, Brazil, May 2007. IEEE Computer Society. doi: 10.1109/RSP.2007.33. (Cited on pages 104 and 180.)

[II10] 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*. Institute of Electrical and Electronics Engineers, New York, 2010. OCLC: 682577410. (Cited on pages 64 and 170.)

[IMPQ17] M. Ianni, R. Marotta, A. Pellegrini, and F. Quaglia. A non-blocking global virtual time algorithm with logarithmic number of memory operations. In *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–8, October 2017. doi: 10.1109/DISTRA.2017.8167662. (Cited on page 50.)

[ISKM+16] Miloš Ivanović, Boban Stojanović, Ana Kaplarević-Mališić, Richard Gilbert, and Srboljub Mijailovich. Distributed multi-scale muscle simulation in a hybrid MPI–CUDA computational environment. *SIMULATION*, 92(1):19–31, January 2016. ISSN 0037-5497. doi: 10.1177/0037549715620299. (Cited on page 50.)

[Jef85] David R. Jefferson. Virtual Time. *ACM Trans. Program. Lang. Syst.*, 7(3):404–425, July 1985. ISSN 0164-0925. doi: 10.1145/3916.3988. (Cited on pages 50 and 51.)

[Kac15] Peter Kacsuk. Enabling Distributed Simulations Using Big Data and Clouds. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '15, pages 125–126, New York, NY, USA, 2015. ACM. doi: 10.1145/2769458.2769485. event-place: London, United Kingdom. (Cited on page 48.)

[Kag15] Henning Kagermann. Change Through Digitization—Value Creation in the Age of Industry 4.0. In Horst Albach, Heribert Meffert, Andreas Pinkwart, and Ralf Reichwald, editors, *Management of Permanent Change*, pages 23–45. Springer Fachmedien Wiesbaden, Wiesbaden, 2015. doi: 10.1007/978-3-658-05014-6_2. (Cited on page 10.)

[KAK+10] Roger Kneebone, Sonal Arora, Dominic King, Fernando Bello, Nick Sevdalis, Eva Kassab, Raj Aggarwal, Ara Darzi, and Debra Nestel. Distributed simulation – Accessible immersive training. *Medical Teacher*, 32(1):65–70, January 2010. ISSN 0142-159X. doi: 10.3109/01421590903419749. (Cited on page 46.)

[KJ98] Jefferson M. Koonce and William J. Bramble Jr. Personal Computer-Based Flight Training Devices. *The International Journal of Aviation Psychology*, 8(3):277–292, July 1998. ISSN 1050-8414. doi: 10.1207/s15327108ijap0803_7. (Cited on pages 10, 27, and 164.)

[KPMW80] J Kent Peacock, Eric Manning, and J. W Wong. Synchronization of distributed simulation using broadcast algorithms. *Computer Networks (1976)*, 4(1):3–10, February 1980. ISSN 0376-5075. doi: 10.1016/0376-5075(80)90024-0. (Cited on page 50.)

[KPWM79] J Kent Peacock, J. W Wong, and Eric G Manning. Distributed simulation using a network of processors. *Computer Networks (1976)*, 3(1):44–56, February 1979. ISSN 0376-5075. doi: 10.1016/0376-5075(79)90053-9. (Cited on page 44.)

[KSL99] F. Kuhns, D. C. Schmidt, and D. L. Levine. The design and performance of a real-time I/O subsystem. In *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium*, pages 154–163, June 1999. doi: 10.1109/RTTAS.1999.777670. (Cited on page 34.)

[KWTM11] S. Kite, C. Wood, S. J. E. Taylor, and N. Mustafee. Sakergrid: Simulation experimentation using grid enabled simulation software. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 2278–2288, December 2011. doi: 10.1109/WSC.2011.6147939. (Cited on page 48.)

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. (Cited on pages xxxiv, 23, 24, 50, 74, and 171.)

[Lan12] Christopher Landauer. Flight Systems are Cyber-Physical Systems, November 2012. (Cited on page 57.)

[Lee17] Alfred T. Lee. *Flight Simulation : Virtual Environments in Aviation*. Routledge, March 2017. doi: 10.4324/9781315255217. (Cited on page 27.)

[Lev09] John Levine. *Flex & Bison: Text Processing Tools*. "O'Reilly Media, Inc.", August 2009. Google-Books-ID: nYUkAAAAQBAJ. (Cited on pages 105 and 180.)

[LHM+07] P. Lendermann, M. U. Heinicke, L. F. McGinnis, C. McLean, S. Strassburger, and S. J. E. Taylor. Panel: distributed simulation in industry - a real-world necessity or ivory tower fancy? In *2007 Winter Simulation Conference*, pages 1053–1062, December 2007. doi: 10.1109/WSC.2007.4419704. (Cited on page 46.)

[LL73] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1):46–61, January 1973. ISSN 0004-5411. doi: 10.1145/321738.321743. (Cited on pages xxxix and 63.)

[LLCS14] Da Chuan Li, Qing Li, Nong Cheng, and Jing Yan Song. SOA-Cloud Computing Based Fast and Scalable Simulation Architecture for Advanced Flight Management System, 2014. url: "`https://www.scientific.net/AMR.1016.471`". (Cited on page 48.)

[LR09] E. W. Lynch and G. F. Riley. Hardware Supported Time Synchronization in Multi-core Architectures. In *2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pages 88–94, June 2009. doi: 10.1109/PADS.2009.19. (Cited on page 50.)

[Mal15] Frédéric Mallet. MARTE/CCSL for Modeling Cyber-Physical Systems. In *MARTE/CCSL for Modeling Cyber-Physical Systems*. Springer Fachmedien Wiesbaden, June 2015. (Cited on pages 104 and 180.)

[Mas06] A. J. Masys. Verification, Validation and Accreditation: An HLA FEDEP Overlay. In *2006 Canadian Conference on Electrical and Computer Engineering*, pages 1850–1853, May 2006. doi: 10.1109/CCECE.2006.277782. (Cited on page 49.)

[McG02] Ian McGregor. Equipment Interface: The Relationship Between Simulation and Emulation. In *Proceedings of the 34th Conference on Winter Simulation: Exploring New Frontiers*, WSC '02, pages 1683–1688, San Diego, California, 2002. Winter Simulation Conference. (Cited on page 25.)

[MD17] M. Marzolla and G. D'Angelo. Parallel sort-based matching for data distribution management on shared-memory multiprocessors. In *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–8, October 2017. doi: 10.1109/DISTRA.2017.8167660. (Cited on page 49.)

[MHSS05] E. M. Moreira, R. Helena, C. Santana, and M. J. Santana. Using consistent global checkpoints to synchronize processes in distributed simulation. In *Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 43–50, October 2005. doi: 10.1109/DISTRA.2005.40. (Cited on page 51.)

[Mis86] Jayadev Misra. Distributed Discrete-event Simulation. *ACM Comput. Surv.*, 18(1):39–65, March 1986. ISSN 0360-0300. doi: 10.1145/6462.6485. (Cited on page 45.)

[MS] Gilberto Mosqueda and Bozidar Stojadinovic. Implementation and Accuracy of Continuous Hybrid Simulation with Geographically Distributed Substructures. page 184. (Cited on page 50.)

[NAT13] A. Nouman, A. Anagnostou, and S. J. E. Taylor. Developing a Distributed Agent-Based and DES Simulation Using poRTIco and Repast. In *2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*, pages 97–104, October 2013. doi: 10.1109/DS-RT.2013.18. (Cited on page 47.)

[Nel16] B. L. Nelson. 'Some tactical problems in digital simulation' for the next 10 years. *Journal of Simulation*, 10(1):2–11, February 2016. ISSN 1747-7778. doi: 10.1057/jos.2015.22. (Cited on page 48.)

[noaa] Airbus Home. url: "`https://www.airbus.com/`". (Cited on page 7.)

[noab] Association Nationale Recherche Technologie. url: "`http://www.anrt.asso.fr/fr`". (Cited on page 6.)

[noac] Commercial Aircraft. url: "`https://www.airbus.com/aircraft.html`". (Cited on page 8.)

[noad] Ecole doctorale MITT Mathématiques Informatique Télécommunications de Toulous - ED 475 Toulouse. url: "`http://www.edmitt.ups-tlse.fr/`". (Cited on page 9.)

[noae] Français - ISAE-SUPAERO. url: "`https://www.isae-supaero.fr/fr/`". (Cited on page 9.)

[noa15] Cifre, November 2015. url: "`http://www.anrt.asso.fr/fr/cifre-7843`". (Cited on page 6.)

[noa19] OMNeT++ Discrete Event Simulator. Contribute to omnetpp/omnetpp development by creating an account on GitHub, May 2019. url: "`https://github.com/omnetpp/omnetpp`". original-date: 2018-12-03T16:18:31Z. (Cited on page 104.)

[NRS09] Eric Noulard, Jean-Yves Rousselot, and Pierre Siron. CERTI, an Open Source RTI, why and how. pages 1–11, 2009. (Cited on pages 65 and 170.)

[Pin16] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, February 2016. (Cited on pages xl and 31.)

[PS15] Mayur Pandey and Suyog Sarda. *LLVM Cookbook*. Packt Publishing Ltd, May 2015. Google-Books-ID: WqS_CQAAQBAJ. (Cited on page 105.)

[PSG+14] Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre 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)*, pages 309–318. IEEE, 2014. (Cited on pages 122 and 184.)

[PSGk14] M. Prasad, A. Singh, M. Gangadhar, and L. Sobhan kumar. Real Time Simulation system for aerospace interceptors. In *International Conference on Computing and Communication Technologies*, pages 1–5, December 2014. doi: 10.1109/ICCCT2.2014.7066714. (Cited on pages 48 and 51.)

[RCV12] M. Rak, A. Cuomo, and U. Villano. mJADES: Concurrent Simulation in the Cloud. In *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 853–860, July 2012. doi: 10.1109/CISIS.2012.134. (Cited on page 48.)

[Rey82] Paul F. Reynolds, Jr. A Shared Resource Algorithm for Distributed Simulation. In *Proceedings of the 9th Annual Symposium on Computer Architecture*, ISCA '82, pages 259–266, Los Alamitos, CA, USA, 1982. IEEE Computer Society Press. event-place: Austin, Texas, USA. (Cited on page 45.)

[RGO] Arnon Rotem-Gal-Oz. Fallacies of Distributed Computing Explained. (Cited on page 49.)

[Rit78] Dennis M. Ritchie. *The C Programming Language*. 1978. (Cited on pages 127 and 185.)

[RJB04] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004. (Cited on pages 104 and 180.)

[RNP10] Stuart Jonathan Russell, Peter Norvig, and Fabrice Popineau. *Intelligence artificielle*. Pearson education, Paris, 2010. OCLC: 708384789. (Cited on pages xxxviii and 37.)

[RW89] R. Righter and J. C. Walrand. Distributed simulation of discrete event systems. *Proceedings of the IEEE*, 77(1):99–113, January 1989. ISSN 0018-9219. doi: 10.1109/5.21073. (Cited on page 46.)

[SKK+12] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and S. Wang. Toward a Science of Cyber–Physical System Integration. *Proceedings of the IEEE*, 100(1):29–44, January 2012. ISSN 0018-9219. doi: 10.1109/JPROC.2011.2161529. (Cited on pages 12 and 164.)

[SMM06] Stojadinovic Bozidar, Mosqueda Gilberto, and Mahin Stephen A. Event-Driven Control System for Geographically Distributed Hybrid Simulation. *Journal of Structural Engineering*, 132(1):68–77, January 2006. doi: 10.1061/(ASCE)0733-9445(2006)132:1(68). (Cited on page 50.)

[SO10] Anthony Steed and Manuel Fradinho Oliveira. Chapter 11 - Latency and consistency. In Anthony Steed and Manuel Fradinho Oliveira, editors, *Networked Graphics*, pages 355–392. Morgan Kaufmann, Boston, January 2010. doi: 10.1016/B978-0-12-374423-4.00011-2. (Cited on page 49.)

[SS89] Torsten Söderström and Petre Stoica. *System identification*. Prentice Hall International series in systems and control engineering. Prentice Hall, New York, 1989. (Cited on page 20.)

[SSF08] S. Strassburger, T. Schulze, and R. Fujimoto. Future trends in distributed simulation and distributed virtual environments: Results of a peer study. In *2008 Winter Simulation Conference*, pages 777–785, December 2008. doi: 10.1109/WSC.2008.4736140. (Cited on page 46.)

[Tan87] Andrew S. Tanenbaum. *Operating systems: design and implementation*. Prentice-Hall software series. Prentice-Hall, Englewood Cliffs, NJ, 1987. OCLC: 14001792. (Cited on pages 71 and 171.)

[Tay19] Simon J. E. Taylor. Distributed simulation: state-of-the-art and potential for operational research. *European Journal of Operational Research*, 273(1):1–19, February 2019. ISSN 0377-2217. doi: 10.1016/j.ejor.2018.04.032. (Cited on page 46.)

[TB07] Dr Andreas Tolk and James L Boulet. Lessons Learned on NATO Experiments on C2/M&S Interoperability. page 9, 2007. (Cited on page 49.)

[TCQ+18] Fei Tao, Jiangfeng Cheng, Qinglin Qi, Meng Zhang, He Zhang, and Fangyuan Sui. Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, 94(9):3563–3576, February 2018. ISSN 1433-3015. doi: 10.1007/s00170-017-0233-1. (Cited on page 48.)

[TGP07] J. L. Tripp, M. B. Gokhale, and K. D. Peterson. Trident: From High-Level Language to Hardware Circuitry. *Computer*, 40(3):28–37, March 2007. ISSN 0018-9162. doi: 10.1109/MC.2007.107. (Cited on page 105.)

[TM03] Andreas Tolk and James A. Muguira. The levels of conceptual interoperability model. In *Proceedings of the 2003 fall simulation interoperability workshop*, volume 7, pages 1–11. Citeseer, 2003. (Cited on pages 46, 49, and 83.)

[Tol] Andreas Tolk. The Next Generation of Modeling & Simulation: Integrating Big Data and Deep Learning. page 9. (Cited on page 48.)

[TPQ+17] Tommaso Tocci, Alessandro Pellegrini, Francesco Quaglia, Josep Casanovas-García, and Toyotaro Suzumura. ORCHESTRA: An Asynchronous Wait-free Distributed GVT Algorithm. In *Proceedings of the 21st International Symposium on Distributed Simulation and Real Time Applications*, DS-RT '17, pages 51–58, Piscataway, NJ, USA, 2017. IEEE Press. event-place: Rome, Italy. (Cited on page 51.)

[TRC+14] S. J. E. Taylor, N. Revagar, J. Chambers, M. Yero, A. Anagnostou, A. Nouman, N. R. Chaudhry, and P. R. Elfrey. Simulation Exploration Experience: A Distributed

Hybrid Simulation of a Lunar Mining Operation. In *2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*, pages 107–112, October 2014. doi: 10.1109/DS-RT.2014.21. (Cited on page 49.)

[TSS+] Matsu Thornton, Holm Smidt, Volker Schwarzer, Mahdi Motalleb, and Reza Ghorbani. Internet-of-Things Hardware-in-the-Loop Simulation Testbed for Demand Response Ancillary Services. page 5. (Cited on page 48.)

[vdBSC17] Tom van den Berg, Barry Siegel, and Anthony Cramp. Containerization of high level architecture-based simulations: A case study. *The Journal of Defense Modeling and Simulation*, 14(2):115–138, April 2017. ISSN 1548-5129. doi: 10.1177/1548512916662365. (Cited on page 49.)

[VH08] András Varga and Rudolf Hornig. An Overview of the OMNeT++ Simulation Environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). event-place: Marseille, France. (Cited on page 104.)

[VJ14] M. Venu and I. Joe. Improving performance of optimistic simulation for distributed simulation system using speculative computation. In *2014 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 428–432, October 2014. doi: 10.1109/ICTC.2014.6983173. (Cited on page 51.)

[VMB12] K. Vanmechelen, S. De Munck, and J. Broeckhove. Conservative Distributed Discrete Event Simulation on Amazon EC2. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 853–860, May 2012. doi: 10.1109/CCGrid.2012.73. (Cited on page 47.)

[WG17] Philipp Wehner and Diana Göhringer. Internet of Things Simulation Using OMNeT++ and Hardware in the Loop. In Georgios Keramidas, Nikolaos Voros, and Michael Hübner, editors, *Components and Services for IoT Platforms: Paving the Way for IoT Standards*, pages 77–87. Springer International Publishing, Cham, 2017. doi: 10.1007/978-3-319-42304-3_4. (Cited on page 48.)

[WLC07] Jeff Van West and Kevin Lane-Cummings. *Microsoft Flight Simulator X For Pilots: Real World Training*. John Wiley & Sons, June 2007. Google-Books-ID: RT9v6Bhl420C. (Cited on pages 10 and 164.)

[Wol06] Pierre Wolper. *Introduction à la calculabilité: cours et exercices corrigés*. Dunod, Paris, 2006. OCLC: 634642848. (Cited on pages xxxix and 37.)

[Wol09] W. Wolf. Cyber-physical Systems. *Computer*, 42(3):88–89, 2009. ISSN 0018-9162. (Cited on page 19.)