THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par **l'Institut Supérieur de l'Aéronautique et de l'Espace**
Spécialité : Sûreté de logiciel et calcul de haute performance

---

Présentée et soutenue par **Jean-Baptiste CHAUDRON**
le **25 janvier 2012**

Architecture de simulation distribuée temps-réel

JURY

M. Daniel Hagimont, président
M. Martin Adelantado, co-directeur de thèse
Mme Claudia Frydman, rapporteur
M. Jean-Louis Igarza
M. Daniel Simon, rapporteur
M. Pierre Siron, directeur de thèse

École doctorale : **Mathématiques, informatique et télécommunications de Toulouse**

Unité de recherche : **Équipe d'accueil ISAE-ONERA MOIS**

Directeur de thèse : **M. Pierre Siron**
Co-directeur de thèse : **M. Martin Adelantado**

If I can't dance, I won't be a part of your revolution.
Emma Goldman

Je dédie ce travail de thèse à la mémoire de mon grand-père Dado,
j'aurais tellement aimé qu'il soit présent.

Je dédie aussi ce travail de thèse à la mémoire de mon cousin Nicolas,
je pense beaucoup à lui et à nos moments de jeunesse partagés.

Remerciements

Une thèse est un long chemin demandant beaucoup de travail, un brin de folie, quelques litres de cafés, la lecture de centaines d'articles, de nombreuses discussions avec des gens très forts et des heures de réflexion. La rédaction des remerciements constitue le point de conclusion de cette combinaison complexe de petites choses ; c'est à la fois heureux parce que l'on a abouti un projet conséquent mais aussi triste car une étape de notre vie s'achève et il faut tourner une page. Paradoxalement, cette thèse sur une architecture de simulation temps réel a aussi débuté pour moi par un évènement qui m'a montré que le terme *temps réel* ne s'applique pas uniquement dans le domaine des systèmes embarqués mais aussi dans la vie de tous les jours. Mes remerciements seront donc longs car je souhaiterais remercier et citer toutes les personnes qui ont participé pendant ces années de travail à mon évolution personnelle et professionnelle. Par ces quelques mots, je voudrais leur témoigner ma gratitude.

Tout d'abord, pour mener à bien un travail de thèse, il faut des personnes qui vous guident et vous dirigent : Pierre Siron et Martin Adelantado ont été mes encadrants et sont devenus mes amis. Je remercie Pierre pour sa rigueur scientifique et son soucis du détail qui ont permis de donner plus de clarté au contenu de ces travaux. Je lui suis reconnaissant d'avoir accepté de me donner ma chance sur cette problématique de thèse alors que je venais tout juste de découvrir ce domaine de recherche. Je remercie Martin pour sa simplicité et sa vision toujours claire des problèmes qui m'ont permis de relativiser durant les différentes étapes de ce doctorat. J'ai aussi beaucoup apprécié nos élucubrations scientifiques allant de la théorie des cordes jusqu'aux mondes parallèles en passant par l'incontournable théorie de la mécanique quantique[1].

Je tiens à remercier les membres du jury pour avoir accepté de consacrer un peu de leur temps pour évaluer mon travail et participer à ma soutenance : Daniel Hagimont qui a présidé cette commission d'examen, Daniel Simon et Claudia Frydman qui ont acceptés de relire mon manuscrit et d'y apporter quelques remarques constructives et enfin Jean-Louis Igarza qui a également étudié ce document.

Mon travail de thèse s'est déroulé, dans les meilleures conditions, au sein de deux laboratoires : le département du DTIM de l'ONERA ainsi que le département du DMIA de l'ISAE.

Du coté du DTIM, je voudrais tout d'abord remercier Eric Noulard pour avoir toujours été disponible pour répondre à mes (trop ?) nombreuses questions. C'est un scientifique déconcertant autant par ses connaissances que par sa faculté de compréhension. Il a énormément contribué à ce travail de thèse et je le remercie pour son aide précieuse. Je tiens aussi à saluer Jean-Michel Mathé avec qui j'ai pu échanger sur de nombreux sujets dont des problématiques asymétriques et napoléoniennes (tant d'*histwar* à raconter !) et Jean-Loup Bussenot pour ses coups de pouces

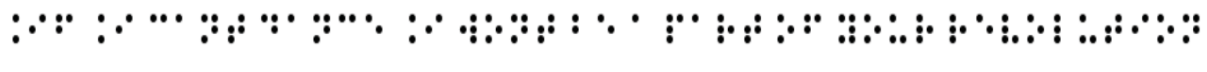
[1] *Le futur est devant nous, Marti(n) !*

techniques, ses prêts matériels et pour avoir partagé avec moi quelques discussions intéressantes sur des sujets divers et variés. Je remercie aussi tous les autres membres du DTIM pour faire de ce département un lieu où il fait "bon" de travailler : Bernard, Josette, Jean-Yves, Frederic, Bruno, Claire (P) (merci pour son aide sur la théorie de l'ordonnancement), Claire (S), Olivier (merci pour les discussions sportives), Rémi, Julien, Guy, Virginie, Christiane et les autres ainsi que tous les doctorants (ou maintenant docteurs) Mikel, Joël, Cedric, Sophie, Julien, Stéphanie, Pierre, Vincent, Florian, Dimitri, Rémy, Matthieu, Christophe, Maria, Charles, Adrien, William,...Bonne continuation à tous !

Du coté du DMIA, je voudrais remercier Régine Leconte pour sa bonne humeur quotidienne, pour ses scripts shells *fulgurants*[2] mais surtout pour ses nombreux coups de pouce afin de mener à bien les quelques expérimentations décrites dans ce document. Je voudrais aussi remercier David Saussié avec qui j'ai énormément apprécié de travailler. Notre rencontre s'est fait au bon moment et, sans sa contribution, ce travail de thèse n'aurait pas été complété par le cas d'étude sur le simulateur de vol d'avion. Notre collaboration a été très agréable et extrêmement enrichissante donc...Merci David ! Je salue aussi tous les membres du DMIA permanents, doctorants et stagiaires qui permettent d'y travailler dans une bonne ambiance.

Avant de devenir doctorant, j'ai été ingénieur au département du DPRS, toujours au centre ONERA de Toulouse. Je voudrais tout d'abord remercier Armand Oyzel qui m'a un peu mis le pied à l'étrier alors que je sortais de mes études. J'ai beaucoup appris en travaillant avec lui et il a été un co-bureau vraiment très agréable. Par la suite, nous avons gardé nos habitudes pour les cafés du matin et de l'après-midi accompagnés de Franck Morel. Je les remercie donc tous les deux pour ces quelques minutes de détente quotidiennes parfois humoristiques, engagées, techniques ou politiques mais surtout toujours extrêmement agréables. Je salue aussi tous mes anciens collègues Luis, Thomas (R), Sébastien, Thomas (C), Kioumars, Jean, Jean-Louis, Muriel, Claire, Bruno et Peter (merci pour nos discussions préliminaires sur le simulateur de vol). Je pense aussi aux personnes que j'ai pu rencontrer durant mon parcours : Marc Boyer, Nicolas Maille, Gregory Claude, Frederic Dehais, Claude Barrouil, Claudette Cayrol, Leïla Amgoud ainsi que David Guéron (maintenant docteur !) que j'apprécie beaucoup.

Enfin, pour soutenir une thèse, il faut une école doctorale et des personnes qui s'occupent de votre dossier, je voudrais donc remercier Monique Péron, Annie Carles-Bailhé, Maryse Herbillon et Martine Labruyère pour m'avoir permis de mener à bien toutes les procédures administratives nécessaires à ce doctorat.



Plus personnellement maintenant, je passe ici une dédicace pour tous mes *fwouewes de l'enfer*, un peu comme ma seconde famille, du coin de la rue devant chez Jess jusqu'au Canada en passant par les virées à la montagne. Entre les folies de l'adolescence jusqu'aux soirées arrosées et les moments de réflexion de nos vies d'adulte où on essaye de construire un monde meilleur, ils ont toujours été présent. Merci à toute la bande de sales fous d'ici et d'ailleurs : Mika, Océane, Steph et la petite Nina sky, Farid, Smail, Mounir, Nico, Nono, Jaja, Bibi, Jess, Kangoo, Chinois, Sala...PURA VIDA à tous et merci pour tout du fond du cœur.

Je remercie mon pote Max (dit De Kellogs) pour ces rôles multiples qu'il a pu jouer dans ma vie : colocataire, confident, ami, co-pilier de comptoir, ...Comme un vieux couple, on se brouille de temps en temps mais qu'il trouve en ces quelques lignes un témoignage de ma profonde amitié.

[2] *des fois, peut être, trop efficaces...*

J'espère que tu trouveras ta voie, l'artiste ! Je tiens à saluer ici aussi sa famille : sa promise Leti[3], sa maman Angèle, son papa Gérard, son frère Jo et sa petite sœur Amandine. Je passe une dédicace à Smain et un spécial PULL UP à Rasta[4].

Je tiens à saluer mon ami maintenant américain P'tit Alex qui me manque beaucoup depuis qu'il est parti vers de nouvelles aventures. Comme on dit, loin des yeux, près du cœur ! Je te souhaite le meilleur mon pote et je te remercie pour tout. Je salue aussi le bout-en-train Nicolas[5], son frangin Loïc et Flore (avec leur petit Diego), Jérôme, Annie, Hervé et Edwige, Leti ainsi que Jean-Mi et Daria.

Je remercie mon ami d'enfance Robin et sa chérie Julie (sans oublier leur petit bout Loucas) pour ces très agréables moments partagés autour d'un bon repas ou d'un petit apéro. Je salue mon pote Xavier et sa petite famille Nathalie, Pierre, Zia et la toute petite Céline ; mon pote Keyven, ses parents et ses frangins ainsi que son équipe : David, Tut et Pierrick. Je continu en passant une dédicace à Tonton Jay et la jolie Paula. Je salue aussi Manu, Mag et le petit Kylian, Baby Jo et Mylène. J'embrasse ma jolie mamita pleine de folie Carine ainsi que Sasa et Aurélie pour les repas du villa Da Vinci et les quelques soirées toulousaines animées. J'embrasse aussi Guy pour ses œuvres encrées et permanentes[6], sa femme Alcine, Manu, Else et Marlène. Enfin, je passe une dédicace à mon pote Mathieu (dit Matoz) poète perdu et amateur de gros son *West Coast* ainsi qu'à sa famille.

Je voudrais saluer mon club de boxe : le TMB (Toulouse Multi Boxe). Je remercie mon entraineur Toto pour ce temps qu'il a accordé à ma préparation pour les différents combats, sa femme Kafia et le petit Gaetano, Christian pour avoir toujours été dans mon coin et les boxeurs Hirach, Aless[7], Karim, Medhi, Alexis, Mounir, Terry, Sébastien, Romain, Flora, Karina, Céline, Aline, Peggy et les autres pour les heures partagées ensemble à perdre des litres de sueur et les quelques coups échangés pour les préparations des échéances de chacun.

Mes pensées vont aussi aux personnes qui sont parties pendant ces années de travail, je tiens à honorer : la mémoire de John avec qui j'ai eu la chance de partager quelques belles soirées toulousaines ; la mémoire de Julien (dit Ju-Ju) avec qui j'ai pu échanger sur une passion commune pour les plantes cactées ainsi que des discussions fort intéressantes sur la programmation[8] et le réseau lors de quelques repas et de quelques bringues ; la mémoire de Fanya qui m'a toujours témoigné beaucoup d'affection.

Malgré ces quelques amis fidèles et dévoués, une thèse ne se déroule pas bien sans un certain équilibre familial. Je voudrais remercier ma famille pour tous ces bons moments partagés, je suis très chanceux d'être arrivé dans une famille recomposée mais néanmoins soudée où les différences de chacun sont une force.

Tout d'abord, je voudrais dédier cette thèse à ma maman Catherine. Je ne serais jamais arrivé à ce stade sans son soutien à tous les niveaux. Elle a été un pilier sur lequel j'ai toujours pu me reposer ; qu'elle trouve ici l'expression de mon amour inconditionnel. A mes magnifiques petites sœurs (plus si petites avec le temps...) : Alexandra, la plus grande, la plus dévouée ; Caroline, la

[3] *Le terme "promise" a été choisi scientifiquement, paraissant le plus adapté pour décrire l'état quantique permanent de ce domaine de recherche.*

[4] *Nous garderons ce surnom même si son importance capillaire a très largement diminué ces derniers temps !*

[5] *I'm waiting for your "One Man Show" my friend !*

[6] *Pour plus de détails...voir Dom'Tattoo Studio !*

[7] *Merci pour tout tonton, tu as été là pour moi quand j'en avais besoin.*

[8] *Encore un linuxien !*

moyenne, la plus entière ; Marie-Lou (dit Mi-Lou), la plus petite, la plus douce ; je vous remercie pour m'avoir apporté chacune à votre façon tout votre amour et votre soutien, je vous souhaite tout le bonheur du monde, et que la vie vous apporte le meilleur, je suis très heureux et fier d'être votre grand-frère. Je salue aussi Olivier (dit B.P) qui est un b...-p... très cool. J'embrasse mon parrain Vincent, sa femme Sabine, ma jolie et adorable filleule Marine ainsi que Arthur (plus si petit lui aussi, décidément...). J'embrasse aussi mon oncle Tonton François, ma tante Valérie, ma cousine Charlotte et mon cousin Pierre (plus du tout petit maintenant...) avec lequel j'apprécie de partager des moments simples de bricolage ou de discussion de mecs. Enfin, je pense à ma grand-mère Manou et à tout ce qu'elle représente pour moi, je la remercie pour l'amour et l'affection qu'elle dégage et pour tous ses petits plats si bons[9].

Je remercie mon papa Laurent qui m'a transmis sa passion pour la recherche. Pendant mon époque rebelle[10], je disais que je ne ferais jamais comme lui ; et j'ai finalement suivi cette voie. Je ne sais pas si j'ai trouvé *la voie* mais j'espère que lui trouvera dans ces quelques mots un témoignage de mon amour de fils. Je pense aussi à sa maman Suzy (ma grand-mère donc[11]) qui a tant fait pour nous ainsi qu'à ma tante et marraine Nadine dont la gentillesse est proportionnelle à son stress[12] ; je voudrais leur témoigner tout mon amour. J'embrasse aussi ma toute petite sœur Maya qui est bien jolie ainsi que Sabine et Théo. Enfin, je voudrais passer une spéciale dédicace à mon cousin Tanguy que je remercie pour m'avoir tellement appris sur le jongle, les cactus et les plantes grasses ; c'est une personne que j'adore et que j'admire beaucoup.

Je voudrais aussi remercier ma belle-famille : Korine pour m'avoir toujours soutenu et être restée une belle-maman adorable, tolérante et à l'écoute ; Rémi qui est un homme bien et qui devait être au premier rang le jour où ils ont distribué la gentillesse ; Carla que j'ai connue toute petite et pour qui j'ai beaucoup d'affection ; Eric qui est un homme[13] que je respecte beaucoup ainsi que Sandrine, Anthony, Chloé, Florence, Aurélie, Ludo et les autres. J'embrasse Patrick, Océane ainsi que le petit Lenny et je les remercie pour tous ces bons moments passés ensemble que ce soit sur Toulouse ou sur Perpignan. Je pense aussi à Fabrice qui a initié ma conversion à la nourriture saine, Nathalie, Babeth, Jérôme ainsi que sa femme et ses enfants.

Pour finir, je voudrais dédier tout ce travail à mon doudou Alexandra qui a partagé ma vie ces dernières années et qui a donné tellement plus de sens à tout cela. C'est la personne qui connait le mieux toutes les facettes de ma personnalité, je la remercie pour son soutien, son amour, sa tolérance et sa patience. Il est difficile d'exprimer tout ce que je ressens pour elle en si peu de mots et, malgré notre relation parfois compliquée, j'aimerais juste lui dire que dans la folie de ma vie d'artiste, elle m'a offert huit années de bonheur.

Voilà, ces remerciements sont terminés, j'ai essayé de faire une liste exhaustive de toutes les personnes qui me sont chères. J'espère que je n'ai oublié personne mais, si c'est le cas, qu'ils ne s'arrêtent pas sur cet oubli et qu'ils sachent que je suis reconnaissant d'avoir eu la chance de partager avec eux quelques moments de vie si anodins en apparence.

Merci à tous.

[9] *A ce sujet d'ailleurs, je mangerais un bon gateau à la crème de marrons !*
[10] *Est elle vraiment terminée ?*
[11] *Quel esprit de déduction !*
[12] *C'est dire comme elle est gentille...*
[13] *Même un bonhomme !*

Résumé - Abstract

Résumé

Ce travail de thèse s'inscrit dans le projet plus global PRISE (Plate-forme de Recherche pour l'Ingénierie des Systèmes Embarqués) dont l'objectif principal est le développement d'une plate-forme d'exécution pour les logiciels embarqués. De tels logiciels sont dits critiques et ils sont, par conséquent, soumis à des règles de conception spécifiques. Notamment, ces logiciels doivent répondre à des contraintes de temps réel et ainsi garantir des comportements temporels prédictifs afin de toujours donner des résultats justes avec le respect d'échéances temporelles.

L'objectif de cette thèse est d'évaluer l'utilisation des techniques de la simulation distribuée (et particulièrement de la norme HLA) pour répondre aux besoins de simulation hybride et temps réel de la plate-forme. Afin de respecter ces contraintes et garantir la prédictibilité temporelle d'une simulation distribuée, il faut avoir une vision complète de l'ensemble du problème et notamment des différents niveaux d'actions : applicatif, intergiciel, logiciel, matériel et aussi formel pour la validation du comportement temporel.

Cette thèse se base sur la RTI (Run Time Infrastructure, intergiciel HLA) de l'ONERA : le CERTI et propose une démarche méthodologique adaptée à ces différents niveaux d'actions. Des cas d'étude, notamment un simulateur du vol d'un avion, ont été spécifiés, implémentés et expérimentés sur la plate-forme PRISE.

Mots clés : PRISE, simulation distribuée, temps réel, HLA, CERTI.

Abstract

This work takes place in the global project PRISE (Plate-forme de Recherche pour l'Ingénierie des Systèmes Embarqués) in which the focus is to develop an execution platform for embedded software. Embedded software are said criticals and, therefore, are subject to specific design rules. Particularly, these software must meet real time constraints and thus ensure a temporal predictive behaviour in order to always give accurate results with respect to corresponding timing deadlines.

The main objective of this thesis is to study the use of distributed simulation techniques (and specifically the HLA standard) to meet the real-time and hybrid simulation needs of the PRISE platform. To comply with these real-time constraints and ensure the predictability of a distributed simulation, we must have a complete view of the whole problem and in particular the different levels of action : application, middleware, software, hardware and also a formal level for validation of the timing behaviour.

This work is based on the RTI (Run Time Infrastructure, HLA middleware) from ONERA laboratory called : the CERTI and proposes a methodological approach adapted to take into account these different levels of action. Some case studies, including a flight simulator of an aircraft, have been specified, implemented and tested on the PRISE platform.

Keywords : PRISE, distributed simulation, real time, HLA, CERTI.

Table des matières

Listes des figures

Liste des tableaux

Listes des accronymes

AADL Architecture Analysis and Design Language

ABE ALSP Broadcast Emulator

ACE Adaptative textcolorblueCommunication textcolorblueEnvironment

ACM ALSP Common Module

ADEOS Adaptative Domain Environment for Operating System

ADF Aéroport Du Futur

ADIRU Air Data Inertial Reference Unit

AFDX Avionics Full DupleX switched ethernet

AIS ALSP Infrastructure Software

ALSP Aggregate Level Simulation Protocol

API Application Programming Interface

ARTISST ARTISST is a Real-Time System Simulation Tool

ATM Asynchronous Transfer Mode

B&B Branch and Bound

BF Best Fit

CAN Controller Area Network

CAN-DM Controller Area Network Deadline Monotonic

CDO Common Development Organisation

CFM Compact-Fit Moving

CFNM Compact-Fit Non-Moving

CMB Chandy-Misra-Bryant

CNES Centre National d'Etudes Spatiales

CNRS Centre National de la Recherche Scientifique

CORBA Common Object Request Broker Architecture

CRC Central RTI Component

CSMA/CD Carrier Sense Multiple Access/Collision Detection

CSMA/DCR Carrier Sense Multiple Access/Deterministic Collision Resolution

DARPA Defence Advanced Research Project Agency

DCPS Data Centric Publish Subscribe

DDS Data Distribution Services

DiffServ Differentiated Services

DIS Distributed Interactive Simulation

DLRL Data Local Reconstruction Layer

DM Deadline Monotonic

DMIA Département de Mathématiques, Informatique Automatique

DoD Department Of Defense

DS Deferable Server

DTIM Département Traitement de l'Information et Modélisation

EDD Earliest Due Date

EDF Earliest Deadline First

ENSEEIHT Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique, d'Hydraulique et des Télécommunications

FDD FOM Document Data

FDDI Fiber Distributed Data Interconnect

FDK Federation Developers Kit

FDMA Frequency Division Multiple Access

FED Federation Execution Model

FEDEP FEderation Development and Execution Process

FF First Fit

FFDU First Fit Decreasing Utilization

FIFO First In First Out

FIP Factory Instrumentation Protocol

FLEX-MAT FLEXible Middleware And Transport

FMLP Flexible Multiprocessor Locking Protocol

FOM Federation Object Model

GALT Greatest Avalaible Logical Time

GPS Global Positionning System

gRMA graphical tool forRate Monotonic Analysis

HI-BP Hybrid Improvement heuristic for Bin-Packing

IEEE Institute of Electrical and Electronics Engineers

IESTA Infrastructure d'Evaluation et de Simulation du Transport Aérien

IMU Inertial Measurement Unit

INRIA Institut National de Recherche en Informatique et en Automatique

IntServ Integrated Services

IP Internet Protocol

IRU Inertial Reference Unit

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

ISO International Standard Organization

JTC Joint Training Confédération

JVM Java Virtual Machine

LAN Local Area Network

LBTS Lower Bound Time Stamp

LFU Least Frequently Used

LISYC Laboratoire d'Informatique des SYstèmes Complexes

LLF Least Laxity First

LRC Local RTI Component

LRU Least Recently Used

MAC Medium Access Control

MLLF Modified LLF

MOM Management Object Model

MPCP Multiprocessor Priority Ceiling Protocol

M&S Modeling and Simulation

MUF Maximum Urgency First

NF Next Fit

NLR Nationaal Lucht en Ruimtevaartlaboratorium

NP-EDF Non-Preemptive Earliest Deadline First

NP-RM Non-Preemptive Rate Monotonic

NTP Network Time Protocol

ODE Ordinary Differential Equation

ODP-RM Open Distributed Processing Reference Model

OMG Object Management Group

OMT Object Model Template

ONERA Office National d'Etudes et de Recherches Aérospatiales

ORB Object Request Broker

OSE Operating System Embedded

OSEK Offene Systeme und deren schnittstellen für die Elektronik im Kraftfahrzeug

OSI Open Systems Interconnection

OTAN Organisation du Traité de l'Atlantique Nord

PCP Priority Ceiling Protocol

PDA Processor Demand Analysis

PDU Protocol Data Unit

PERTS Prototyping Environment for Real-Time Systems

PES Priority Exchange Server

PFD Primary Fligh Display

PIP Priority Inheritance Protocol

POSIX Portable Operating System Interface for uniX

PRISE Plate-forme de Recherche et d'Ingéniérie sur les Systèmes Embarqués

PS Polling Server

PTP Precision Time Protocol

PUA Processor Utilization Analysis

QdS Qualité de Service

QoS Quality of Service

RCIM Real-time Clock & Interruption Module

RM Rate Monotonic

RMA Rate Monotonic Analysis

RMUS Real-time Multi UAV Simulator

RO Receive Order

RPC Remote Procedure Call

RSVP Resource reSerVation Protocol

RTA Response Time Analysis

RT-CORBA Real-Time Common Object Request Broker Architecture

RTCP Real-time Tranport Control Protocol

RTI Run Time Infrastructure

RTIA Run Time Infrastructure Ambassador

RTIG Run Time Infrastructure Gateway

RTP Real-time Transport Protocol

RTPS Real-Time Publish-Subscribe

RTSJ Real-Time Specifications for Java

SCADE Safety Critical Application Development Environment

SER Systèmes Embarqués et Répartition

SIMNET SIMulation NETwork

SNA Systems Network Architecture

SOM Simulation Object Model

SRP Stack Resource Policy

SS Sporadic Server

STL Standard Template Library

STM Smart Time Management

STRICOM Simulation, TRaining and Instrumentation COMmand

STOW Synthetic Theatre Of War

TAO The ACE ORB

TCP Transmission Control Protocol

TDMA Time Division Multiple Access

TDMI Timed Distributed Method Invocations

TLSF Two Level Segregate Fit

TOS Type Of Service

TSO Time Stamped Order

TTP Time Tiggered Protocol

TTRT Target Token Rotation Time

UAV Unmanned Aerial Vehicle

UDP User Datagram Protocol

VRTP Virtual Reality Transfert Protocol

WAN World Area Network

WCET Worst Case Execution Time

WCT Wall Clock Time

WCTT Worst Case Transit Time

WF Worst Fit

Chapitre 1

Introduction Générale

1.1 Contexte et Problématique

Cette thèse intitulée "*Architecture de Simulation distribuée temps réel*" s'inscrit dans le projet plus global PRISE[1] dont l'objectif principal est l'étude des nouvelles approches et des nouveaux concepts pour les systèmes embarqués. Les systèmes embarqués sont des systèmes critiques soumis à des contraintes de sécurité et de sûreté de fonctionnement. La conception de ces systèmes est alors régie par des normes et des guides de sûreté rigoureux tels que la norme DO-178B [Fed03] pour les systèmes aéronautiques. En effet, s'il arrive que certains de ces systèmes embarqués subissent une défaillance, ils mettent en danger des vies humaines ou encore mettent en péril des investissements importants. De plus, dans le cas des systèmes embarqués, le bon fonctionnement du système n'est pas seulement conditionné par la validité des résultats logiques. En effet, ces systèmes doivent toujours donner des résultats avec le respect d'échéances temporelles (*i.e.* deadlines). De tels systèmes sont dits soumis à des contraintes de temps réel et ils doivent avoir des comportements temporels prédictibles [Sta88].

Au fil des années, les systèmes embarqués sont devenus de plus en plus complexes avec un nombre important de composants qui interagissent dans une chaîne de calcul global. Les systèmes temps réel complexes peuvent profiter des avancées dans les technologies des réseaux [Sta92, Kop11]. De tels systèmes sont appelés des systèmes distribués. Ils sont composés de plusieurs programmes séparés physiquement sur plusieurs nœuds (processeurs) qui interagissent ensemble au travers d'un réseau afin d'achever un calcul commun. En revanche, l'utilisation des systèmes distribués afin de répondre au problème des systèmes temps réel a soulevé deux problèmes majeurs :

1. l'hétérogénéité des applications réparties ainsi que de leurs supports d'exécution : systèmes d'exploitation différents, prise en compte des protocoles réseau, méthodes de communication hétérogènes, etc.
2. la complexité des applications ainsi distribuées rend l'étude du comportement temporel global difficile : durée d'exécution des programmes, durée des communications, prise en compte des pannes, stratégies de recouvrement, ... Il devient donc plus difficile de garantir le respect des échéances temporelles.

Historiquement, il appartenait aux développeurs d'applications de résoudre eux-mêmes les problèmes liés à la répartition : fiabilisation des communications, algorithmes répartis ou encore hétérogénéité des environnements. En particulier, ils devaient programmer directement le maté-

[1] *Plate-forme de Recherche pour l'Ingénierie des Systèmes Embarqués*

riel de communication de chaque machine, s'assurer de la coordination entre les composants, et choisir une représentation commune de l'information acceptée par tous les participants. Pour faciliter la conception et le développement d'applications réparties, des modèles de répartition ont été proposés, c'est-à-dire des ensembles d'abstractions permettant de spécifier les interactions entre composants. L'utilisation de modèles de répartition, mis en œuvre par des intergiciels (middlewares), permet donc au développeur de se concentrer sur les aspects fonctionnels de l'application. Elle affranchit le programmeur de certaines contraintes, telles que la gestion de la communication entre nœuds, ou l'hétérogénéité des architectures matérielles et logicielles. Le développement des standards pour les intergiciels tels que RPC[2] [BN84, Inc95] ou CORBA[3] [OMG98a, OMG98b] a permis de résoudre les problèmes d'hétérogénéité en proposant une charte commune permettant à tous les acteurs de se baser sur un mode de communication et d'exécution commun. Un intergiciel ou un middleware (*C.f.* Figure 1.1) doit être vu comme un logiciel spécialisé dans l'interconnexion de différents systèmes hétérogènes dans un environnement distribué [All03, Pal07].

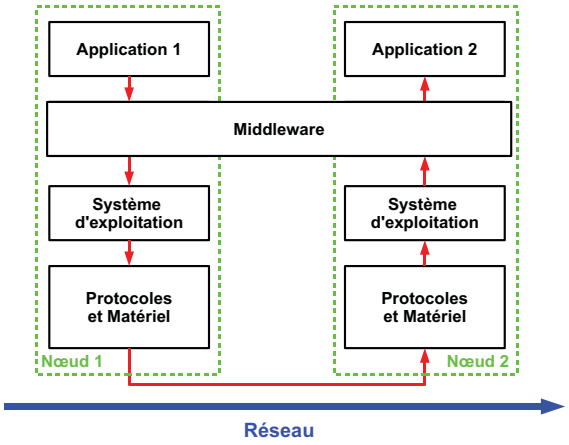


FIG. 1.1 – Illustration du rôle d'un intergiciel

Néanmoins, les normes et les modèles pour les technologies de distribution, dans leur version originale, n'étaient pas adaptés pour prendre en compte des contraintes temps réel. L'utilisation grandissante des middlewares pour l'interconnexion de systèmes temps réel a conduit à la création de différents projets de recherche spécifiques afin de concevoir des middlewares dédiés aux applications distribuées temps réel tels que ARMADA [ABD+97, ADF+97] ou MIDART [GSMT97]. De même, les concepteurs des standards ont fait évoluer les spécifications afin de s'adapter aux besoins spécifiques du temps réel. Ainsi la norme CORBA a évolué vers le standard RT-CORBA[4] [OMG05, SK00] et plus récemment DDS[5] [OMG04, OMG07]. Ces différents standards fournissent des spécifications supplémentaires permettant aux applications de s'assurer d'une certaine qualité de service ainsi qu'un comportement prédictible dans l'utilisation des services de l'intergiciel sous-jacent.

Le domaine de la simulation informatique désigne un ensemble de techniques et de méthodes selon lesquelles on exécute un programme informatique sur un ordinateur en vue de simuler un phénomène réel complexe (par exemple un phénomène chimique, physique, biologique, ...). Dans ce domaine de recherche, l'utilisation grandissante des technologies du réseau a aussi conduit à l'émergence des simulations particulières appelées des *simulations distribuées*. Ce type de si-

[2] *Remote Procedure Call*
[3] *Common Object Request Broker Architecture*
[4] *Real-Time CORBA*
[5] *Data Distribution Services*

mulation fait intervenir plusieurs simulateurs différents connectés par un ou plusieurs réseaux informatiques. De la même façon que pour les applications distribuées classiques, dans une simulation distribuée, l'interopérabilité entre les composants répartis est essentielle afin d'assurer un comportement global cohérent. En ce sens, l'ensemble des acteurs ainsi distribués doit communiquer et interagir en respectant un cadre commun défini par une norme de simulation distribuée telle que DIS[6] ou ALSP[7] (les standards existants sont décrits dans le chapitre 2.3). L'utilisation de tels standards permet la mise en place de modèles complexes pour l'exécution de simulations de plus en plus exigeantes en terme de puissance de calcul et de capacité de stockage. Le support logiciel permettant cette interconnexion est fourni par une architecture de simulation distribuée. Ainsi, ces intergiciels spécifiques, de la même façon que les intergiciels CORBA ou RPC, normalisent les communications et facilitent l'interopérabilité entre les simulateurs.

L'approche par simulation trouve toute sa justification dans l'ingénierie des systèmes complexes pour lesquels il est difficile d'avoir une vision globale, même si l'on est capable d'appréhender chacun des composants pris individuellement. Cela permet donc d'étudier le fonctionnement et les propriétés du système modélisé ainsi qu'à en prédire son évolution. Dans notre cas, l'approche par simulation distribuée permettra de mieux maîtriser la complexité des nouveaux concepts de systèmes embarqués. Ainsi, pour les besoins du projet PRISE, nous avons choisi d'utiliser les technologies de la simulation afin d'étudier les nouvelles approches de systèmes embarqués. Dans le domaine de la simulation distribuée, un standard particulier s'impose : le standard HLA (*High Level Architecture*) disponible sous deux formes :

- le standard US DoD[8] 1.3 [DoD98a, DoD98b, DoD98c]
- le standard IEEE[9] 1516 [IEE00b, IEE00c, IEE00d].

Cette norme permet d'interconnecter des simulateurs entre eux en définissant un format pour les échanges ainsi qu'un ensemble de règles facilitant l'interopérabilité et la réutilisation des applications de simulation. Ces deux propriétés sont recherchées pour la plate-forme PRISE afin de maintenir et enrichir les applications développées. Au niveau de l'architecture logicielle, le standard HLA repose sur un middleware spécifique nommé la RTI[10] qui constitue l'intergiciel spécialisé pour fournir des services de simulation. La RTI permet l'interconnexion de différents simulateurs compatibles avec le standard HLA (*C.f.* figure 1.2) sur une architecture distribuée.

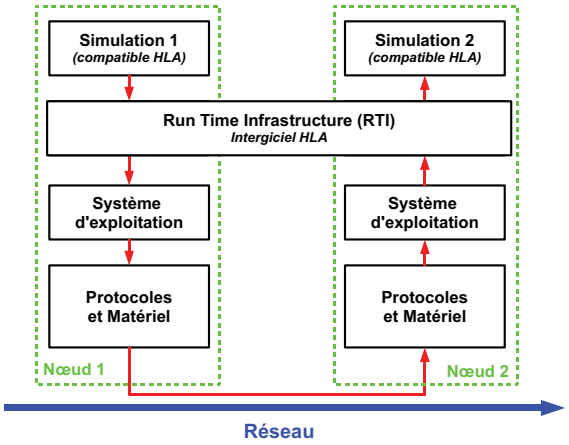


FIG. 1.2 – Illustration du rôle d'une RTI

[6] *Distributed Interactive Simulation*
[7] *Aggregate Level Simulation Protocol*
[8] *Department Of Defense*
[9] *Institute of Electrical and Electronics Engineers*
[10] *Run Time Infrastructure*

Les simulations de nos études pourront être des simulations *Hardware-in-the-loop* c'est-à-dire qu'elles incluront du matériel réel dans la boucle de simulation tels que des capteurs, des actuateurs ou encore des systèmes embarqués réels. De la même façon, ces simulations pourront être *Human-in-the-loop* c'est-à-dire qu'elles impliqueront un ou des opérateur(s) humain(s) dans la boucle de simulation ainsi que les outils pour lui permettre d'interagir avec la simulation pendant son exécution. Dans tous les cas, les simulations que nous souhaitons réaliser seront soumises à des contraintes temporelles qu'elles devront vérifier afin d'assurer un comportement global cohérent. Cependant, le standard HLA n'est pas adapté, dans sa version actuelle, pour répondre aux contraintes des systèmes temps réel. En effet, ce standard ne propose aucune spécification afin de garantir une certaine qualité de service ou une validation de bout en bout[11]. Par conséquent, la norme HLA souffre d'une mauvaise réputation pour l'interconnexion de systèmes temps réel. La problématique est donc de définir des mécanismes et éventuellement des spécifications qui permettraient d'utiliser une RTI (intergiciel HLA) pour des applications temps réel. Cette approche relève bien de la problématique d'une *Architecture de Simulation Distribuée Temps Réel* dont les principes, les spécifications et l'implantation sur une architecture matérielle et logicielle doivent permettre de répondre aux contraintes et aux exigences du système simulé. Notre travail se situe donc à la frontière de trois domaines distincts :

1. les **systèmes temps réel** ;
2. les systèmes distribués et particulièrement les technologies de **middlewares** ;
3. la **simulation distribuée** et particulièrement la norme HLA.

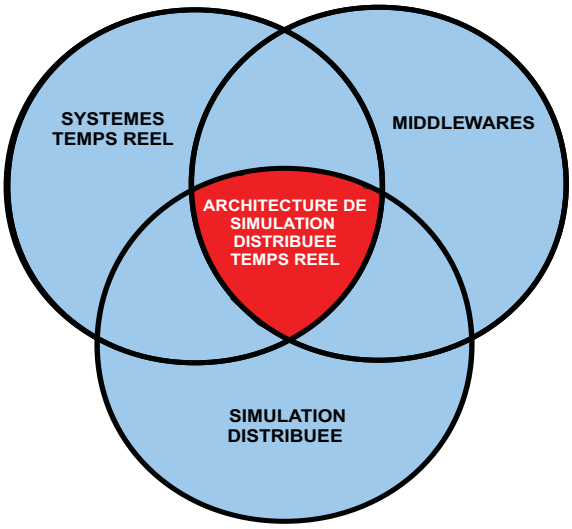


FIG. 1.3 – Intersection des domaines de recherche pour la thèse

1.2 Approche et Contributions

Depuis 1996, l'ONERA[12] développe et maintient son propre middleware RTI conforme à la norme HLA : le CERTI [Sir98, BS02]. Cet intergiciel est un logiciel Open Source disponible sur le site Web : `https ://savannah.nongnu.org/projects/certi` et fonctionnant sous plusieurs systèmes d'exploitation dont Linux et Windows. C'est ce logiciel que nous utiliserons pour ce

[11] *La validation du comportement temporel de toutes les exécutions et de toutes les communications du système distribué considéré.*

[12] *Office National d'Etudes et de Recherches Aérospatiales*

travail afin d'étudier les différentes pistes permettant d'obtenir des propriétés temps réel pour une simulation HLA et ainsi de répondre aux besoins de nos applications. L'avantage fondamental de l'utilisation du CERTI est que nous possédons ses spécifications, nous maîtrisons son code source et donc son fonctionnement afin d'assurer le déterminisme des applications.

Nous proposons différentes contributions pour répondre à la problématique d'une architecture de simulation distribuée temps réel :

1. le développement de cas d'étude spécifiques : une application concernant le domaine aéronautique et une autre application concernant le domaine spatial (plus précisément des satellites) ;
2. la définition de différents modes d'exécution spécifiques ;
3. l'étude de différents mécanismes de synchronisation relatifs à ces modes d'exécution ;
4. la proposition de modèles formels afin de décrire formellement une application de simulation distribuée ;
5. la proposition de méthodes algorithmiques associées aux modèles formels pour une validation de bout en bout d'une application de simulation distribuée ;
6. l'implémentation de différentes techniques et fonctionnalités afin de rendre le CERTI prédictible ;
7. la réalisation d'expérimentations et de mesures sur la plate-forme dédiée au projet PRISE ;
8. la proposition d'une méthode d'analyse systématique pour l'évaluation d'une simulation distribuée temps réel.

1.3 Organisation du document

Ce document est organisé en deux parties distinctes ainsi que des annexes.

1. La première partie porte sur **l'état de l'art** sur le domaine d'étude de cette thèse et comporte deux chapitres :
   - Le **chapitre 2** permet d'introduire les principes fondamentaux du domaine de la simulation. Les concepts de la simulation distribuée sont présentés ainsi que les différentes normes et technologies existantes pour la réalisation de telles simulations. Ensuite, nous nous intéressons plus précisément à la norme HLA ainsi qu'à la RTI implémentée par l'ONERA : le *CERTI*.
   - Dans le **chapitre 3**, nous étudions les différents travaux et évolutions proposés dans la littérature afin de permettre l'utilisation de la norme HLA pour répondre aux critères du temps réel. Nous parlons aussi des comparaisons avec les autres standards telles que RT CORBA ou DDS et leurs spécifications pour le temps réel.

2. La seconde partie présente **les contributions** de cette thèse et comporte trois chapitres ainsi que la conclusion :
   - Le **chapitre 4** porte sur la description de l'architecture de la plate-forme ainsi que sur les modes d'exécutions envisagés pour des simulations temps réel. Nous insistons aussi sur les techniques de synchronisation utilisées pour chaque mode d'exécution.
   - Le **chapitre 5** permet de présenter les différents modèles formels utilisés afin de valider le comportement temporel de nos applications. Un accent particulier a été mis sur l'étude des algorithmes de gestion du temps fournis par le CERTI car ces services proposent une alternative prometteuse pour la synchronisation d'une simulation distribuée.

- Le **chapitre 6** comporte l'ensemble des résultats expérimentaux réalisés pour les différentes applications considérées. Ce chapitre permet de valider expérimentalement la démarche adoptée dans nos travaux.
- Enfin, le **chapitre 7** permet de conclure et présenter les perspectives envisagées pour ces travaux de thèse.

3. Les annexes sont composées de trois parties distinctes ainsi que des références bibliographiques et netographiques :
   - L'**annexe A** présente les différentes solutions techniques possibles pour fournir des supports exécutifs prédictibles et déterministes pour l'exécution d'une application temps réel.
   - L'**annexe B** traite des différents points essentiels à prendre en compte sur les technologies des réseaux afin d'assurer des supports déterministes pour les communications dans un système temps réel distribué.
   - L'**annexe C** s'intéresse à la théorie de l'ordonnancement qui fournit des méthodes algébriques et algorithmiques pour vérifier le comportement temporel d'une application temps réel.
   - Dans ce travail, un effort a été fourni afin de rassembler une bibliographie complète (mais non exhaustive) des différents domaines de recherche. Cette partie des annexes est composée de deux volets bibliographiques : (1) le premier volet est composé des références bibliographiques annotées dans la suite du document sous une forme avec des lettres et une couleur bleu foncé telle que : [IEE00c] ; (2) la deuxième partie est composée des références netographiques annotées dans la suite du document sous une forme avec des chiffres et une couleur bleu cyan telle que : [1].

Première partie

ETAT DE L'ART

Chapitre 2

La simulation distribuée et la norme HLA

Préambule

> *Afin de bien cerner la problématique de cette thèse, il convient d'en aborder précisément chaque partie. Ce chapitre traite des notions relatives au domaine de la simulation. Nous introduisons, dans un premier temps, les concepts de base de la simulation. Nous évoquons ensuite les problématiques de la simulation distribuée ainsi que les différents standards et architectures créés pour mettre en place de telles simulations. Ensuite, nous présentons les fondements de la norme HLA qui a été choisie pour ce travail. Enfin, nous décrivons l'implémentation de cette norme réalisée à l'ONERA, nommée CERTI, et nous illustrons ses caractéristiques techniques.*

2.1 Les principes de la simulation

2.1.1 Généralités et avantages

Le terme simulation regroupe un ensemble de techniques et de principes permettant l'étude de systèmes réels à partir d'un modèle aussi fidèle que possible [Can01]. Les anglo-saxons utilisent généralement l'expression M&S[1] pour parler de ces techniques. L'utilisation de la simulation s'est de plus en plus répandue dans de nombreux domaines (militaire, aéronautique, spatial, ...) pour remplir des objectifs divers tels que le prototypage, l'évaluation de performance, l'aide à la prise de décision ou encore l'entraînement.

Les techniques de simulation proposent des avantages indéniables sur de vraies expériences pour des systèmes réels. Les expériences sur le système réel peuvent être coûteuses, dangereuses voire impossibles. Par exemple, le système réel est inexistant ou encore l'échelle de temps de la dynamique de son évolution est incompatible avec l'expérimentateur humain (soit beaucoup trop grande ou beaucoup trop petite). De plus, la manipulation facile des modèles (modification de paramètres) et la suppression des effets indésirables (perturbations sur le modèle réel) permettent d'avoir une vision plus complète du système étudié.

[1] *Modeling and Simulation*

2.1.2 Concepts de base

La formalisation de la théorie de la simulation a été introduite par Bernard P. Zeigler [ZPGK00]. Dans ces travaux, il propose un cadre formel regroupant quatre notions essentielles pour fondement de la simulation : le système source, le cadre expérimental, le modèle et enfin le simulateur (*C.f.* figure 2.1). Ces différentes notions sont reliées par les relations de modélisation et de simulation.

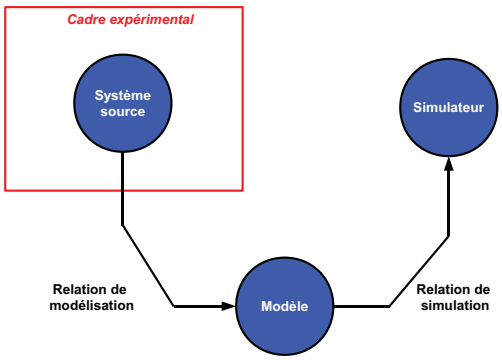


FIG. 2.1 – Les quatres notions essentielles de la théorie M&S

Le ***système source*** est l'objet que l'on cherche à étudier. Il peut être vu comme une source de données observables. Par exemple, il peut s'agir d'un phénomène physique ou chimique que l'on souhaite étudier ou encore d'un véhicule à concevoir.

Le ***cadre expérimental*** est une spécification des conditions dans lesquelles le système source est observé. Le cadre expérimental spécifie un contexte particulier correspondant à la situation envisagée pour l'étude. Ce cadre d'étude doit être délimité de façon adéquate afin d'assurer la pertinence du modèle et la précision des résultats obtenus par la simulation.

Le ***modèle*** est l'objet sur lequel on va travailler afin de réaliser la simulation. On le définit comme l'abstraction du système qui sera manipulé et animé pendant la simulation. Le modèle est conçu à partir d'un ensemble de règles ou formalismes (équations, contraintes, logique,...) qui définissent son comportement (*C.f.* partie 2.1.3). Il doit être suffisamment pertinent pour répondre à la problématique de l'étude envisagée.

Le ***simulateur*** est l'entité capable de modéliser l'évolution du système en fonction des instructions du modèle. C'est un système de calcul qui permet de produire le comportement d'un modèle à partir des règles qui régissent celui-ci. Selon les besoins du modèle, le simulateur peut être un dispositif technique (matériel et logiciel) plus ou moins évolué.

La ***relation de modélisation*** définit les parties du système source qu'il faut représenter et la manière dont elles le seront. La modélisation est donc l'opération par laquelle le modèle est construit à partir du système source et de son cadre expérimental.

La ***relation de simulation*** assure que le simulateur reproduit le comportement spécifié en respectant les instructions définies dans le modèle.

2.1.3 Modèle et Formalismes

Le modèle est une abstraction du système source vue dans le contexte du cadre expérimental. Par conséquent, il est indispensable d'avoir un modèle cohérent avec le dit système pour obtenir une simulation pertinente. Le choix d'un modèle s'effectue donc en fonction de nombreux critères tels que les objectifs de l'étude ainsi que l'infrastructure (matérielle et logicielle) choisie (ou imposée) pour réaliser la simulation. Les modèles et surtout leurs comportements peuvent être exprimés dans une variété de formalismes reposant sur des concepts et des propriétés mathématiques selon deux dimensions : la dimension temporelle et la dimension spatiale [Ore87, AK00]. La dimension temporelle peut être continue ou discrète et la dimension spatiale peut être décrite par un ensemble de valeurs fini ou infini. Les différents formalismes pour représenter un système résultent des choix pour les dimensions temporelles et spatiales. Par exemple, le formalisme des équations différentielles est adapté aux modèles à temps continu et à espace d'état infini.

Dans notre cas nous considérerons par la suite des modèles pouvant être décrits par un ensemble fini de variables. De la même façon, nous considérerons par la suite des simulations impliquant des modèles à temps discret. Les changements d'états sont discrets et correspondent à un avancement dans le temps en obéissant aux règles imposées par le modèle. Une exécution peut être vue comme une suite d'états[2] successifs du modèle et la valeur du nouvel état dépend en général des états passés (selon le principe du déterminisme, *C.f.* partie 2.1.6).

2.1.4 La problématique du temps

Il peut y avoir confusion entre différentes sortes de temps lorsque l'on se place dans le contexte de la simulation. En effet, cette notion de temps dans une simulation ne repose pas sur un concept unique. Il existe plusieurs définitions de temps pour une simulation, chacune décrivant un aspect temporel bien spécifique. Richard Fujimoto [Fuj90] propose de différencier ces notions en proposant trois définitions distinctes du temps :

- ***Le temps physique*** est le temps de référence du système physique (le système source que l'on veut modéliser et simuler).
- ***Le temps simulé*** est une représentation du temps physique pendant la simulation. C'est un ensemble de valeurs ordonné représentant différents instants pour le système physique modélisé.
- ***Le temps absolu*** ou temps d'horloge est le temps qui s'écoule pendant l'exécution de la simulation. Il peut être mesuré par le simulateur à partir d'une horloge matérielle telle que l'horloge processeur.

La progression de l'évolution de la simulation dans le temps discret peut se faire selon deux approches [Fuj90] : l'approche dirigée par temps (ou synchrone) et événementielle (ou asynchrone).

L'approche synchrone ou dirigée par le temps est basée sur une boucle temporelle d'un intervalle de temps simulé fixe T (qui peut éventuellement être synchronisée sur une horloge de temps absolu). Ainsi, l'évolution du modèle et de ses composants (sous-modèles, variables, etc ...) se fait de pas en pas. Il peut y avoir plusieurs actions réalisées pendant ce pas de temps. Tous les événements d'un même pas sont supposés indépendants mais valables pour un même instant. Le choix de ce pas de temps est primordial pour la cohérence et la performance de la simulation. Un pas trop petit amènerait à exécuter plusieurs boucles sans aucune action relative à la simulation. Ces avancées non pertinentes peuvent alors limiter les performances lors de l'exécution. Inversement, un pas trop grand amènerait à des problèmes de relation temporelle entre plusieurs

[2] *l'état étant les valeurs de l'ensemble des variables constituant le modèle à un instant t*

actions sur le même pas de temps tel que le problème de causalité (*C.f.* partie 2.1.6). En effet, les relations de cause à effet, ne peuvent être exprimées qu'entre deux pas de temps : de l'instant t (événement *cause*) à l'instant $t + T$ (événement *effet*).

L'approche asynchrone ou dirigée par les évènements, quant à elle, n'impose aucun pas de temps minimal. Dans ce cas, on avance dans le temps de simulation que lorsqu'un événement se produit. Le temps de simulation avance donc par intervalle de durée variable selon l'estampille temporelle des événements à traiter. Ainsi, dans cette approche, seuls les instants significatifs (associés à des événements) seront pris en compte par le simulateur. En revanche, la gestion de l'entrelacement des instants par événement fait appel à des algorithmes de synchronisation spécifiques (pour respecter les principes fondamentaux, *C.f.* partie 2.1.6). Ceci assure que toutes les parties de la simulation ont une compréhension unique des relations « avant et après » entre les actions simulées qui surviennent à des instants spécifiques du temps simulé.

2.1.5 Relation avec le temps réel

La relation entre le temps simulé et le temps physique est évidente car le temps simulé est la représentation discrète du temps physique (relatif au système source) pour la simulation. En revanche, cette progression dans le temps simulé peut être mise en relation, ou pas, avec le temps absolu (le temps d'horloge). Par exemple, dans les simulations pour des environnements virtuels le temps simulé doit avancer en synchronisation avec le temps absolu. En effet, dans ce type de simulation, l'inclusion d'un opérateur humain dans la boucle de simulation impose de garder une synchronisation entre temps d'horloge et temps de simulation car sinon l'environnement simulé paraîtra non réel pour l'opérateur. Si le temps de la simulation avance lentement par rapport au temps d'horloge alors l'environnement virtuel paraîtra lent et répondra aux interactions de l'utilisateur en retard. De même, si la simulation avance rapidement par rapport au temps d'horloge, les utilisateurs ne pourront pas interagir correctement et de façon pertinente avec la simulation.

Nous voyons donc ici apparaître la notion de *simulation temps réel*. Une simulation temps réel est une simulation dans laquelle le temps physique, le temps simulé et le temps absolu doivent être synchronisés. Néanmoins, dans le monde de la simulation, ce terme est souvent utilisé pour décrire d'autres types de simulation. Par exemple, les simulations temps réel désignent parfois les simulations «aussi-rapide-que-possible» parce qu'on voudrait souvent finir la simulation le plutôt possible. Cependant, cela n'impose en aucun cas de maintenir une relation fixe avec le temps d'horloge (le temps absolu). Dans notre cas, nous parlerons de simulation temps réel pour traduire la première notion dans laquelle le temps physique, le temps simulé et le temps d'horloge doivent être synchronisés. Nous nous intéresserons aussi à l'aspect du déterminisme, c'est-à-dire être sûr que la synchronisation et les calculs de la simulation assureront un comportement prédictible. Un tel comportement devrait permettre à la simulation de répondre à des contraintes temporelles (respect des deadlines) pendant toute son exécution.

2.1.6 Les principes de cohérence temporelle

Le comportement des systèmes physiques réels (système source) repose sur deux principes fondamentaux : le principe de *causalité* et le principe de *déterminisme*. Le principe de déterminisme définit que le futur du système peut être déterminé à partir de son état présent et de son état passé. Ainsi, à tout instant t, il existe une valeur positive ϵ telle que le comportement du système puisse être calculé jusqu'à $t + \epsilon$. Tout modèle d'un système physique devra respecter ce principe de déterminisme.

Le principe de causalité repose sur ce principe de déterminisme et a été formulé pour la première fois en 1959 par Mario Augusto Bunge [Bun59]. Ce principe définit que le futur d'un système physique ne peut jamais influencer son passé. L'état du système à l'instant présent t est indépendant de tout se qui peut se produire à toute date t' telle que $t' > t$. Tout simulateur d'un système réel devra garantir le respect de ce principe de causalité. La causalité impose donc un ordre des transitions d'états dans les systèmes physiques. L'ordre dans lequel le simulateur produit les événements doit être cohérent pour que le simulateur soit un reflet juste du système physique simulé.

Un exemple est proposé dans [Fuj98] pour expliquer le problème de causalité dans une simulation militaire (*C.f.* figure 2.2). Trois processus P_1, P_2 et P_3 s'exécutent chacun suivant leur propre temps simulé. Ainsi à un instant t_{reel} (du temps d'horloge), tous ne sont pas au même point du temps simulé : $t_{simule}(P1) \neq t_{simule}(P2) \neq t_{simule}(P3)$. P_1 déclenche un événement T qui représente le tir d'un projectile. Un message est envoyé alors aux deux autres processus logiques P_2 et P_3 (La réception de cet événement de tir est notée R_T). Le processus logique P_3 reçoit le message avant le processus logique P_2. En conséquence, le processus logique P_3 envoie un événement D pour spécifier la destruction de la cible et envoie un message aux deux processus logiques P_1 et P_2 (la réception de l'événement de destruction de la cible est notée R_D). Cependant, l'événement de destruction de la cible D est détecté par P_2 avant qu'il ne détecte l'événement du tir T ce qui induit un problème de causalité dans l'écoulement temporel de ces différents événements.

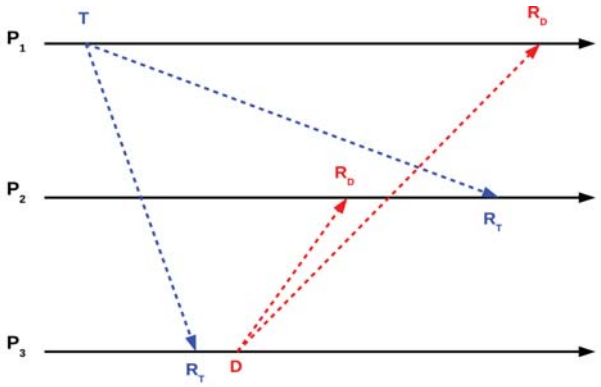


FIG. 2.2 – Illustration du problème de causalité

Ainsi, le principal problème dans la conception d'une simulation (et du simulateur) est donc de pouvoir gérer la cohérence temporelle du temps simulé. Il s'agit de trouver une représentation correcte du temps simulé et de définir un schéma d'exécution qui assure une progression correcte du simulateur dans ce temps simulé. Lamport [Lam78] fut le premier à définir une méthode basée sur l'utilisation d'estampilles temporelles (du temps simulé) pour permettre aux processus de respecter la causalité temporelle des systèmes physiques à simuler. Ce problème constitue un des défis de la simulation distribuée (*C.f.* partie 2.2, page 14) dans laquelle l'asynchronisme des sites constitue un problème majeur (les sites d'un système distribué ne possèdent pas de référence commune du temps absolu).

2.2 La simulation distribuée

2.2.1 Le contexte distribué

Les simulations séquentielles étaient fortement limitées par la performance de la machine supportant l'exécution globale de la simulation. La nécessité d'utiliser des modèles (ainsi que leurs comportements) de plus en plus complexes a amené l'émergence de l'utilisation des technologies de distribution pour la simulation. Ainsi, la simulation distribuée consiste à répartir l'exécution de la simulation sur plusieurs nœuds (processeurs ou ensemble de processeurs) à travers un réseau informatique. L'ensemble des simulateurs distribués participe à une simulation globale et peut être vue comme un unique simulateur. Le support d'exécution de la simulation est donc constitué d'un ensemble de processeurs asynchrones qui s'échangent de l'information par l'intermédiaire de messages. Cette solution permet ainsi de fédérer plus de ressources de calcul afin de simuler plus rapidement ou encore à réaliser des simulations qu'il était impossible de mettre en œuvre de façon séquentielle [RW89]. La simulation distribuée consiste donc à éclater le modèle global de simulation en sous-modèles et à simuler chacun de ces sous-modèles (ou processus de simulation) sur un nœud du système distribué. Il faut alors résoudre trois problèmes afin de réaliser une simulation sur un support distribué :

1. ***Le placement des différents processus de simulations sur les différents nœuds du système distribué :*** Ce problème est souvent ramené à un problème d'optimisation pour lequel on tente d'optimiser la charge des nœuds (due à l'exécution des simulateurs atomiques sur les processeurs) ainsi que la communication réseau (due à la taille des messages et à leur fréquence). On utilise alors des techniques telles que le recuit simulé [Kir83] qui fut utilisé en premier pour le placement de tâches sur des machines multiprocesseurs [BM91]. D'autres techniques ont été envisagées telles que celles des algorithmes génétiques [Hol75, HAR94]. Le problème complexe du placement des tâches dans un système distribué constitue un domaine de recherche à part entière, nous ne le traiterons pas dans ce chapitre.

2. ***L'interopérabilité des simulateurs :*** l'interconnexion de plusieurs simulateurs hétérogènes participant à une seule et même exécution implique de mettre en place une homogénéisation entre ces différents simulateurs pour définir des entités standards ainsi que des moyens de communication. Ce support d'interopérabilité est fourni par un standard de simulation distribuée (*C.f.* partie 2.3, page 17). L'exécution de la simulation est assurée par une architecture de simulation distribuée qui assure le déroulement de la simulation globale avec des simulateurs en concordance avec le standard choisi.

3. ***La construction d'un schéma d'exécution temporel :*** Ce problème revient à trouver un schéma qui garantisse la progression du temps simulé et assure que les processus du modèle (les simulateurs) sont exécutés correctement par rapport à ce temps simulé (la relation de causalité est bien respectée). On affecte alors une horloge logique du temps simulé à chaque simulateur et on synchronise les évolutions de chacune de ces horloges afin d'assurer un écoulement cohérent. Les solutions envisagées sont alors différentes si l'on se place dans le cadre de la simulation distribuée dirigée par le temps (*C.f.* partie 2.2.2) ou de la simulation distribuée dirigée par les événements [Mis86] (*C.f.* partie 2.2.3).

2.2.2 Synchronisation pour la simulation distribuée dirigée par le temps

Dans ce cas, de la même façon que pour la simulation séquentielle, toutes les horloges logiques (de temps simulé) progressent de concert. Lorsqu'elles ont la valeur t, les simulateurs exécutent les événements correspondants avant de progresser jusqu'à la valeur $t + T$ (T étant le pas de temps simulé de la simulation globale). L'horloge globale de ce temps simulé est alors répartie par duplication dans chacun des processus distribués. La solution la plus courante est d'utiliser le concept de *synchroniseur* introduite par Baruch Awerbuch [Awe85]. Plusieurs synchroniseurs ont été envisagés [LT88, CCGZ90] et se distinguent par leur complexité en temps et en nombre de messages ainsi que sur la technique de distribution (ils réalisent un contrôle plus ou moins centralisé). Généralement, l'évolution des processus est cadencée par une horloge globale dont les cycles sont appelés les *pulsations*. L'hypothèse de base repose sur l'idée que les délais de transferts et de traitements des messages sont inférieurs à cette pulsation. Ainsi un message émis au début d'une pulsation sera reçu et traité avant la début de la pulsation suivante. Sous ces hypothèses, les synchroniseurs permettent de maintenir une horloge globale dans chaque simulateur local sur une architecture distribuée. Une étude expérimentale de l'efficacité de ces différents synchroniseurs peut être trouvée dans [IRA88] (étude menée sur une architecture *iPSC* d'Intel). Néanmoins, les techniques de simulation distribuée dirigée par le temps sont bien moins utilisées que celles de la simulation distribuée dirigée par les événements qui proposent des solutions beaucoup plus adaptées au monde de l'informatique distribué.

2.2.3 Synchronisation pour la simulation distribuée dirigée par les événements

2.2.3.1 Les approches conservatives

Afin de préserver une cohérence temporelle dans l'entrelacement des événements, l'approche conservative (ou encore l'approche pessimiste) maintient en permanence un ordre correct dans l'exécution des événements. Dans ce cas le système de simulation n'évoluera que si il considère qu'il n'y a aucun problème d'incohérence temporelle. C'est-à-dire que chaque simulateur ne traitera le message que lorsqu'il est sûr de ne pas recevoir un autre message avec une date de traitement inférieure. Cette approche peut conduire à des situations d'inter-blocage entre les processus de simulation particulièrement lorsqu'il existe des boucles de communications entre les simulateurs. Cela nécessite donc de mettre en place des algorithmes afin de prévenir ces inter-blocages (et donc les éviter) ou bien de les détecter et les résoudre afin de poursuivre normalement l'exécution ensuite.

Afin de prévenir toute situation de blocage, un algorithme a été proposé conjointement par Chandy et Misra [CM79] et Bryant [Bry77] et il est communément appelé l'algorithme CMB ou algorithme des messages *NULL*. L'idée de base de cette solution est de faire émettre aux processus de la simulation des messages de contrôle appelés messages *NULL* en plus des messages de la simulation. Ces messages *NULL* sont des messages qui ne correspondent à aucun événement de la simulation mais ils sont estampillés par une date (de temps simulé). Cette approche est aussi basée sur une notion supplémentaire : le *lookahead* définie comme une capacité du simulateur de prédire son comportement futur (cette valeur de temps simulé est notée L). C'est une sorte de contrat pour lequel le simulateur est en mesure d'assurer que à un instant t du temps simulé, il ne produira pas d'événement de simulation avant le temps $t + L$. Nous verrons par la suite que les valeurs de ce lookahead sont un paramètre discriminant pour la performance de la simulation globale [Pre90, Jan98]. Certaines variantes de cet algorithme ont été proposées [SS88] telles

que l'algorithme des messages *CARRIER-NULL* [WT94, CT95] néanmoins l'algorithme CMB originel reste le plus utilisé.

Les approches de détection consistent à détecter une situation d'inter-blocage (ou de terminaison) une fois qu'elle a eu lieu plutôt que de la prévenir [CM81]. Cette technique consiste à simuler jusqu'à l'apparition d'un inter-blocage, puis le guérir en relançant l'exécution d'un ou plusieurs processus de simulation. Dans ce cas, c'est un processus de contrôle qui est chargé de détecter la présence de l'interblocage en observant tous les processus de la simulation. Pour cela, il met en œuvre un algorithme reparti tel que celui de Dijkskra et Scholten [DS80], Obermack [Obe82] ou encore les méthodes de Chandy et Misra [MC82, CMH83]. Le lecteur intéressé pourra trouver un état de l'art sur les différents algorithmes de détection dans l'article de Ahmed Elmagarmid [Elm86].

D'autres approches consistent à utiliser un schéma de coloration afin de determiner un état global de l'ensemble de la simulation pour les canaux FIFO[3] [CL85, SK86] et les canaux non fifos [LY87, Mat89, Mat93]. Cette approche est basée sur la notion d'état des processus : (1) état avant enregistrement de l'état local et (2) état après l'enregistrement. On associe une "couleur" à chacun de ces états et ainsi chaque simulateur doit rendre compte de son état local à un processus de contrôle qui est le seul capable de connaître l'état global de l'ensemble de la simulation. De la même façon que pour les approches de détection de terminaison et d'inter-blocage, cette approche nécessite le contrôle global par un processus centralisé alors que ce contrôle est totalement décentralisé dans tous les simulateurs pour l'algorithme CMB.

Une autre approche repose sur le concept des fenêtres temporelles entre les processus de simulation [AR92]. Pour cela, on ajoute l'hypothèse d'un délai additionnel de propagation du message à la valeur de l'estampille de l'événement produit par un simulateur. Ainsi, moyennant des resynchronisations entre les processus (selon la borne inférieure des différentes fenêtres), le principe de causalité peut être respecté.

La compréhension de ces différents algorithmes et de leur spécificités est importante dans notre cas. En effet, certains de ces algorithmes sont implémentés pour assurer les mécanismes de gestion du temps du standard HLA (*C.f.* partie 2.5, page 31). Leur compréhension est donc essentielle afin d'assurer un certain déterminisme dans l'utilisation de ces services dans le contexte des simulations temps réel. Le lecteur pourra se référer au livre de Michel Raynal [Ray92] ou au rapport de Richard Fujimoto [Fuj90] dans lesquels l'ensemble de ces algorithmes, leurs spécificités ainsi que des exemples et des illustrations sont disponibles.

2.2.3.2 Les approches optimistes

L'approche optimiste (ou approche spéculative) est en contraste avec l'approche précédente. En effet, aucune prévention des fautes temporelles n'est effectuée et le système de simulation évolue jusqu'à ce qu'une incohérence temporelle survienne. Par exemple, un message peut parvenir à un simulateur avec une date de traitement (estampillé sur le temps logique) qui est inférieure à la date de simulation atteinte par ce processus. Dans ce cas, on effectue des retours en arrière pour revenir à un instant cohérent et on reprend la simulation à ce point. Le principal mécanisme optimiste, nommé le *Time Warp*, a été proposé par Jefferson [Jef85].

Afin de pouvoir réaliser correctement les retours en arrière, Jefferson a défini le concept de *Global Virtual Time* (noté GVT) qui représente la date minimale d'un retour en arrière (en temps simulé). Cette date est calculée à partir des temps locaux de chaque processus ainsi qu'aux

[3] *First In First Out*

estampilles temporelles des messages en transit. Cela nécessite de sauvegarder sur chaque site (1) les états locaux successifs, (2) la liste de messages estampillés (dans le temps virtuel) afin de pouvoir les re-consommer et (3) la liste de messages estampillés afin de pouvoir, si cela est nécessaire, retrouver ceux dont il faut détruire les effets à l'aide d'anti-messages. Un problème de mémoire se pose car toutes ces sauvegardes peuvent très vite occuper un large espace mémoire sur chaque site, le défi consiste donc à maîtriser le besoin en sauvegarde [ACD+93].

Différents mécanismes de rollback tels que l'*Aggressive Cancellation* ou la *Lazy Cancellation* ont été étudiés [Gaf88, MWM88, Wil97, TPF+05] afin de réduire les temps d'exécution des rollbacks (souvent trop importants) et de réduire les besoins en ressources mémoires. L'approche optimiste de Jefferson a été expérimentée et testée sur un calculateur parallèle à mémoire répartie et a mené à un système d'exploitation distribué : Le *Time Warp Operating System* [JBW+87]. Behrok Samadi, dans sa thèse [Sam85], propose une étude complète (théorique et expérimentale) de l'efficacité de certains algorithmes optimistes de rollback ainsi que des mécanismes pessimistes de détection et guérison.

2.3 Architectures et standards pour la simulation distribuée

2.3.1 Historique

Historiquement, ce sont les simulations militaires qui se sont développées en premier. En effet, l'évolution économique mondiale a provoqué une réduction des dépenses militaires pour favoriser d'autres secteurs économique. Afin de conserver un entraînement des troupes suffisant, l'apparition des simulations distribuées et la mise en place de simulateurs en réseau ont apporté de nombreux avantages tant économiques (coût et maintenance) que écologiques par rapport aux exercices effectués en grandeur nature. Le développement de ces simulations a débuté en 1929 avec le *First Link Trailer* qui était un simulateur de combat aérien [Con94]. Depuis l'armée américaine (prédominante dans ce domaine de recherche) a déployé d'énormes moyens pour améliorer les technologies de simulation afin de concevoir et de maintenir des environnements virtuels avec lesquels les utilisateurs pourront interagir comme s'ils étaient dans ce monde virtuel (simulation d'entrainement). Les simulateurs, toujours plus complexes, ont évolué vers les technologies de simulation repartie ou distribuée. Ainsi, sous l'égide de la DARPA[4] et de l'armée américaine, le projet SIMNET[5] a débuté dans les années 1980 grâce à l'utilisation du réseau ARPANET [BI81]. Le but de cette technologie était de permettre à de petites unités militaires (une compagnie par exemple) d'apprendre à s'organiser, communiquer et combattre en équipe par l'intermédiaire de réseaux informatiques [Cal93]. Les recherches et les développements, menés de 1983 à 1990, ont mené à l'élaboration d'un protocole de communication utilisé pour l'échange d'informations entre simulateurs placés dans un environnement virtuel commun. Les différentes expérimentations et démonstrations ont montré la viabilité de la mise en réseau de plusieurs simulateurs individuels. Ainsi, en mars 1993, le projet SIMNET amena à la naissance du protocole standard DIS (*C.f.* partie 2.3.2) ainsi qu'au projet ALSP (*C.f.* partie 2.3.3). Ces différentes technologies ont permis d'aboutir à la norme aujourd'hui en vigueur : la norme HLA (*C.f.* partie 2.4, page 23). La figure 2.3, tirée du document de thèse de Hassen Jawhar Hadj-Amor [HA08], résume cet historique.

[4] *Defence Advanced Research Project Agency*
[5] *SIMulation NETwork*

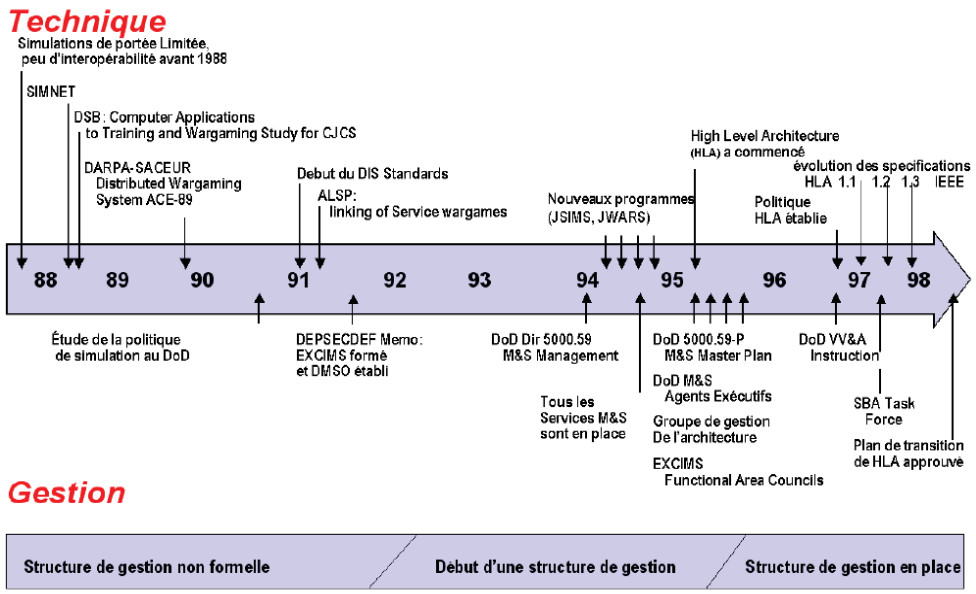


FIG. 2.3 – Historique des standards de simulation distribuée

2.3.2 DIS

Le protocole DIS[6] a été développé pour améliorer les applications fonctionnant avec SIMNET [Com94]. Il repose sur la base des principes concluants de SIMNET mais en proposant une norme plus complète notamment en apportant un formalisme pour le format des paquets échangés entre les simulateurs. Ce standard est approuvé par l'IEEE [IEE93a, IEE95a, IEE95b], sous le nom IEEE 1278 et il est encore d'actualité aujourd'hui. Ce standard est principalement basé sur la définition d'un protocole de communication entre plusieurs simulateurs qui peuvent ainsi interopérer et partager un environnement tactique commun. Ce protocole de communication repose sur l'échange de messages de bas niveaux et précise le codage des octets au niveau binaire. Ces messages appelés PDU[7] sont alors échangés par les différents objets de la simulations (char, avion, fantassin,...). Le standard DIS définit 27 PDUs (représentés dans la table 2.1) qui permettent aux entités (objets de la simulation) de déclarer les variations de leur état et d'effectuer des interactions (un tir par exemple) sur le théâtre de la simulation. Les PDUs définissent l'information échangée entre les sites de simulation.

Dans une simulation DIS, le monde est modélisé comme un ensemble d'entités (les objets de la simulation) qui interagissent les unes avec les autres par le biais d'événements, qu'elles créent et qui peuvent être perçus par les autres entités. Le fonctionnement d'un simulateur conforme au standard DIS repose sur le paradigme ***Joueurs et Fantômes***. Chaque objet de la simulation est géré sur son propre ordinateur par un objet logiciel appelé *le joueur*. Et, sur toutes les autres stations participant à la simulation, une version simplifiée du joueur est modélisée dynamiquement par un objet logiciel appelé *le fantôme*. De cette façon, les objets fantômes mettent à jour leur position à chaque itération de la boucle de simulation grâce à un algorithme d'extrapolation appelé de *dead-reckoning* qui permet de limiter l'utilisation de la bande passante du réseau [Aro97]. La

[6] *Distributed Interactive Simulation*
[7] *Protocol Data Unit*

Nom du PDU	Explication
`Acknowledge`	Acquitte la réception des PDUs `Start/Resume`, `Stop/Freeze`, `Create Entity` et `Remove Entity`
`Action Request`	Envoie une requête à une entité pour effectuer une action
`Action Response`	Acquitte la réception d'un PDU `Action Request`
`Collision`	Produit par une entité lorsqu'une entité détecte une collision avec une autre entité
`Comment`	Permet de transmettre un message arbitraire
`Create Entity`	Informe de la création d'une nouvelle entité
`Data`	Répond aux requêtes `Data Query` ou `Set Data`
`Data Query`	Effectue une requête à une entité pour obtenir des données
`Designator`	Désigne une opération à une entité
`Detonation`	Transmet la détonation ou l'impact d'une munition
`Electromagnetic Emission`	Véhicule l'information sur l'émission électronique et la contre-mesure
`Entity State`	Comporte les informations concernant l'état d'une entité particulière
`Event Report`	Permet à une entité de rapporter l'occurrence d'un événement important au gestionnaire de simulation (tels que un dommage, une mort,...)
`Fire`	Communique le tir d'une arme
`Receiver`	Communique l'état courant du récepteur (éteint, allumé ou en réception)
`Remove Entity`	Informe du retrait d'une entité de la simulation
`Repair Complete`	Permet à l'entité de réparation de notifier à l'émetteur d'un `Service Request` que la réparation est terminée
`Repair Response`	Acquitte la réception d'un message `Repair Complete`
`Resupply Cancel`	Annule un service soit par le récepteur, soit par l'émetteur
`Resupply Offer`	Communique une offre de ravitaillement
`Resupply Received`	Acquitte la réception d'un ravitaillement
`Service Request`	Permet d'effectuer une requête pour obtenir de la logistique
`Set Data`	Initialise ou change l'information d'un état interne
`Signal`	Transmet la voix, le son et d'autres données
`Start/Resume`	Communique le départ et la reprise d'un exercice
`Stop/Freeze`	Communique l'arrêt et le gel d'un exercice
`Transmitter`	Fournit des informations détaillées sur un émetteur radio

TAB. 2.1 – Liste des PDUs du protocole DIS

cohérence temporelle des événements de simulation était vérifiée à l'aide d'une horloge commune GPS[8] ou bien à l'aide du protocole NTP[9] [CL94]. Ces différentes techniques fournissaient une précision suffisante pour assurer la qualité de service recommandée par le standard DIS. En effet, la technologies DIS était principalement destinée aux applications temps réel car souvent un opérateur humain était intégré dans la boucle de simulation. Nous traiterons l'aspect de qualité de service et de temps réel de DIS dans le chapitre suivant (*C.f.* partie 3, page 39).

Le standard DIS a permis de réaliser de grands projets de simulation regroupant plusieurs acteurs. Le projet STOW[10] est un bon exemple qui démontra la puissance de la technologie DIS [Tie96]. Il offrit une infrastructure d'exécution pour des exercices interarmées faisant in-

[8] *Global Positionning System*
[9] *Network Time Protocol*
[10] *Synthetic Theatre of War*

tervenir des simulateurs pilotés, des jeux de guerre ainsi que des unités de terrains réelles. Ces simulations étaient composées de milliers d'entités réparties géographiquement entre les USA, le Royaume-Uni et l'Allemagne. Le standard DIS a été créé pour des applications purement militaires et devint même le standard de simulation distribuée chez les militaires occidentaux (standard OTAN STANAG 4482). Malgré ce succès, ces spécificités militaires constituaient un inconvénient majeur pour son application dans d'autres domaines, et ce standard ne réussit pas à s'imposer dans le domaine civil. Par exemple dans notre cas, il n'est pas adapté pour simuler des systèmes embarqués spatiaux ou aéronautiques car les PDUs sont définis pour les simulations militaires.

2.3.3 ALSP

Le standard ALSP[11] est un protocole de simulation réalisé par la MITRE Corporation au début des années 90 pour le compte du STRICOM[12]. L'objectif était de faire interagir, au travers de réseaux, des simulations constructives existantes [Dub93, WW94]. ALSP propose un protocole de communication de haut niveau qui utilise des messages textuels. Ce principe est plus souple que les messages binaires de DIS (les PDUs), et facilite ainsi leur interprétation (par un superviseur humain par exemple). En revanche, l'utilisation de ce type de message augmente considérablement le trafic réseau. De plus, ALSP fournit des couches logicielles de services pour faciliter l'utilisation et le contrôle d'une simulation globale. Les simulateurs (notés les acteurs) sont interconnectés au sein de confédérations et manipulent des entités appelées objets, échangeant des informations sur eux à travers une architecture logicielle, l'AIS[13]. L'AIS comprend trois composants essentiels (*C.f.* figure 2.4) :

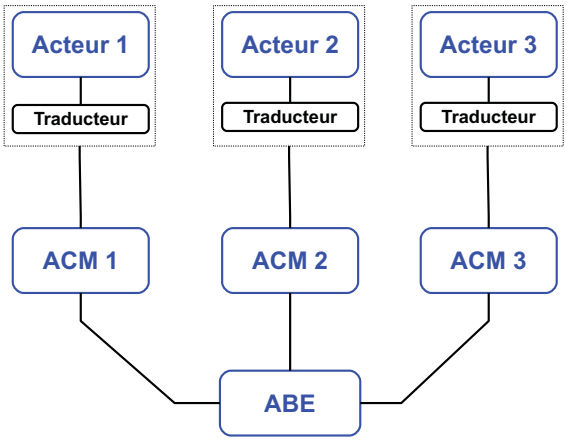


FIG. 2.4 – Architecture d'une confédération avec trois acteurs

Le traducteur est une portion de code qui assure la liaison entre l'acteur et la confédération en convertissant les données au format proposé par ALSP. Il peut être intégré au code de l'acteur ou extérieur comme par exemple, une bibliothèque liée à l'acteur lors de l'édition de liens de l'application. Cet élément convertit les événements internes en messages ALSP et vice versa. Il coordonne aussi l'horloge de l'acteur avec le temps de la confédération selon des mécanismes de gestion du temps (respect de la causalité). **L'ACM**[14] est un composant logiciel greffé à chacun des

[11] *Aggregate Level Simulation Protocol*
[12] *Simulation, TRaining and Instrumentation COMmand of the US Army*
[13] *ALSP Infrastructure Software*
[14] *ALSP Common Module*

acteurs de la simulation. L'ACM coordonne les adhésions et les démissions de la confédération, filtre les messages arrivant et gère la propriété ainsi que les droits d'accès aux attributs. **L'ABE**[15] est un processus chargé de la communication entre les différents ACMs. Sa principale fonction est de redistribuer les messages sur ses différents canaux de communication.

Un des avantages de l'architecture logicielle proposée par ALSP est la possibilité de relier deux ABE en réalisant ainsi une interconnexion de confédérations. Cela donne une grande souplesse dans l'organisation et l'optimisation d'un réseau de simulation distribuée basé sur ALSP. Cela a permis le développement de partenariat et la coopération entre différents acteurs dans une simulation ALSP, telle que la JTC[16] [MZ96]. Cette simulation grande échelle a permis la collaboration de différents corps de l'armée américaine tels que l'US Marine Corps, l'US Air Force ou encore l'US Navy. Un autre avantage principal d'ALSP est son mécanisme de gestion du temps. En effet, ALSP coordonne le temps entre les acteurs d'une confédération, qui n'ont pas forcément le même temps local (temps simulé). La méthode de synchronisation est basée sur l'algorithme des messages NULL de Chandy, Misra et Bryant comme cela est précisé dans [CPW+96]. En pratique, le simulateur (ou l'acteur) demande, via l'ACM, l'autorisation d'avancer par l'envoi d'un message `AdvanceRequest()`) et attend l'accord (message `GrantRequest()`) pour cette avance. Un telle mécanisme garantit le respect du principe de causalité entre les différents acteurs d'une simulation ALSP sans avoir recours à une horloge commune comme dans les simulations DIS.

2.3.4 La naissance de la norme HLA

Au milieu des années 90, deux solutions concurrentes étaient disponibles pour l'interconnexion de simulations : DIS, plus orienté simulation interactive (pouvant inclure un opérateur humain dans la boucle), et ALSP, pour les simulations constructives non temps réel (simulations analytiques telles que l'étude de scénarios). Ces deux technologies utilisaient des principes bien différents et n'étaient pas interopérables (compatibles) entre elles. DIS, malgré un succès indéniable au niveau militaire, finissait par s'enliser dans des considérations techniques de bas niveau, et ne remplissait pas ses promesses d'interopérabilité. De plus, ce standard était aussi mal adapté aux besoins des simulations à caractère plus général. Cela obligeait les utilisateurs non-militaires à développer leurs propres outils propriétaires. ALSP, quant à lui, était un protocole particulier destiné à une communauté bien précise et ne permettait pas son ouverture et son utilisation. Ces inconvénients ont entraîné une rapide disparition de ce standard à la fin des années 90. Néanmoins, l'architecture proposée par ALSP était suffisamment intéressante pour tenter de fusionner les principes de DIS et de ALSP. Le DMSO[17] a alors établi un ensemble d'objectifs relatifs aux développements de nouvelles simulations [DoD95]. Ces initiatives ont abouti à la création de la norme HLA en tant que standard fournissant une architecture de haut niveau, générique et ouverte remplissant les promesses d'interopérabilité pour tout type de simulation. En effet, HLA n'est pas spécifique au domaine militaire et propose de réunir au sein de la même simulation des simulateurs d'origines (métiers) différents. En plus, HLA développe la réutilisation des simulations qui facilite le maintien et l'intégration de simulations pré-existantes dans de nouveaux projets globaux. Ainsi certains cas d'études propres de ALSP et DIS ont pu être directement testés avec le standard HLA tels que la confédération JTC [PB97, GPFF97] ou l'environnement virtuel STOW [MAJ+98, MB98]. De même, certaines parties ou l'intégralité des simulations existantes SIMNET, DIS ou ALSP ont aussi pu être connectées à des simulations globales HLA grâce au développement de passerelles [CWP97]. Le standard HLA est décrit dans la partie 2.4.

[15] *ALSP Broadcast Emulator*
[16] *Joint Training Confédération*
[17] *Defense Modeling and Simulation Office*

2.3.5 Analogie avec le standard CORBA

CORBA[18] est né dans les années 1990 du besoin de faire communiquer ensemble des applications en environnement hétérogène avec des systèmes d'exploitation et des langages différents. De la même façon que HLA, CORBA est une norme [OMG98a, OMG98b] créée par l'OMG[19] (consortium regroupant 750 membres : vendeurs de logiciels, développeurs et utilisateurs) dans le cadre du projet ODP-RM[20]. L'objectif était de maximiser la réutilisation des logiciels pour le développement d'applications orientées objets. Ce standard permet à des composants logiciel d'interagir par l'envoi de messages, indépendamment du système d'exploitation, du langage de programmation ou de l'infrastructure matérielle d'exécution. Pour cela, CORBA définit :

- un modèle objet ainsi qu'une architecture de communication client/serveur entre ces composants distribués ;
- les spécifications d'un mécanisme de communication entre les composants nommé l'ORB[21] qui constitue l'intergiciel CORBA ;
- la description d'une interface de programmation commune (API) pour les services ;
- un langage de description d'interface spécifique (IDL) ainsi qu'un ensemble de traducteurs permettant de produire le code correspondant dans le langage requis.

CORBA n'est donc pas un standard dédié à la simulation distribué néanmoins la filiation entre CORBA et HLA est évidente de part leurs spécifications basées sur le paradigme de programmation orientée objet. De plus, la première implémentation du standard HLA, fournie gratuitement par le DMSO entre 1997 et 2002 était basée sur la norme CORBA [HNB97, BFMT98]. En effet, la RTI-NG était conçue au-dessus de l'ORB TAO[22][70, 71] basé sur la structure ACE[23] [1]. D'autres travaux ont fondé leurs implémentations HLA au dessus des services proposés par le standard CORBA (et les ORBs sous-jacents) [AA00, HMPM01]. CORBA est un standard plus mature et largement utilisé, nous verrons dans la suite du document que ce standard a notamment proposé des extensions pour répondre aux contraintes du temps réel (*C.f.* partie 3.2.2, page 50). Le tableau 2.2 présente quelques ORBs (logiciel CORBA) disponibles pour interconnecter des systèmes distribués.

Nom du logiciel	Distributeur	Licence	Languages supportés	Références
`VisiBroker`	Borland	Commerciale	C++, .Net, Java	[78]
`MICO`	Object Security	GPL	C++, .Net	[40]
`ROBIN`	Fermi National Accelerator Laboratory	GPL	C++, Java	[58]
`TAO`	Object Computing Inc.	GPL	C++	[70, 71]
`Omni ORB`	Independant	GPL	C++, Python	[44]

TAB. 2.2 – Liste de quelques ORBs disponibles

[18] *Common Object Request Broker Architecture*
[19] *Object Management Group*
[20] *Open Distributed Processing Référence Model*
[21] *Object Request Broker*
[22] *The ACE ORB*
[23] *Adaptative Communication Environment*

2.4 Les principes de la norme HLA

2.4.1 Concepts de base

Le projet de fusion de ALSP et de DIS a d'abord abouti à la création du standard HLA US DoD 1.3 [DoD98a, DoD98b, DoD98c]. Cette norme fut ensuite approuvée par l'IEEE[24] sous le nom HLA IEEE 1516 en septembre 2000 [IEE00b, IEE00c, IEE00d]. Ce standard IEEE 1516 a été récemment mise à jour (en 2010) pour intégrer un certain nombre de modifications [Mö08, Mö09]. HLA a aussi été standardisée par l'OMG [OMG02] et l'OTAN[25] sous le nom STANAG 4603. HLA est une spécification pour une architecture de haut niveau et non une implémentation particulière (un logiciel). Ainsi de la même façon que pour les précédents standards, l'objectif est de réunir plusieurs simulateurs au sein d'une même exécution.

Une **fédération** est le terme HLA qui désigne une simulation globale (distribuée ou non) composée d'un ensemble de simulateurs élémentaires (*fédérés*) s'échangeant des informations. Un **fédéré** est donc le terme désignant un des membres de la fédération tel qu'une application de simulation, un collecteur de données ou encore une interface pour un opérateur humain dans le cas d'une simulation *Human-in-the-loop*. La **RTI**[26] constitue l'implémentation informatique des spécifications d'interface de HLA. Cet intergiciel (logiciel distribué) est l'outil permettant l'interconnexion effective et l'exécution globale.

Ce standard HLA, tel qu'il est décrit dans les deux normes, est constitué de 4 éléments :

1. Les **règles** de HLA que toute simulation développée à l'aide de ce standard doit respecter (*C.f.* partie 2.4.2, page 24).

2. Un modèle objet générique **OMT**[27]) qui fournit une standardisation de la documentation du modèle objet HLA (*C.f.* partie 2.4.3, page 24).

3. Une spécification d'interface de programmation **API**[28] regroupant un ensemble de fonctions bas niveau qui permettent de programmer des applications utilisant les services standards de l'architecture (*C.f.* partie 2.4.4, page 26). Ces services sont fournis par la RTI[29] qui est l'implémentation de la norme (*C.f.* partie 2.4.5, page 30).

4. Un certain nombre de recommandations pour la conception et le développement des fédérations HLA : le **FEDEP**[30]. Il ne faisait pas partie intégrante de la norme 1.3 mais il est complètement intégré dans la norme IEEE 1516 [IEE03, IEE07].

Les deux normes en vigueur (US DoD 1.3 et IEEE 1516) sont globalement les mêmes dans les principes et les fonctionnalités qu'elles proposent. Néanmoins certaines différences apparaissent au niveau des spécifications d'interface telles que des noms de services différents, des types de données différents ou encore quelques principes sémantiques qui peuvent être différents. Le lecteur pourra trouver un guide précis et détaillé de l'ensemble de la norme HLA dans le document [Ade10].

[24] *Institute of Electrical and Electronics Engineers*
[25] *Organisation du Traité de l'Atlantique Nord*
[26] *Run Time Infrastructure*
[27] *Object Model Template*
[28] *Application programming Interface*
[29] *Run Time Infrastructure*
[30] *Federation Development and Exection Process*

2.4.2 Règles

Les standards définissent dix règles régissant toutes les simulations HLA [DoD98a, IEE00c]. Le respect de ces règles est indispensable pour assurer le bon déroulement d'une simulation HLA. Les cinq premières concernent le fonctionnement global des fédérations et les cinq dernières concernent le fonctionnement des fédérés :

Règle 1 : Les fédérations doivent avoir un modèle objet de fédération FOM[31] conforme à l'OMT de HLA. Le FOM doit décrire toutes les données échangées au cours de l'exécution d'une fédération et doit indiquer les conditions de ces échanges.

Règle 2 : Les représentations d'objets associés à la simulation doivent être dans les fédérés et non dans la RTI.

Règle 3 : Au cours de l'exécution de la fédération, les données décrites dans le FOM ne peuvent faire l'objet d'échange entre fédérés que par l'intermédiaire de la RTI.

Règle 4 : Au cours de l'exécution de la fédération, les fédérés doivent interagir avec la RTI conformément à la spécification de l'interface HLA.

Règle 5 : Au cours de l'exécution de la fédération, l'attribut d'une instance d'un objet ne peut être la propriété que d'un seul fédéré à n'importe quel instant.

Règle 6 : Les fédérés doivent avoir un modèle objet de simulation SOM[32] conforme à l'OMT de HLA. Le SOM correspond aux informations qui pourront êtres rendues publiques au cours de l'exécution de la fédération.

Règle 7 : Les fédérés doivent mettre à jour et prendre en compte des modifications d'attributs des objets, et aussi envoyer et/ou recevoir des interactions conformément à la spécification de leur SOM.

Règle 8 : Les fédérés doivent être capables de transférer ou d'accepter la propriété des attributs dynamiquement pendant l'exécution d'une fédération, conformément aux spécifications indiquées dans leurs SOM.

Règle 9 : Les fédérés doivent pouvoir faire varier les conditions sous lesquelles ils fournissent des mises à jours des attributs des objets, conformément aux spécifications de leur SOM.

Règle 10 : Les fédérés doivent pouvoir être capables de contrôler (avec l'aide de la RTI) le temps local d'une manière qui leur permettra de coordonner l'échange de données avec les autres membres de la fédération.

2.4.3 Le modèle objet OMT

L'OMT [DoD98b, IEE00b] est un formalisme qui permet de décrire, de façon orientée objet, les entités utilisées durant l'exécution d'une fédération et les interactions possibles entre ces entités. L'OMT est donc un support d'échange standardisé entre les différents développeurs qui peuvent intervenir dans la création d'une fédération mais aussi d'une aide pour la réutilisation des modèles objets ainsi définis. Il existe trois types de modèles objets : celui de la fédération *FOM*, celui de la simulation *SOM* et celui du modèle de management de la simulation *MOM*[33]. Le bénéfice de l'utilisation des formats de l'OMT (FOM, SOM ou éventuellement MOM) est de définir une référence commune pour décrire les modèles objets des simulations afin qu'ils

[31] *Federation Object Model*
[32] *Simulation Object Model*
[33] *Management Object Model*

soient utilisables dans toute la communauté HLA. L'OMT permet donc de décrire des modèles objets, on retrouve donc la notion de classe d'objet définie par leurs attributs ; les objets sont des instances de classes avec leur propre identité. Cependant l'OMT reste limité à certains aspects du paradigme de programmation objet [Lut97]. Par exemple, il ne permet pas de définir des comportements ou des méthodes pour les objets représentés, cela est laissé à la charge du fédéré (le simulateur gérant l'objet).

2.4.3.1 Le FOM

Le FOM présente toute l'information partagée et échangée par les fédérés au sein de cette fédération. C'est un contrat régissant les échanges de la fédération afin qu'ils soient interprétables par tous les fédérés intéressés. Il existe deux types d'objet pouvant être échangé dans une fédération HLA : les **Objets** et les **Interactions**. Les objets sont des informations considérées comme persistantes et on gardera des traces de ces échanges (par exemple un véhicule qui évolue dans la simulation). Inversement, les interactions sont considérées comme des données éphémères qui seront simplement émises et reçues sans garder de trace de ces échanges dans les fédérés (par exemple une collision entre deux véhicules). L'information contenue dans le FOM d'une fédération est regroupée dans un fichier texte nommé le fichier FED[34] pour la norme US DoD 1.3 [DoD99] ou FDD[35] pour la norme IEEE 1516. Ce fichier sera utilisé par la RTI lors de l'exécution d'une simulation.

2.4.3.2 Le SOM et le MOM

Le SOM de HLA est une spécification des capacités intrinsèques du fédéré (les informations internes que pourra fournir le fédéré) fournies aux fédérations HLA. Le SOM peut être considéré comme une fiche signalétique du fédéré qui favorise la réutilisation d'un fédéré. Ce modèle objet est principalement utilisé lors des procédures de validation et de certification des fédérations HLA. La MOM possède une définition plus globale mais il reste peu utilisé par les concepteurs HLA. Il est défini dans les documents de spécifications d'interface [DoD98c, IEE00d] et non dans la description des modèles OMT. Il permet d'identifier les objets et les interactions qui permettent de connaître l'état et l'évolution d'une fédération. C'est en quelque sorte une bibliothèque de définitions qui permet la collecte d'informations sur l'état de la fédération et la connaissance de ces états au travers d'interactions. Dans la suite, nous ne nous intéresserons plus à ces deux types de modèle objet.

2.4.3.3 Les types de transport

Les modes de transport disponibles dans la norme HLA pour les objets décrits dans le FOM sont les modes *reliable* et *best-effort*. Le mode reliable permet de garantir que la valeur envoyée (pour la mise à jour de l'attribut d'un objet par exemple) sera toujours reçue. Ce mode de transport est implémenté en utilisant le protocole réseau TCP/IP. Inversement, l'utilisation du mode best-effort ne permet pas de garantir la réception. Il est codé avec le protocole UDP/IP et il est plus rapide car il n'y a aucun mécanisme d'acquittement. De plus, le concepteur peut aussi spécifier si ses objets seront transmis en utilisant les mécanismes de gestion du temps ou non (*C.f.* partie 2.5, page 31).

[34] *Federation Execution Model*
[35] *FOM Document Data*

2.4.4 Les spécifications d'interface

Les spécifications d'interface [DoD98c, IEE00d] définissent un ensemble de services permettant l'échange d'information entre les fédérés. Ces informations ne peuvent pas transiter directement d'un fédéré à un autre mais doivent obligatoirement passer par l'intergiciel RTI (*C.f.* règle 3). Il existe deux grandes catégories de services :

- Les services à l'initiative des fédérés (appelés services et notés en bleu dans la suite) sont les services fournis par la RTI que peuvent appeler les fédérés lorsqu'ils souhaitent accéder à une fonctionalité.
- Les services à l'initiative du RTI (appelés callbacks et notés en vert dans la suite) sont les services fournis par les fédérés auxquels peut accéder la RTI. Il faut donc noter que la réalisation d'un fédéré HLA implique de mettre en œuvre ces callbacks dans l'implémentation du fédéré.

Indépendamment de ces spécificités, les services sont classés en quatre familles indispensables de mécanismes, permettant (1) la gestion de la fédération, (2) la gestion des déclarations, (3) la gestion des objets, (4) la gestion du temps. A ces familles s'ajoutent deux familles de services optionnels d'optimisation permettant (5) la gestion de la propriété et enfin (6) la gestion de la distribution des données.

2.4.4.1 Gestion de la fédération

La famille des services de gestion de la fédération regroupe les services et callbacks permettant la création et la participation à une fédération. Inversement, cette famille permet le retrait d'un fédéré participant à une fédération ou encore la destruction de la fédération. Cette famille de services permet aussi la mise en place de points de synchronisation entre fédérés de la même fédération. Dans HLA, la gestion de la fédération se fait donc au niveau des fédérés. Il n'y a pas de phase d'initialisation de la fédération destinée à attendre que tous les fédérés qui sont censés y participer soient bien présents. Il n'y a pas non plus de phase de clôture de la fédération assurant que tous les fédérés d'une fédération aient bien quitté la fédération avant de la détruire. Ainsi HLA permet aux fédérés de rejoindre l'exécution d'une fédération et de la quitter à n'importe quel moment.

Nom du service/callback	Explications
`createFederationExecution()`	Création d'une fédération donnée
`joinFederationExecution()`	Participation à une fédération précédemment créée
`resignFederationExecution()`	Retrait d'un fédéré participant à une fédération
`destroyFederationExecution()`	Destruction d'une fédération.
`registerFederationSynchronizationPoint()`	Mise en place de points de synchronisation entre fédérés

TAB. 2.3 – Exemple de services pour la famille *Gestion de fédération*

2.4.4.2 Gestion des déclarations

Contrairement au modèle de communication par messages entre clients et serveurs, le modèle de communication HLA est basé sur des principes de publication et d'abonnement qui permettent d'effectuer un routage implicite des données. Chaque fédéré peut ainsi déclarer ses intentions de générer ou de recevoir de l'information. Pour cela, il exprime son intérêt pour publier ou recevoir les valeurs correspondantes à un ou plusieurs attributs d'une classe d'objet ou à tous les paramètres d'une classe d'interaction. Inversement, un fédéré peut exprimer son intention de ne plus publier, ni souscrire à une classe d'objet ou d'interaction. Au niveau de la gestion des déclarations, la différence fondamentale entre les deux supports de communication HLA (objet et interaction) est la suivante :

- un fédéré peut publier ou souscrire uniquement pour un sous-ensemble des attributs d'une classe d'objet ;
- en revanche, la publication ou la souscription à une classe d'interaction concerne obligatoirement tous les paramètres qui la composent.

Nom du service/callback	Explications
`publishObjectClassAttributes()`	Intention de **générer** de l'information pour les attributs d'un objet
`publishInteractionClass()`	Intention de **générer** de l'information pour une interaction
`subscribeObjectClassAttributes()`	Intention de **recevoir** de l'information pour les attributs un objet
`subscribeInteractionClass()`	Intention de **recevoir** de l'information pour une interaction
`unpublishObjectClassAttributes()`	Intention de ne plus **générer** de l'information pour les attributs d'un objet
`unpublishInteractionClass()`	Intention de ne plus **générer** de l'information pour une interaction

TAB. 2.4 – Exemple de services pour la famille *Gestion des Déclarations*

2.4.4.3 Gestion des objets

Cette famille de services sert principalement à gérer la création (ou la destruction) d'instances des classes d'objets (et des classes d'interactions) mises à jour. La gestion des objets comprend également les services d'envoi et de réception de valeurs pour les objets et les interactions. Les services et callbacks pour la gestion des objets permettent :

1. D'une part, à un fédéré d'enregistrer, de créer, de mettre à jour et supprimer une instance d'objet d'une classe d'objet qu'il publie ;
2. D'autre part, à un fédéré souscrivant à une classe d'objet d'être informé de la création, des valeurs ou de la destruction d'une instance de cette classe par un autre fédéré.

Il n'existe pas de fonctionnalité équivalente pour les instances d'interactions. En effet, comme nous l'avons vu précédemment (*C.f.* partie 2.4.3.1, page 25), HLA considère les objets comme des entités persistantes dans la simulation et la RTI garde une trace de toute instance d'objet créée afin d'informer tout fédéré rejoignant la fédération (et souscrivant à la classe de l'instance) de

l'existence de cette instance (mais pas de ses valeurs). En revanche, HLA considère les interactions comme des événements fugaces en conséquence la RTI ne garde aucune trace des interactions envoyées et l'envoi d'une interaction ne sera répercutée que pour les fédérés présents dans la fédération au moment de l'envoi (et souscrivant à la classe de l'interaction).

Nom du service/callback	Explications
registerObjectInstance()	Création d'une instance de classe d'objet que le fédéré publie
discoverObjectInstance()	Information au fédéré souscrivant de la création d'une instance de classe objet
deleteObjectInstance()	Suppression d'une instance d'objet enregistrée
removeObjectInstance()	Information au fédéré souscrivant de la destruction d'une instance de classe objet
updateAttributeValues()	Mise à jour des valeurs d'attributs d'une instance
reflectAttributeValues()	Information au fédéré souscrivant des modifications des valeurs d'une instance

TAB. 2.5 – Exemple de services pour la famille *Gestion des Objets*

2.4.4.4 Gestion du temps

L'objectif de la gestion du temps est de contrôler les instants où les différents fédérés effectuent des actions afin de déterminer un ordre et éventuellement le respect de la causalité. HLA supporte plusieurs politiques de gestion du temps qui influeront sur la manière dont les fédérés avanceront dans le temps simulé ainsi que sur leurs capacités respectives à produire et à recevoir des messages horodatés (message possédant une estampille temporelle du temps simulé). Ainsi les services et callbacks de l'API HLA permettent à un fédéré de fixer ou de modifier sa propre politique de gestion du temps. Les principes de base des mécanismes pour la gestion du temps dans HLA seront présentés plus précisément dans la suite (*C.f.* partie 2.5, page 31).

Nom du service/callback	Explications
enableTimeRegulation()	Demande pour être un fédéré régulateur pour l'avancée du temps
enableTimeConstrained()	Demande pour être un fédéré contraint pour l'avancée du temps
disableTimeRegulation()	Demande pour ne plus être un fédéré régulateur pour l'avancée du temps
disableTimeConstrained()	Demande pour ne plus être un fédéré contraint pour l'avancée du temps
timeAdvanceRequest()	Demande pour avancer dans le temps
timeAdvanceGrant()	Acceptation de l'avance dans le temps

TAB. 2.6 – Exemple de services pour la famille *Gestion du Temps*

2.4.4.5 Gestion de la propriété

Les fonctionnalités pour la gestion de la propriété permettent l'échange, l'acquisition et la cession de la propriété d'un attribut d'objet publié par le fédéré (il faut que le fédéré déclare la publication des attributs qu'il souhaite échanger, acquérir ou encore céder). La propriété donne le droit de mise à jour et HLA garantit que, tout au long de l'exécution d'une fédération, un attribut d'une instance d'objet sera possédé par au plus un fédéré. Il existe de plus, dans toutes les classes d'objets HLA, un attribut particulier `privilegeToDelete` (inclus par défaut dans toute déclaration de publication ou souscription) qui permet au fédéré qui le possède pour une instance d'objet donnée de supprimer cette instance. Par défaut, un fédéré créant une instance d'objet possède cet attribut ainsi que les attributs des classes que le fédéré a déclaré publier. Le `privilegeToDelete` fonctionne comme tout autre attribut et il est possible d'échanger, acquérir et céder sa propriété de la même façon que n'importe quel autre attribut d'instance.

Nom du service/callback	Explications
`attributeOwnershipAcquisitionIfAvailable()`	Demande d'acquisition de la possession d'attributs possédés par aucun fédéré
`attributeOwnershipAcquisition()`	Demande de négociation pour l'acquisition d'attributs avec les fédérés qui les possèdent
`cancelAttributeOwnershipAcquisition()`	Demande d'annulation pour une acquisition négociée en cours
`unconditionnalAttributeOwnershipDivestiture()`	Demande de cession de la propriété d'attributs que le fédéré possède

TAB. 2.7 – Exemple de services pour la famille *Gestion de la Propriété*

2.4.4.6 Gestion de la distribution des données

Il est possible d'affiner les échanges des données entre fédérés en définissant des régions d'influence. Ces services permettent d'affiner les déclarations faites à l'aide des services de la gestion des déclarations. En effet, dans la gestion des déclarations, le routage des messages est effectué par la RTI qui filtre les messages en envoyant les mises à jour d'objet ou les interactions uniquement aux fédérés concernés selon leurs intentions de publication et de souscription. Les services de gestion de la distribution des données permettent d'étendre ce principe de filtrage en définissant des espaces de routage (ou zones) [MS97]. Ces services étendent le filtrage en n'émettant les messages qu'aux fédérés se trouvant dans la même zone et permettent ainsi d'optimiser les échanges de données au sein de la fédération. Très tôt, un ensemble de scénarios pour évaluer la nécessité de la distribution de données a été créé [CK97]. Ces mécanismes sont particulièrement utiles dans les fédérations grande échelle [HWTC01]. Cette classe de services est optionnelle et peut ne pas être pris en compte dans une simulation.

2.4.4.7 Une simulation type

Les différents services proposés par la norme permettent à un ensemble de fédérés de dialoguer de façon cohérente au sein d'une même fédération lors d'une simulation ainsi que d'en gérer le déroulement (*C.f.* tableau 2.8). Par exemple, les services de gestion de la fédération assurent les phases de création et de suppression d'une fédération. Les services de gestion des déclarations, des

objets et du temps sont utilisés dans les phases d'initialisation et de terminaison ainsi que celle de la boucle de simulation. Les services de gestion de la propriété et de gestion de la distribution des données sont des services d'optimisation optionnels. Un exemple très complet de ces différentes phases de vie d'une fédération HLA (ainsi que les détails techniques de l'implémentation) est disponible dans le chapitre 3 de la thèse de Jérôme Latour [Lat05].

1. Création
⇒ Création de la Fédération par l'un des fédérés mis en jeu ⇒ Adhésion à la fédération du fédéré créateur pour tous les fédérés
2. Initialisation
⇒ Déclaration des intentions de publication et de souscription de chaque fédéré ⇒ Mise en place de la politique de gestion du temps ⇒ Phase de synchronisation initiale entre les fédérés ⇒ Enregistrement des objets simulés
3. Boucle de simulation
⇒ Avance dans le temps (pour chaque fédéré) ⇒ Calculs locaux (pour chaque fédéré) ⇒ Envoi de mises à jour pour les entités simulées suivant le modèle de comportement
4. Terminaison
⇒ Suppression des objets enregistrés ⇒ Désactivation de la politique de gestion du temps
5. Suppression
⇒ Tous les fédérés quittent la fédération ⇒ Destruction de la fédération par le fédéré créateur

TAB. 2.8 – Schéma d'exécution d'une simulation HLA

2.4.5 La RTI : l'intergiciel HLA

La RTI est le logiciel permettant l'exécution d'une simulation globale selon les principes de la norme HLA. C'est un intergiciel (ou middleware) donc un intermédiaire entre les différentes applications (les simulateurs) et les couches basses de communication (protocole réseau et système d'exploitation). Comme nous l'avons expliqué précédemment, il fournit différents services aux applications selon les spécifications de la norme implémentées. C'est le composant clé d'une fédération HLA car son implémentation et ses algorithmes sont déterminants pour la qualité et la performance de la simulation globale. Le premier intergiciel HLA disponible et majoritairement utilisé à la naissance de la norme HLA était la RTI du DMSO nommé : RTI-NG [HNB97, BFMT98]. Les fichiers binaires de cette RTI sont toujours disponibles [64] et ne sont exécutables

que sur d'anciens systèmes d'exploitation tels que Windows NT ou des vieux noyaux Linux. Depuis, l'utilisation du standard a amené l'implémentation de différentes RTIs (commerciaux ou libres) qui diffèrent par la version de la norme HLA supportée, le support au langage de programmation ou encore la plate-forme d'exécution (le système d'exploitation). Le tableau 2.9 donne une liste (non exhaustive) des RTIs disponibles actuellement.

Nom du logiciel	Distributeur	Normes respectées	Licence	Languages supportés	Références
`Chronos RTI`	Magnetar Games	1516	Commerciale	`C++` .Net	[11]
`MÄK RTI`	MÄK Technologies	1.3 1516	Commerciale	`C++` Java	[42] [WG01]
`Pitch RTI`	Pitch Technologies	1.3 1516	Commerciale	`C++` Java	[47] [KO01]
`PoRTIco`	Littlebluefrog labs	1.3, 1516 1516	CDDL[36]	`C++` Java	[48]
`EODiSP HLA`	P&P Software	1516 (partiel)	GPL[37]	Java	[18] [KO06]
`yaRTI`[38]	Dominique Canazzi	1.3	GPL	Ada	[88] [Can99]
`CERTI`	ONERA	1.3 1516 (partiel)	GPL, LGPL[39]	`C++`, `F90`, Java Matlab, Python	[9] [BS02, NRS09]
`STOW RTI`	MIT Lincoln Laboratory	1.3	Commerciale	`C++`	[63] [CCM+97]
`RTI KIT, FDK`[40]	PADS[41]	1.3 1516	GPL, LGPL	`C++`	[63, 20] [MF01]

TAB. 2.9 – Liste des RTIs disponibles

De nombreuses autres implémentations existent et sont parfois créées uniquement pour répondre à des besoins spécifiques. Ces intergiciels ne sont alors pas disponibles ni utilisables en dehors de leur contexte universitaire, militaire ou industriel. Nous pouvons citer comme exemple UK-RTI [CM97] de l'armée anglaise, le SMS-RTI [JH05] de l'institut des sciences et de technologie de Corée ou encore le StarLink+ du département de la défense de Chine [LYTW06]. Les comparaisons entre ces différentes implémentations de RTIs sont difficiles. Cela nécessite d'obtenir les licences des RTIs lorsqu'ils sont commerciaux. De plus, les fédérations s'exécutant parfaitement sur une RTI ne sont pas assurées de fonctionner avec une autre RTI (par exemple pour un problème de compatibilité des langages de programmation). Néanmoins, certains travaux ont proposé des évaluations de performances entre différentes RTIs sur des benchmarks spécifiques [KCL01, MlR09].

2.5 Les mécanismes de gestion du temps

2.5.1 La mise en œuvre

Les mécanismes de gestion du temps offerts par HLA et hérités de ALSP constituent un des atouts majeurs de ce standard de simulation. Les principes de base de ces mécanismes sont fondés sur des travaux et algorithmes issus de la communauté de la simulation à événements discrets (*C.f.* partie 2.2.3, page 15). Le premier fondement des mécanismes de gestion du temps est qu'il n'existe aucune référence à une horloge partagée par une fédération. Chaque fédéré doit avoir sa

propre référence temporelle (appelé temps logique du fédéré). De ce fait, toute correspondance entre ce temps logique et un autre référentiel temporel (comme par exemple une horloge temps réel) est à la charge du fédéré [Fuj98].

Cette échelle de temps logique (ou temps simulé) est partagée par tous les fédérés afin de déterminer leur temps local (l'instant du temps simulé dans lequel le fédéré se trouve) et le temps des événements qu'ils produisent (la date de validité de l'événement). En ce sens, la norme HLA distingue deux types de messages selon qu'ils soient ou non estampillés dans ce temps logique. Les messages peuvent être non horodatés dans le temps logique (de type RO[42]). Dans ce cas, ils ne sont pas datés dans le temps de simulation et sont simplement envoyés et reçus sans vérification d'une cohérence de leur validité par rapport au temps simulé. Ils sont donc reçus dans l'ordre dans lequel le réseau et la RTI peut les délivrer (*C.f.* figure 2.5).

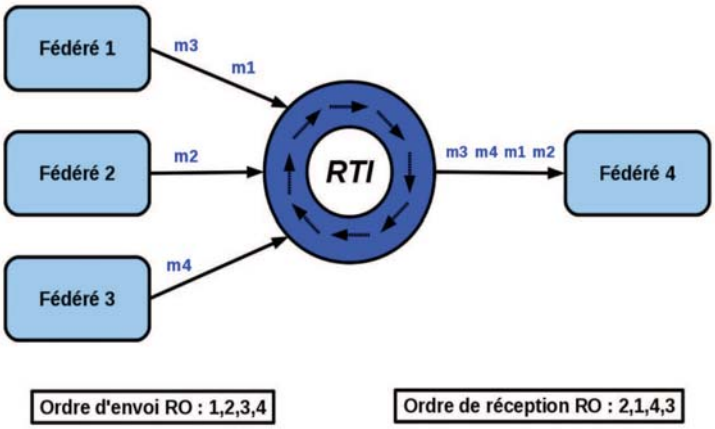


Fig. 2.5 – Transmission des messages RO

Inversement, les messages peuvent être horodatés dans le temps logique (estampillé avec une valeur de ce temps logique), ces messages sont appelés les messages TSO[43]. Dans ce cas, la RTI devra s'assurer que ces messages sont délivrés en concordance avec les mécanismes de synchronisation utilisés par les simulateurs émetteurs ou récepteurs de ces messages. Ils seront délivrés dans l'ordre de leur estampille de temps logique (*C.f.* figure 2.6).

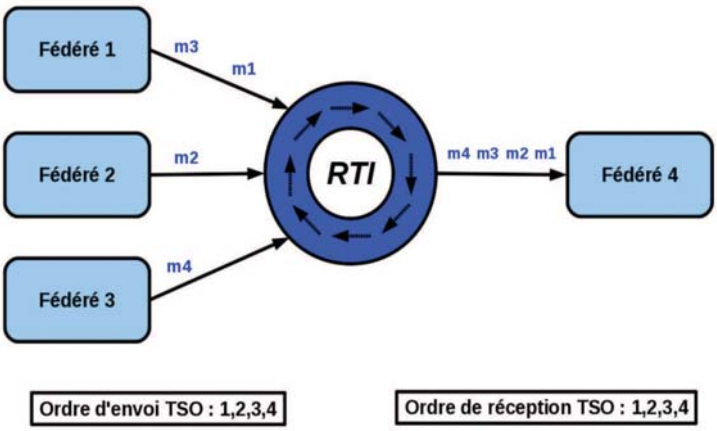


Fig. 2.6 – Transmission des messages TSO

[42] *Receive Order*
[43] *Time Stamped Order*

Les fédérés émetteurs de messages TSO sont dit régulateurs et les fédérés qui reçoivent ces messages sont dits contraints dans le temps. Plus précisément, les fédérés régulateurs, par envoi de message TSO, conditionnent l'avancement dans le temps des fédérés contraints. Un fédéré peut aussi être régulateur et contraint c'est-à-dire qu'il régule l'avance dans le temps de tous les autres fédérés contraints mais sa propre avancée dans le temps dépend de celle de tous les autres fédérés régulateurs. Inversement, un fédéré ne recevant et ne produisant aucun message TSO est dit ni régulateur ni contraint et, dans ce cas, il régule seul son avance dans le temps auquel cas son horloge logique n'a pas une grande signification. Le rôle d'un fédéré dans la gestion du temps est donc déclaré de sa propre initiative. Normalement, conformément à l'esprit HLA, tout fédéré peut changer son rôle aussi souvent qu'il le souhaite au cours de l'exécution d'une fédération. Néanmoins les opérations de changement de statut, telles que le passage de non-régulateur à régulateur, sont délicates à mettre en œuvre.

2.5.2 Les différents mécanismes

Le standard HLA supporte trois mécanismes d'avance dans le temps pour les fédérés qui participent à une fédération. Les deux premiers mécanismes reposent sur les principes des synchronisations pessimistes (*C.f.* partie 2.2.3.1, page 15) et le troisième sur les principes de l'avancée optimiste (*C.f.* partie 2.2.3.2, page 16). Les détails de l'implémentation sont laissés au choix du concepteur de la RTI comme nous le verrons dans la suite (*C.f.* partie 2.5.3, page 34).

Le premier mécanisme est celui de l'avance par pas de temps et les fédérés ainsi synchronisés sont dits *Time Stepped*. Dans ce cas, les fédérés demandent explicitement une avance à l'instant t du temps simulé en appelant le service `timeAdvanceRequest(`t`)`. Cela indique qu'ils ont traité l'émission de messages avec une estampille inférieure à ce temps t. En retour, la RTI accordera cette avance par la callback `timeAdvanceGrant(`t`)` lorsque l'avancement des autres fédérés participant à la fédération sera sûr de garantir la causalité.

Le second mécanisme est celui de l'avance sur occurrence d'événement et les fédérés ainsi synchronisés sont dits *Event Driven*. Ici, l'avancement dans le temps est conditionné par la réception des messages TSO. Les fédérés demandent de recevoir le prochain événement disponible entre son temps logique et le temps t (et sûr du point de vue de la causalité) en appelant le service `nextEventRequest(`t`)` pour la norme 1.3 ou `nextMessageRequest(`t`)` pour la norme 1516. En retour, la RTI lui délivrera le premier message disponible (sûr du point de vue de la causalité) et accordera une avancée dans le temps à l'estampille t' de ce message par la callback `timeAdvanceGrant(`t'`)` (avec $t' \leq t$).

Le dernier mécanisme est celui de l'avancée optimiste et il est beaucoup moins utilisé dans le monde de la simulation HLA. Dans ce cas, chaque fédéré avance dans le temps à sa propre vitesse indépendamment des autres. En appelant le service `flushQueueRequest(`t`)`, il va recevoir tout les messages RO disponibles ainsi que tous les messages TSO avec une estampille inférieure à t. Ensuite la RTI accordera l'avance au temps t par la callback `timeAdvanceGrant(`t`)`. Cependant, la RTI ne peut garantir qu'il n'y aura aucune nouvelle occurrence d'événement dans le passé du fédéré et la détection d'un état de violation de la causalité est laissé à la charge du fédéré optimiste. De telles situations impliquent que ce type de fédéré doit invoquer des services d'annulation `retract()` (permettant un retour arrière ou *rollbacks*) afin de revenir sur un état de simulation cohérent (c'est-à-dire un état respectant la causalité). L'appel à ce service `retract()` par un fédéré peut ensuite être propagé par la RTI aux autres fédérés avec l'utilisation de la callback `requestRetraction()` en fonction des événements incorrects transmis et consommés par les fédérés.

Un des atouts du standard HLA est de permettre à plusieurs fédérés utilisant des politiques de gestion du temps différentes de participer et de coopérer au sein de la même simulation globale. Cela constitue un avantage évident pour la réutilisation et l'interopérabilité dans les simulations HLA [IS01].

2.5.3 Les différentes implémentations

Comme nous l'avons expliqué précédemment, les mécanismes de gestion du temps reposent sur les principes élaborés par la communauté de la simulation à événements discrets. Les mécanismes d'avancée par pas de temps et sur occurrence d'événement reposent sur les méthodes de synchronisation pessimistes[44]. Les mécanismes d'avancée optimiste reposent sur les méthodes de synchronisation du même nom. Le choix des algorithmes et des méthodes d'implémentation afin d'assurer le fonctionnement correct de ces mécanismes est laissé à la charge des concepteurs de RTI. Certaines implémentations reposent sur l'algorithme du NULL message tel que CERTI (la RTI présentée dans la partie 2.6) ou la RTI Version F.0 [CFWW97] (qui n'est plus maintenue) ou la RTI STM[45] [HTHL05] construite au dessus de la dernière version de la RTI-NG. Cela repose sur un contrat de chaque fédéré appelé le *lookahead* pour lequel chaque fédéré s'engage à ne pas diffuser des messages de simulation avec une estampille temporelle inférieure à son temps local plus ce *lookahead*. D'autres implémentations reposent sur des algorithmes de détermination d'un état global dans lequel la RTI sert de processus central capable de déterminer l'état global de la fédération. Par exemple, la SMS-RTI est basée sur une variante de l'algorithme de Mattern [HKK07]. Les méthodes d'implémentation pour les fédérés optimistes ont aussi fait l'objet de différents travaux. Ils sont tous basés sur la méthode du Time Warp de Jefferson et se distinguent par les différentes techniques de mise en œuvre des gestionnaires de retour arrière (*rollback*) en cas de faute temporelle [VM00, WTY+04]. En revanche, nous pouvons noter que les détails techniques des implémentations et des mises en œuvre des mécanismes de gestion du temps dans une RTI ne sont, bien souvent, pas décrits et plus particulièrement pour les RTIs commerciaux. Dans ce cas, les concepteurs se limitent à garantir la sémantique des services tels qu'ils sont définis dans la norme HLA.

La base commune pour implémenter les services conservatifs de gestion du temps dans la RTI est le calcul du *LBTS*[46] pour chaque fédéré (appellation norme 1.3), ou encore noté GALT[47] pour la norme 1516. Le LBTS (ou GALT) est la date minimale possible à laquelle le fédéré peut recevoir un événement TSO. Cette valeur est déterminée en regardant le message le plus proche dans le temps qui peut être généré par les fédérés régulateurs et un fédéré contraint ne peut aller au delà de cette date. En ce sens, la RTI doit identifier les fédérés qui doivent participer au calcul du LBTS et ceux qui exigent le résultat du calcul du LBTS. Les fédérés régulateurs génèrent les messages TSO donc ils doivent participer au calcul des LBTS. Les fédérés contraints reçoivent les messages TSO donc ils exigent les résultats du calcul des LBTS. Dans le cas des services optimistes le fédéré doit être capable de calculer une borne inférieure sur le temps logique de tout retour en arrière. Cette borne inférieure est le GVT défini par Jefferson et permet aux fédérés d'annuler les opérations incohérentes temporellement (calculs et messages). Dans ce cas, la valeur de LBTS fournit les informations nécessaires pour permettre au fédéré de calculer son GVT et enclencher les mécanismes de libération mémoire des états conservés.

[44] *Il faut faire attention de ne pas confondre ces deux mécanismes avec les principes de simulation dirigée par le temps et la simulation dirigée par événement*
[45] *Smart Time Management*
[46] *Lower Bound Time Stamp*
[47] *Greatest Avalaible logical Time*

2.6 Le CERTI : La RTI de l'ONERA

2.6.1 Histoire

Afin d'effectuer des simulations distribuées, de mieux cerner les principes et le fonctionnement de la norme HLA et de pouvoir gérer et optimiser son propre outil, l'ONERA s'est tourné vers ce standard et cela s'est concrétisé par le développement d'un prototype de RTI dès 1997 [Des97, Sir98] nommé le CERTI, acronyme de CERT-RTI[48]. Le CERTI a été développé en `C++` à l'ONERA-Centre de Toulouse et il est toujours maintenu et amélioré par une communauté active [9]. En effet, le CERTI est, depuis 2002, un outil Open-source [BS02, NRS09] sous licence GPL [25] et LGPL [35].

2.6.2 Architecture

L'architecture du CERTI est basée sur une architecture de processus qui s'échangent des messages en utilisant des sockets de communications. L'architecture est divisée en trois composants principaux qui rappellent l'architecture des logiciels du standard ALSP (*C.f.* figure 2.7) : la libRTI, le RTIA[49] et le RTIG[50]. Ces trois composants assureront la communication et l'exécution des fédérés. Le fédéré reste, selon les principes HLA, un processus dont la conception est laissée à la charge de l'utilisateur du CERTI.

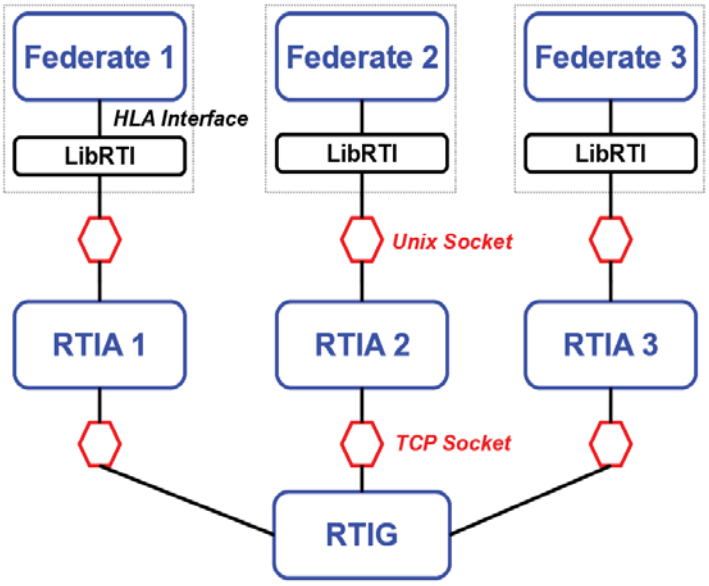


FIG. 2.7 – Architecture du CERTI mettant en jeu trois fédérés

La librairie libRTI est l'interface entre le fédéré et le reste du RTI. C'est un ensemble de fonctions (ou méthodes) `C++` dont la syntaxe est définie par la norme HLA (US DoD 1.3 et IEEE 1516). Ces fonctions sont regroupées dans une classe `RTIAmbassador`, qui encapsule en fait les envois de messages vers le processus RTIA. La libRTI est complètement incluse dans le processus Fédéré qui comprend également le code de la réception des messages (classe `FedAmbassador`).

[48] *Centre d'Etudes et de Recherche de Toulouse - Run Time Infrastructure*
[49] *RTI Ambassador*
[50] *RTI Gateway*

Le RTIA est un processus fonctionnant, généralement, localement sur le même hôte que le fédéré (il est en effet possible de remplacer la socket Unix par une socket TCP/IP et ainsi de distribuer le RTIA sur une autre machine). Il reçoit les requêtes transmises par la librairie via la socket UNIX et les réponses transmises par le RTIG via les sockets TCP/IP et/ou UDP/IP. On peut le comparer au LRC[51] souvent mis en œuvre dans la terminologie HLA pour différents RTI. Le RTIA conserve un certain nombre d'informations localement comme le modèle objet de la simulation. Le modèle objet conservé par le RTIA est identique à celui du RTIG. De cette façon certaines requêtes peuvent être traitées localement par le RTIA sans avoir recours à des communications avec le RTIG. Cette méthode simplifie le nombre de requêtes et accélère le traitement en évitant des échanges réseaux superflus. Toutes les autres requêtes plus complexes doivent être transmises au RTIG.

Le RTIG est le serveur central que l'on peut identifier comme le CRC[52] souvent mis en œuvre dans la terminologie HLA pour différentes RTI. D'une part, il permet l'échange des données entre les différents fédérés en dialoguant avec les RTIAs correspondants. D'autre part, son architecture centralisée permet une implémentation simple de certains services du RTI. Tous les échanges effectués entre les fédérés d'une fédération passent par le RTIG qui filtre les différents messages puis les redirige de manière appropriée. Par exemple, un appel à `updateAttributeValues()` transmis par le RTIA d'un fédéré sera retransmis à tous les fédérés ayant souscrit à l'attribut mis à jour.

Ces trois composants s'échangent des données via des sockets de trois types. Les sockets Unix sont dédiées aux échanges RTIA ⇔ Fédéré (sous Unix/Linux). Les sockets UDP/IP sont dédiés aux échanges RTIA ⇔ RTIG (sous Unix/Linux/Windows et si le choix du type de transmission du message de simulation est `BEST EFFORT`). Enfin, les sockets TCP/IP sont pour les échanges RTIA ⇔ RTIG (sous Unix/Linux/Windows et si le choix du type de transmission du message de simulation est `RELIABLE`). Les sockets TCP/IP sont aussi implémentés pour les échanges RTIA ⇔ Fédéré sous Windows.

2.6.3 Etudes réalisées

Le CERTI a, depuis sa création, permis de réaliser de nombreuses études dans différents domaines. Ces différentes études ont aussi permis de modifier cette RTI et de la rendre plus riche de fonctionnalités :

- Des concepts de sécurité et de niveaux de sécurité dans un middleware ont été étudiés [BCSZ98].
- Une exemple concret de la multi-résolution dans une simulation HLA a été mis en œuvre pour une simulation de combat aérien [AS01].
- Des études de performances d'une simulation HLA ont aussi été menées avec un concept intéressant de simuler le comportement d'une fédération HLA avec une autre fédération HLA [dBS02].
- Les concepts de ponts de fédération permettant de relier différentes fédérations parfois chacune avec une RTI différente [BS03a, BS03b].
- La définition et le développement d'un outil d'aide à la conception et au prototypage de simulation HLA nommé DARE-HLA [Lat05].

[51] *Local RTI Component*
[52] *Gentral RTI Component*

- Des extensions pour intégrer des communications par mémoire partagée entre les différents processus du CERTI sur des architectures multi-processeurs [ABR+04].
- La conception d'un démonstrateur pour le projet ADF[53] afin de trouver des solutions pour optimiser les flux d'avions, de passagers et de fret des zones aéroportuaires, tout en maintenant la sécurité sur un aéroport [Ade04].
- L'intégration dans une infrastructure d'évaluation des impacts acoustiques et chimiques du transport aérien (projet IESTA[54]) [AORC08, AOC08, AOC09].

Le travail de cette thèse sera donc de fournir des éléments de réponse afin de permettre la conception, l'exécution et la validation de simulations distribuées de systèmes embarqués avec le CERTI. Le CERTI servira d'architecture de support afin de concevoir une *architecture de simulation distribuée temps réel.*

[53] *Aéroport Du Futur*
[54] *Infrastructure d'Evaluation et de Simulation du Transport Aérien*

Chapitre 3

L'utilisation des intergiciels dans le contexte du temps réel

Préambule

Dans le chapitre précédent, nous avons présenté la simulation distribuée, la norme HLA et ses principes. Ce chapitre traite plus précisément la problématique de l'utilisation de la norme HLA dans le contexte des systèmes temps réel. Pour cela, nous introduisons dans un premier temps les principes de base des systèmes temps réel distribués. Ensuite, nous présentons les différents standards pour les intergiciels temps réel qui existent, les spécifications et les techniques qu'ils mettent en œuvre afin d'assurer un comportement temporel prédictible pour les applications distribuées et interconnectées. Bien qu'à ce jour l'utilisation de la norme HLA ne soit pas adaptée aux contraintes de déterminisme et de prédictibilité des applications temps réel, de nombreux travaux ont été menés dans la littérature. Nous présentons donc dans un troisième temps les différentes recherches pour la mise en œuvre de simulations HLA temps réel. Enfin, nous terminons ce chapitre en décrivant les différentes mises à jour proposées afin d'ajouter à la norme HLA des spécifications pour la prise en compte de paramètres de qualité de service.

3.1 Les systèmes temps réel distribués

3.1.1 Principes généraux

Comme nous l'avons précisé dans l'introduction, un système informatique est dit temps réel lorsqu'il est soumis à des contraintes temporelles. Il n'est pas nécessairement rapide mais il doit répondre rigoureusement aux contraintes temporelles imposées par l'application. Les systèmes informatiques temps réel se différencient donc des autres systèmes informatiques par la prise en compte de ces contraintes temporelles dont le respect est aussi important que l'exactitude du résultat attendu. Autrement dit, le système ne doit pas simplement délivrer des résultats exacts, il doit les délivrer dans des délais imposés par les spécifications du système [Sta88]. L'émergence des réseaux de communication ainsi que la multiplication des fonctionnalités que l'on souhaite enfouir dans les systèmes, a conduit au fil du temps à concevoir des systèmes temps réel distribués dans lesquels les logiciels, physiquement distribués sur plusieurs unités de traitement, communiquent entre eux [Sta92, Kop11].

La QoS[1] ou QdS[2] représente l'aptitude d'un service à répondre adéquatement à des exigences. Ces exigences peuvent être liées à plusieurs aspects d'un service : son accessibilité, sa disponibilité, sa fiabilité, etc. Dans le cas qui nous intéresse, la qualité de service d'un système temps réel constitue son aptitude à respecter ses contraintes temporelles. Ainsi, on distingue deux classes de systèmes suivant l'importance accordée aux contraintes temporelles et aux problèmes que peuvent causer leurs violations [27] (illustration sur la figure 3.1) :

- Les *systèmes temps réel durs* sont exclusivement constitués de traitements qui possèdent des contraintes temporelles strictes. C'est-à-dire qu'en fonctionnement nominal, tous les traitements effectués par le système doivent respecter leurs contraintes temporelles. Ces systèmes sont souvent mis en jeu lorsque les dépassements de contraintes peuvent conduire à des situations critiques, voire catastrophiques : système de surveillance de centrale nucléaire ou encore, pour le cas qui nous intéresse, les systèmes embarqués d'un aéronef.
- La classe des *systèmes temps réel souples* est moins exigeante quant au respect absolu de toutes les contraintes temporelles affectées à ses traitements. Ils peuvent souffrir d'un "taux acceptable" de fautes temporelles de la part d'une partie des traitements sans que cela ait des conséquences catastrophiques. Ces systèmes sont souvent mis en jeu dans des applications multimédias telles que la téléphonie, la vidéo, la visioconférence ou encore les jeux en réseau.

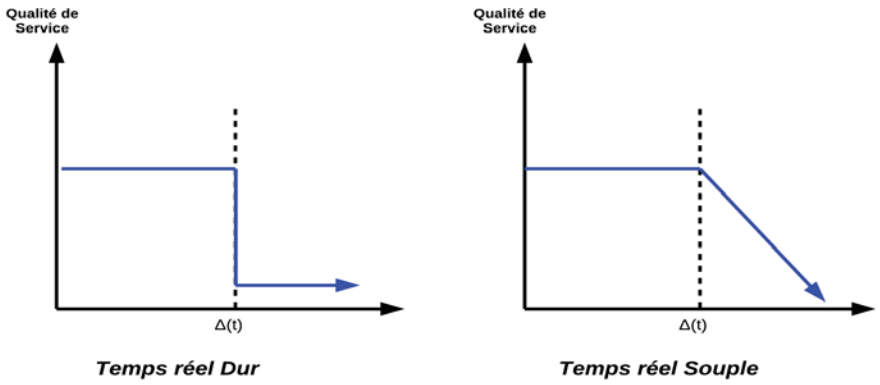


FIG. 3.1 – QoS dans les systèmes temps réel durs et souples soumis à une contrainte $\Delta(t)$

Il est important de noter qu'un système informatique temps réel n'est pas forcément plus rapide qu'un autre système informatique [HGCD00]. En effet, la confusion entre le terme *temps réel* et le terme *performance* (rapidité de calcul) est courante. Disposer d'un système rapide ne constitue pas un critère suffisant pour le qualifier de temps-réel. La performance d'un système temps réel doit être suffisante pour répondre aux besoins de l'application temps réel considérée. Ainsi, l'ordre de grandeur des constantes de temps associées à un système temps-réel est fondamental. Par exemple, dans la plupart des systèmes avec un opérateur humain dans la boucle, les constantes de temps sont issues des capacités physiologiques de perception chez l'être humain et sont de l'ordre de la centaine de millisecondes. En contrepartie, dans un système embarqué, les applicatifs temps réel doivent s'interfacer avec des équipements physiques tels que des capteurs ou des actuateurs. Dans ce cas, les constantes de temps sont beaucoup plus petites et peuvent être de l'ordre de la milliseconde. Dans ce cas, des phénomènes comme la dérive des horloges dans

[1] Quality Of Service
[2] Qualité De Service

un système distribué ne peuvent plus être négligés. Cela contraint davantage le comportement temporel du système et les études de prédictibilité et de déterminisme.

Dans un système temps réel distribué, il est essentiel que chaque composant ait un comportement déterministe. Alors, pour doter un applicatif distribué de propriétés temps-réel il faut s'assurer que toutes les couches logicielles et matérielles ont été conçues de manière à ce que la maîtrise des comportements temporels soit assurée. Ainsi, le support d'exécution tel que le choix du système d'exploitation ou du langage de programmation (*C.f.* partie 3.1.2) et le support de communication comme le choix des protocoles réseau (*C.f.* partie 3.1.3) doivent offrir des services qui garantissent des exécutions et des communications en temps borné et qui assurent le déterminisme du système distribué temps réel. De plus, des méthodes formelles spécifiques de validation sont essentielles afin de s'assurer que selon les hypothèses relatives au support d'exécution et au support de communication, l'application distribuée respectera ses échéances temporelles (*C.f.* partie 3.1.4).

3.1.2 Les supports d'exécution

Les langages de programmation sont la base d'une implémentation d'un système informatique. Ils permettent au concepteur de définir les ensembles d'instructions logiques ou mathématiques qui seront effectuées par l'ordinateur lors de l'exécution du programme. La technique la plus répandue pour programmer les systèmes informatiques est d'utiliser des langages de bas niveau tels que l'assembleur [Car05] ou le langage `C` [Ker88]. L'avènement du paradigme de programmation orienté objet a amené des nouveaux langages tels que ADA [BW98] [2, 3], `C++` [Str97] [7] ou encore JAVA [GJSB05] [31].

Néanmoins, ces langages n'étaient pas nativement destinés à la programmation de systèmes temps réel. L'émergence de la problématique temps réel a permis la définition d'interfaces de programmation (API) spécifiques permettant une gestion adaptée du système d'exploitation temps réel sous-jacent. Un effort important a donc été consacré à l'uniformisation des APIs offertes par les systèmes d'exploitation pour leur normalisation. C'est ainsi qu'est apparue la norme POSIX (pour les langages C, C++ et Ada) développée depuis la fin des années 80 par l'IEEE et normalisée par l'ANSI et l'ISO. La norme est constituée de plusieurs documents dont l'IEEE Std 1003.1b-1993 [IEE93b] qui normalise une API temps-réel et l'IEEE Std 1003.1c-1995 [IEE95c] qui constitue une extension sur les processus légers (threads). Des résumés des spécifications POSIX sont disponibles dans les documents [CLG90, Har93]. Nous pouvons aussi citer les efforts de normalisation pour le langage `Java` (RTSJ[3]) [GB00] [65] ou encore le standard OSEK[4] [OSE04, OSE05] [46] incontournable dans l'industrie automobile.

Il existe de très nombreux systèmes d'exploitation totalement ou partiellement temps-réel, le plus répandu étant VxWorks [Win08] développé par la société Wind River [82]. Ce système d'exploitation est compatible avec un grand nombre de processeurs et utilisé par la NASA pour plusieurs missions spatiales dont Mars Pathfinder et Mars Reconnaissance Orbiter. QnX [Hil92] [54] et LynxOS [Fra05] [38] sont d'autres exemples utilisés aussi bien dans l'industrie que dans le monde de la recherche ou de l'enseignement. En ce qui concerne Linux, plusieurs contributions ont permis d'améliorer le noyau pour le rendre compatible avec les exigences du temps-réel [FK03]. XENOMAI [Ger04] [87] constitue une solution Open Source fondée sur la collaboration entre le noyau Linux et un noyau temps-réel auxiliaire (basé sur la technologie Adeos [Yag02] [4]). Un système d'exploitation temps réel est aussi constitué d'un élément essentiel appelé l'ordonnanceur (*C.f.* figure 3.2) qui permet d'assurer la réalisation d'un ensemble de tâches en respectant

[3] *Real-Time Specifications for Java*
[4] *Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug (Allemand)*

les spécifications du concepteur basées sur la théorie de l'ordonnancement (*C.f.* partie 3.1.4).

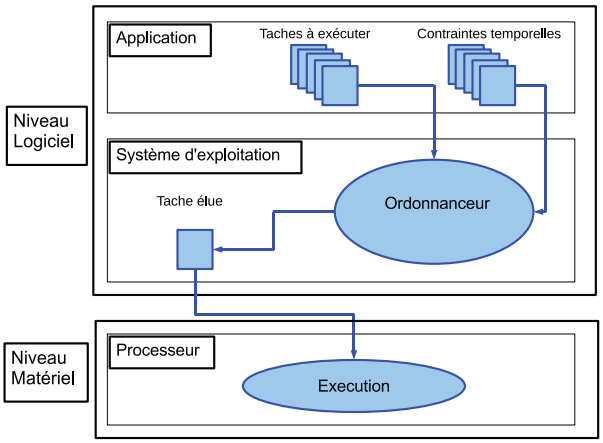


FIG. 3.2 – Illustration du rôle de l'ordonnanceur dans un système d'exploitation

Le lecteur intéressé pourra se référer aux états de l'art sur les systèmes d'exploitation temps réel [RPA02, BM05]. Une étude plus complète de l'ensemble de la problématique relative aux supports d'exécution des systèmes informatiques temps réel est disponible en annexe (*C.f.* annexe A, page 151). Néanmoins, nous insisterons ici sur un point essentiel pour l'informatique temps réel : la différence entre les threads et les processus. Les processus (dits processus lourds) sont des flots d'exécution qui ne peuvent pas, par défaut, partager leurs mémoires. Chaque processus lourd possède son propre espace d'adressage virtuel et ses propres variables. Les threads sont nés avec l'essor des machines parallèles et sont des flots d'exécution internes à un processus, ainsi l'exécution du code du processus peut être réalisée par différents fils d'exécution (*C.f.* figure 3.3). Un processus lourd peut contenir plusieurs centaines de processus légers constituant autant de flots d'exécution parallèles indépendants.

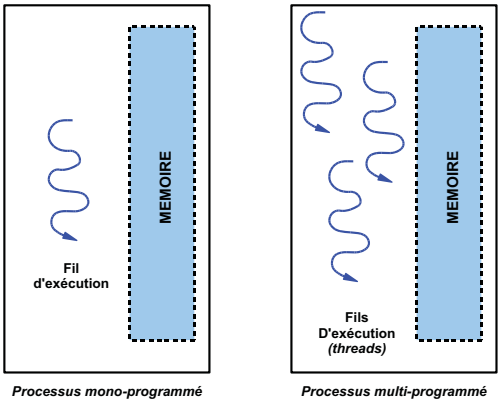


FIG. 3.3 – Processus mono-programmés et multi-programmés

Les threads proposent donc une solution performante en évitant notamment les commutations de contexte coûteuses aux processus (à cause des transferts de mémoire). En revanche, le fait qu'ils partagent la mémoire pose des problèmes supplémentaires sur la gestion des ressources communes

(situation de blocage, altération de la zone mémoire). La distinction entre ces deux concepts est importante car cela permet de définir l'unité d'exécution considérée pour le modèle formel envisagé. Dans le cas de RT CORBA nous verrons que c'est le thread qui est l'unité d'exécution envisagée (*C.f.* partie 3.2.2, page 50) ainsi que dans les approches pour la conception de RTI temps réel (*C.f.* partie 3.4.3, page 63). En revanche, comme nous l'avons vu précédemment (*C.f.* partie 2.6, page 35), l'architecture du CERTI nous impose de choisir le processus comme unité d'exécution. Nous nous affranchissons par conséquent des problèmes de gestion de ressources.

3.1.3 Les supports de communication

Les réseaux informatiques sont apparus dans les années 70-80 avec la naissance des technologies IP [CK74] et des protocoles de transport complémentaires TCP [Pos80a] et UDP [Pos80b] pour leur mise en œuvre notamment sur le réseau ARPANET [BI81]. Depuis lors, de nombreux supports physique ainsi que de nombreux protocoles réseaux sont apparus et actuellement la diversité des technologies est importante. Cette diversité a nécessité des efforts de normalisation dont la norme en vigueur est la norme OSI[5] [Bur85]. Ce modèle, basé sur le modèle SNA[6] [McF76] de IBM, décompose les technologies réseaux en 7 couches distinctes allant de l'infrastructure physique jusqu'à l'application considérée. Le modèle OSI est fondé sur le concept de piles de protocoles indépendants et souffre de sa trop forte normalisation. Généralement, les couches 5, 6 et 7 sont difficiles à distinguer pour le concepteur et donc elles sont rarement utilisées ou implémentées séparément.

Nom de la couche	Explication
1. La couche « physique »	Transmission effective des signaux sur le support physique Emission et la réception d'un ou plusieurs bit(s)
2. La couche « liaison de données »	Encodage des données pour le transport par la couche physique Détection d'erreur de transmission et synchronisation
3. La couche « réseau »	Transport, adressage et routage des paquets encodés Le protocole réseau le plus connu est IP.
4. La couche « transport »	Communication effective d'un bout à l'autre fragmentation des données en petits paquets, contrôle de transmission Les protocoles de transport les plus connus sont TCP et UDP.
5. La couche « session »	Établissement et maintient des sessions
6. La couche « présentation »	Représentation des données du codage des données applicatives Conversion entre données applicatives et octets effectivement transmis
7. La couche « application »	Point d'accès aux services réseaux logiciels : navigateur, logiciel d'email, FTP, chat...

Tab. 3.1 – Liste des couches du modèle OSI

Toutefois, bien que les techniques de réseaux assurant la distribution des applications offrent de nombreux avantages (performance, maintenance, ...), plusieurs précautions sont nécessaires. En effet, de la même façon que pour les supports d'exécution, les protocoles réseaux n'étaient pas nativement dédiés aux systèmes temps réel. Ces nouvelles contraintes ont amené des évolutions dans les technologie existantes. Un réseau offre une qualité de service donnée s'il est capable d'assurer le transport des données en respectant les exigences de l'application en termes de

[5] *Open Systems Interconnection*
[6] *Systems Network Architecture*

disponibilité, débit, délais de transmission, gigue, taux de perte de paquets. Ces paramètres de qualité de service reposent sur la couche physique choisie pour le transport (niveau 1 du modèle OSI). Cette couche physique repose sur plusieurs paramètres :

- Le support physique de transmission qui peut être des câbles dans lesquels circulent des signaux électriques tels que les technologies Ethernet ou ATM[7], des fibres optiques qui propagent des ondes lumineuses tels que la technologie FDDI[8] ou encore l'atmosphère (ou le vide spatial) où circulent des ondes radio dont un exemple bien connu est le *Wi-Fi*.
- La portée géographique du réseau qui peut aller du réseau local LAN[9] au réseau mondial WAN[10]
- L'arrangement physique, c'est-à-dire la configuration spatiale du réseau (topologie), a aussi une influence telle que la topologie en bus ou en étoile.

Au dessus de cette couche physique, la sous-couche MAC[11] (correspondant à la couche 2 du modèle OSI), est chargée de gèrer l'accès à la ressource de transmission (ressource commune à toutes les stations, et donc à toutes les tâches). Le choix de ce protocole est essentiel pour assurer le déroulement temps-réel des applications [KSY84]. Une classification des protocoles MAC temps-réel a été proposée dans [MZ95, VdC96] :

1. Une approche basée sur l'arbitrage d'accès, où l'accent est mis sur le processus d'arbitrage d'accès qui détermine quel nœud a le droit d'utiliser la ressource de transmission (d'émettre un message sur le réseau). Les protocoles de ce type les plus connus sont CAN[12] [ISO99, THW94] basé sur une notion de priorité dans les messages, FIP[13] [Cot96] basé sur un processus central de contrôle déterminant l'accès au bus ou encore IEEE 802.5 (réseau Token Ring) [FN69, IEE98] basé sur la circulation d'un jeton à priorité qui autorise le nœud qui le détient à émettre sur le support de communication.

2. Une approche basée sur le contrôle de transmission, où l'accent est mis sur le processus de contrôle de transmission qui détermine combien de temps le nœud a le droit d'utiliser la ressource de transmission. Les protocoles de ce type les plus connus sont FDDI[14] basé sur la circulation d'un jeton temporisé qui autorise le nœud qui le détient à émettre, TTP[15] [KG94] ou TDMA[16] [IEC00] basé sur la distribution d'un droit de parole par partage du temps global ou encore la variante CSMA/DCR[17] du protocole MAC classique d'Ethernet CSMA/CD[18] normalisé par l'INRIA [Le 93] basé sur la résolution des conflits au bus en temps borné par tri dichotomique.

Dans cette étude, nous nous intéresserons à la technologie Ethernet qui est la technologie réseau la plus utilisée actuellement [Spu00]. C'est un protocole probabiliste, nommé CSMA/CD [IEE00a], qui gère l'accès à la ressource physique du réseau Ethernet. Ce protocole peut sembler inadapté pour un contexte temps réel car il est non-déterministe ; cependant des techniques d'évaluation du délai probabiliste ont été proposées très tôt [Hai83]. Dans notre travail, nous souhaitons employer des composants réseaux standards qui seront intégrés à la plate-forme PRISE. Les technologies Ethernet sont maintenant très performantes et largement utilisées même

[7] *Asynchronous Transfer Mode*
[8] *Fiber Distributed Data Interconnect*
[9] *Local Area Network*
[10] *World Area Network*
[11] *Medium Access Control*
[12] *Controller Area Network*
[13] *Factory Instrumentation Protocol*
[14] *Fiber Distributed Data Interconnect*
[15] *Time Tiggered Protocol*
[16] *Time Division Multiple Access*
[17] *Carrier Sense Multiple Access/Deterministic Collision Resolution*
[18] *Carrier Sense Multiple Access with Collision Detection*

dans le domaine aéronautique avec les réseaux AFDX[19] [CES03, Cha07]. De plus, l'utilisation de commutateurs diminue les domaines de collisions et assurent des performances acceptables pour les contraintes des systèmes embarqués [Geo05] ; ainsi sur la plate forme PRISE, nous utiliserons un réseau gigabit Ethernet commuté. De plus, des protocoles additionnels assurant certaines propriétés déterministes, tels que RTP[20] [Sch03] pour l'établissement de priorités ou RSVP[21] [Bra97] pour la réservation de ressources, sont utilisables sur des supports Ethernet.

Une étude plus complète de l'ensemble de la problématique relative aux supports réseaux pour assurer les communications dans les systèmes informatiques distribués temps réel est disponible dans l'annexe B, page 161.

3.1.4 Validation et ordonnancement

Le programme applicatif est divisé en différentes entités distinctes appelées *les tâches* qui peuvent être des processus lourds ou des processus légers (threads) décrits précédemment (*C.f.* partie 3.1.2, page 41). Chacune de ces tâches a un rôle qui lui est propre (réaliser un calcul, acquérir des informations des entrées du système, ...). La caractéristique majeure des systèmes temps réel est la prise en compte de la notion de temps qui induit l'utilisation de modèles pour décrire le comportement temporel de ces tâches. Il est possible de classer les tâches mises en jeu dans un système temps réel en trois catégories (*C.f.* figure 3.4) :

1. les tâches périodiques ont des activations régulières et le délai entre deux activations successives est connu et constant.
2. les tâches sporadiques ont des activations irrégulières mais on connaît le délai minimum entre deux activations successives (appelé le *délai d'inter-arrivée* et noté DI).
3. les tâches apériodiques sont des tâches sur lesquelles nous n'avons aucune information.

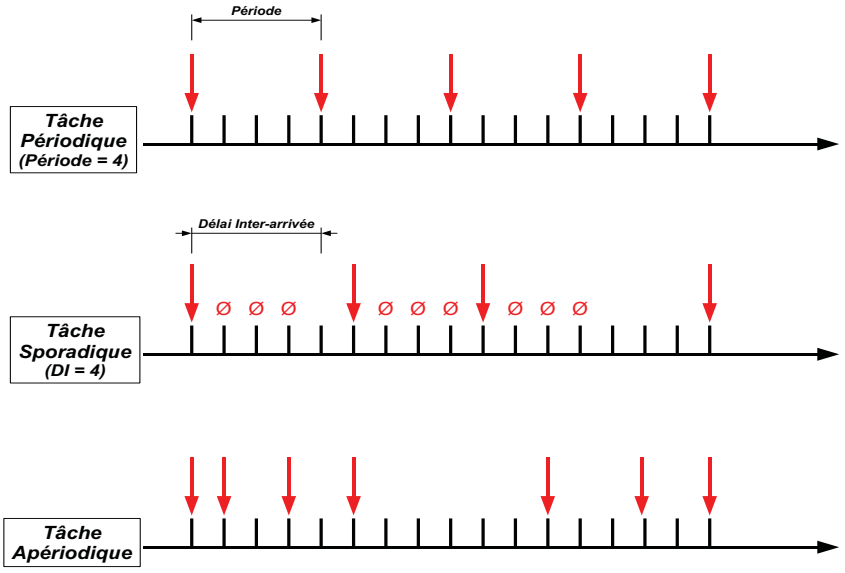


FIG. 3.4 – Illustration des différents modèles temporels de tâche

[19] *Avionics Full DupleX switched ethernet*
[20] *Real-time Transport Protocol*
[21] *Resource ReSerVation Protocol*

En ce qui concerne le mode d'exécution des tâches, on distingue deux classes qui diffèrent sur la notion d'interruptibilité. Une tâche est dite *préemptible* si, au cours de son exécution, elle peut être interrompue par une autre tâche, et reprise ultérieurement au point d'exécution précédent. Une tâche est dite *non-préemptible* si son exécution ne peut pas être interrompue par aucune autre tâche. Par la suite nous considérerons le modèle des tâches préemptibles qui est le plus utilisé pour l'exécution des tâches dans les système d'exploitation. La théorie de l'ordonnancement fournit des méthodes algébriques et algorithmiques permettant de vérifier *le comportement temporel* d'une application temps réel composée d'un ensemble de tâches. Les bases de cette théorie ont été proposées par Liu et Layland [LL73] pour la validation de systèmes mono-processeurs dans lesquels plusieurs tâches périodiques peuvent concourir pour l'obtention de l'unique processeur. Pour cela Liu et Layland ont proposé un modèle de tâche périodique basé sur les attributs temporels de ces tâches. Une tâche périodique τ_i est décrite par un quadruplet $< r_i, C_i, D_i, T_i >$ tel que :

- r_i est la date d'activation de la tâche ;
- C_i est la durée d'exécution de la tâche dans le pire des cas ;
- D_i est le délai (durée) critique à respecter pour la tâche ;
- T_i est la période (durée) de la tâche.

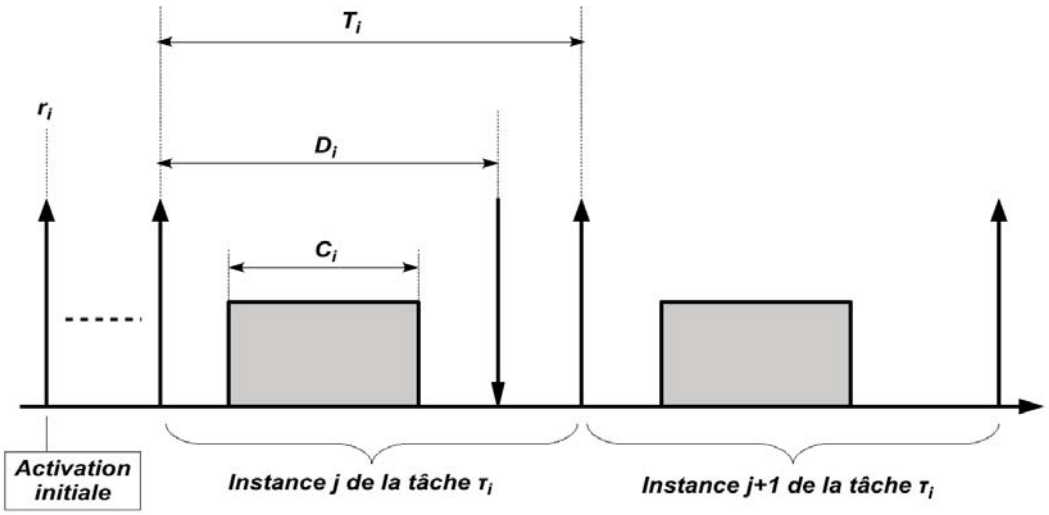


FIG. 3.5 – Illustration graphique du modèle de tâche périodique τ_i

La durée d'exécution C_i de la tâche correspond au temps processeur maximum (au pire des cas) nécessaire pour que la tâche puisse réaliser son traitement. La détermination de ce paramètre est essentielle et constitue un domaine de recherche à part entière [PS97, WEE+08, PK89, CPRS02]. Il faut également distinguer la notion d'instance de la notion de tâche. Une tâche correspond à l'entité à considérer c'est-à-dire le code exécutable qui est soumis à des contraintes temporelles. Une instance de tâche est une des occurrences de la tâche périodique c'est-à-dire une de ses exécutions périodiques. Une illustration plus explicite pour une tâche périodique τ_i et ses instances est disponible sur la figure C.4. Le modèle des tâches périodiques est le plus utilisé car c'est celui qui assure le comportement le plus prédictible. La plupart des techniques pour la prise en compte des tâches sporadiques et apériodiques visent à se ramener au cas de tâches périodiques particulières avec des techniques de traitement d'arrière-plan ou de traitement par scrutation [Spr90].

La problématique de l'ordonnancement consiste à définir une politique d'élection adéquate qui garantit le respect des contraintes temporelles pour toutes les tâches mises en concurrence

dans le système. Cette politique d'élection doit permettre de définir une séquence d'ordonnancement valide. Une séquence d'exécution est dite valide si toutes les tâches se sont exécutées dans le respect de leurs contraintes temporelles. En général, un algorithme d'ordonnancement est nécessaire pour pouvoir garantir l'ordonnançabilité d'un ensemble de tâches sur une architecture donnée (matérielle et logicielle). Un algorithme d'ordonnancement est un algorithme qui utilise les caractéristiques temporelles sur l'ensemble des tâches et détermine la séquence d'ordonnancement propice au problème. Les algorithmes d'ordonnancement sont basés sur la notion de *priorité*. Cette notion permet d'établir une relation d'importance entre les différentes tâches pour l'accès au processeur. Par conséquent, un algorithme d'ordonnancement doit, à un instant donné, choisir la tâche la plus prioritaire à exécuter par le processeur. En ce qui concerne ce choix, on distingue deux grandes classes d'algorithmes d'ordonnancement préemptif :

1. Dans **les algorithmes statiques**, les priorités des tâches sont fixes et ne changent pas pendant leur exécution. Les priorités des tâches sont calculées une seule fois avant l'exécution du système et sont basées sur des paramètres qui ne changent pas pendant l'exécution du système, où la tâche active de plus haute priorité est sélectionnée et exécutée. L'avantage dans cette classe d'algorithmes est la possibilité d'identifier, avant l'exécution du système, les éventuels problèmes d'ordonnancement tels que les tâches qui dépasseront leurs échéances temporelles. Ces algorithmes sont les plus utilisés dans le contexte des systèmes embarqués car ils ont un comportement prédictible. Les algorithmes statiques (pour le cas mono-processeur) les plus connus sont l'algorithme *Rate Monotonic* [LL73, LSD89], l'algorithme *Deadline Monotonic* [LW82] et l'algorithme de *Audsley* [Aud91].

2. **Les algorithmes dynamiques** permettent des changements de priorité pour les tâches du système. Les priorités des tâches sont calculées plusieurs fois pendant l'exécution du système et sont basées sur des paramètres qui peuvent varier pendant l'exécution. Malgré leurs propriétés intéressantes, les algorithmes dynamiques présentent quelques inconvénients dans le cadre d'applications temps réel dur car ils ont un comportement difficilement prédictible et sont moins contrôlables. Ils sont donc rarement utilisés dans le contexte des systèmes temps réel embarqués. Les algorithmes dynamiques (pour le cas mono-processeur) les plus connus sont l'algorithme *Earliest Deadline First* [Jac55, LL73] et l'algorithme *Least Laxity First* [Sor74, Mok83].

La vérification du comportement temporel des tâches est assurée par l'application de "test d'acceptabilité ou de faisabilité" associé aux algorithmes choisis. Le concepteur peut ainsi garantir le comportement temporel des tâches du systèmes temps réel. Cependant, dans le cas mono-processeur, d'autres contraintes peuvent s'ajouter aux simples contraintes temporelles. Ces nouvelles contraintes nécessitent l'utilisation de techniques additionnelles afin de garantir l'intégrité du système :

- Le partage des ressources mémoire des tâches est un problème dû à l'utilisation des threads qui partagent leur espace mémoire. Ainsi, une tâche utilisant une ressource peut bloquer momentanément une tâche plus prioritaire qui voudrait utiliser cette même ressource mémoire. Cela peut conduire à des problèmes d'inter-blocage et d'inversion de priorité. Ces problèmes sont résolus grâce à des protocoles d'allocation de ressources qui consistent à adopter des stratégies pour réduire (et borner) le temps de blocage d'une tâche en attente d'une ressource détenue par une tâche qui est moins prioritaire. Les protocoles les plus connus sont le protocole à priorité héritée (PIP[22]) [SRL90], le protocole à priorité plafond (PCP[23]) [GS88, SRL90] ou encore le protocole de gestion de la pile (SRP[24]) [Bak91].

[22] *Priority Inheritance Protocol*
[23] *Priority Ceiling Protocol*
[24] *Stack Resource Policy*

– Dans les algorithmes classiques, les tâches sont supposées sans relation de précédence. Néanmoins, le système peut nécessiter la mise en place de contraintes de précédence entre les tâches afin d'être sûr qu'une tâche donnée s'exécutera bien après la tâche qui doit la précéder (par exemple en attente d'un résultat de calcul). Dans le cas des algorithmes statiques, Blazewicz fut le premier à traiter du problème des précédences [Bla76]. Ces techniques reposent sur le principe des précédences simples qui imposent que les tâches soumises à des relations de précédence doivent avoir la même période. Ce travail a été récemment étendu à l'ONERA pour prendre en compte des précédences complexes pour des tâches temps réel de périodes différentes [FBG+10]. Pour l'ordonnancement dynamique, la prise en compte des contraintes de précédence peut se faire par modification des échéances relatives [CSB90]. E.L. Lawler proposa aussi un algorithme pour minimiser le retard maximal tout en prenant les contraintes de précédence en compte (Latest Deadline First) [Law73]. Le principe est de laisser en attente une tâche prête à être exécutée tant que ses précédences n'ont pas été validées.

Le problème de l'ordonnancement et de la validation du comportement temporel d'un système tout entier s'avère beaucoup plus complexe dans le cas distribué. L'ordonnancement proposé doit prendre en compte les différentes applications inter-connectées qui sont composées de tâches réparties sur différents sites (ou nœuds) et qui communiquent via le réseau. Cette distribution augmente la complexité de l'étude du déterminisme pour cet ensemble de systèmes inter-connectés. L'ordonnancement distribué procède par conséquent au cas par cas, et se heurte à un certain nombre de difficultés telles que le problème du placement des tâches sur les différents nœuds du système [Bac95, TBW92, CA95, RRC03a] et la prise en compte de la ressource de communication (réseau) entre les différents nœuds [SB98, MMMM01]. Il faut aussi prendre en compte la synchronisation des horloges des différents sites [Sun97] et déterminer le choix d'un ordonnancement centralisé qui centralise les informations et les décisions d'ordonnancement ou distribué qui implique plusieurs ordonnanceurs (1 par nœud) qui s'occupent de gérer des sous-ensembles identifiés des tâches du système et des messages pour le réseau.

La problématique de l'ordonnancement est un domaine de recherche actif dans lequel il existe de nombreux travaux depuis l'article de Liu et Layland. Le lecteur intéressé pourra se référer à l'état de l'art très détaillé de David Decotigny [6]. L'annexe C (page 173) présente aussi une étude plus approfondie de la théorie de l'ordonnancement, ses principes et les caractéristiques des différents algorithmes et méthodes.

3.2 Les intergiciels temps réel

3.2.1 Des évolutions vers le temps réel

Les solutions de l'informatique distribuée proposant des solutions adaptées aux applications temps réel complexes, les intergiciels ont été de plus en plus prisés afin d'interconnecter des systèmes temps réel. Cet essor a amené la création de différents projets de recherche spécifiques afin de concevoir des middlewares dédiés aux applications distribuées temps réel tels que ARMADA [ABD+97, ADF+97], MIDART [GSMT97] ou OSA+ [SPB02]. Cependant, la plupart des standards les plus connus, tels que RPC ou CORBA, n'étaient pas appropriés pour les systèmes temps-réel parce qu'ils manquaient de fonctionnalités et de spécifications pour exprimer et assurer une qualité de service aux applications distribuées. Les concepteurs des standards ont donc proposé des évolutions afin de s'adapter aux besoins spécifiques du temps réel. Par exemple, des extensions temps réel ont été envisagées pour le standard d'appel de procédure distantes

RPC [DL91] et la norme CORBA a évolué vers le standard RT-CORBA[25] [OMG05, SK00]. Plus récemment, l'émergence du paradigme de publication et de l'abonnement a amené à la norme DDS[26] [OMG04, OMG07]. Il existe aussi des intergiciels dits *schizophrènes* qui assurent la correspondance entre des applications respectant différents standards [Bak01]. Le plus abouti est le PolyORB initialisé par les travaux de Laurent Potet [Pau02] puis de Thomas Quinot [Qui03]. L'adaptation de cet intergiciel au contexte des systèmes temps réel et embarqué a été étudié dans la thèse de Jerôme Hugues [Hug05].

Les différentes normes de distribution adaptées au temps réel fournissent des spécifications supplémentaires permettant aux applications de s'assurer d'une certaine qualité de service ainsi que d'un comportement prédictible dans l'utilisation des services de l'intergiciel sous-jacent. Il faut bien distinguer les spécifications et les implémentations (*C.f.* figure 3.6). Les spécifications sont les interfaces utilisables par l'application (compatible avec la norme) pour demander l'exécution d'un service par l'intergiciel sous-jacent (l'intergiciel de la norme comme l'ORB pour CORBA). Ainsi, ces spécifications supplémentaires pour le temps réel doivent fournir des services et des méthodes aux concepteurs pour les aider à assurer un certain comportement temporel aux applications interconnectées par le middleware. Ce niveau d'abstraction permet aux concepteurs des applications distribuées de s'affranchir de l'implémentation et donc de la mise en œuvre des services fournis par l'intergiciel. En revanche, la mise en œuvre des services proposés dans la norme devra assurer le respect de la sémantique de ces services.

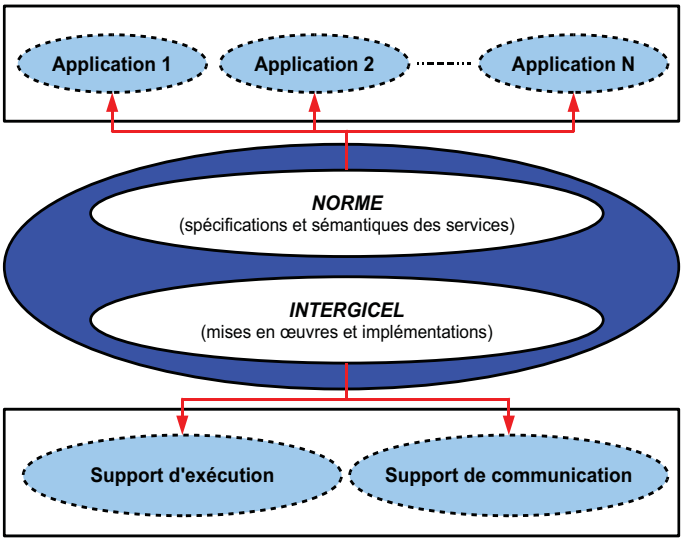


FIG. 3.6 – Distinction entre les spécifications et leur mise en œuvre dans un middleware

Nous présenterons dans la suite les deux standards temps réel les plus utilisés actuellement. La norme RT-CORBA est décrite dans la partie 3.2.2 et les spécificités de la norme DDS sont décrites dans la partie 3.2.3. Nous terminerons sur une discussion des apports de ces différents standards et sur les lacunes de la norme HLA pour le temps réel dans la partie 3.2.4. Notre objectif est donc ici de décrire brièvement les différents services fournis par ces normes et les différentes techniques mises en œuvre afin d'assurer une qualité de service aux applications distribuées inter-connectées.

[25] *Real-Time CORBA*
[26] *Data Distribution Services*

3.2.2 RT CORBA

Le recours régulier à l'utilisation de CORBA dans le contexte des applications temps réel a mené l'OMG à s'intéresser aux extensions possibles de cette norme afin de lui conférer des propriétés temps réel. En effet, le besoin d'interconnexion d'applications temps réel, telles que l'interconnexion de systèmes aéronautiques ont amené l'ajout de fonctionnalités dans les middlewares CORBA telles que des mises à jour pour les service de gestion des événements CORBA et la création de services dédiés à l'ordonnancement des threads CORBA [HLS97], des études de performances des ORBs (intergiciel CORBA) sur différents systèmes d'exploitation [GSHP96, Sch98] ou sur des réseaux ATM [GS96, GS97, GS98]. Ces différentes études et recherches ont abouti au développement d'une démarche de normalisation, afin de pérenniser les recherches, qui a commencé par la rédaction d'un ensemble de propositions au début de l'année 1999 [OMG99a]. Un premier document de spécifications est ensuite apparu [OMG99b] et, de nos jours, la norme en vigueur date de 2005 [OMG05] (un résumé des différentes spécifications de la norme peut être trouvé dans [SK00]). Ces documents se concentrent sur les préoccupations temps réel d'un environnement CORBA pour offrir une meilleure prédictibilité, une meilleure efficacité et un contrôle fin sur les ressources du système. En ce sens, les fonctionnalités offertes par RT-CORBA se décomposent en trois catégories : les fonctionnalités pour la gestion des ressources système, les fonctionnalités pour la gestion des ressources de communication et enfin les méthodes pour assurer une validation de bout en bout du système CORBA distribué.

Les fonctionnalités pour la gestion des ressources système (processeurs, mémoires,...) proposent un ensemble de services pour la gestion des ressources qui permet de résoudre les interférences entre les exécutions des différents traitements. Ces techniques sont basées sur une planification et une affectation de priorités pour les tâches CORBA par l'ordonnancement ; pour cela la norme définit une plage de priorité pour les tâches allant de 1 à 32767. L'unité d'ordonnancement dans l'environnement CORBA est le thread. L'ordonnancement de ces entités de base peut se faire statiquement ou par évaluation en cours d'exécution (dynamiquement) selon les besoins de l'application. Ces deux modes d'ordonnancement sont implantés dans RT-CORBA. La notion de thread dans le contexte de l'ordonnancement statique est analogue à celle définie par la norme POSIX mais RT CORBA utilise en supplément la notion de *pool* de threads[27] qui permet de prévenir et de borner l'apparition de l'inversion de priorité entre les fils d'exécution d'une application CORBA [SMFGG98]. Le pool de threads améliore les performances en permettant la pré-allocation de threads, la séparation et le dimensionnement des groupes de threads [PSCS00]. La partie d'ordonnancement dynamique est basée sur deux notions distinctes. Les premières approches étaient les TDMIs[28] [WBTK95, WDG+99] permettant l'expression des contraintes temporelles pour les requêtes aux travers des nœuds, particulièrement pour la transmission de flux audio et vidéo. La deuxième approche plus tardive [LRCJ04] utilisait la notion de thread distribué issu des travaux sur le système d'exploitation distribué Alpha [Rey92]. Les critères d'échéances sur les exécutions sont définis globalement et répercutés sur chaque nœud. Localement à un nœud, le thread distribué est assimilé au thread classique de l'ordonnancement statique mais, il s'exécute là où il est le plus éligible globalement. Le thread distribué propage ainsi le contexte d'ordonnancement de l'application de nœud en nœud. Bien entendu, dans les deux cas, le standard suppose l'utilisation de systèmes d'exploitation temps réel respectant la norme POSIX en tant que support d'exécution. Des études de compatibilité et de performance ont été menées sur les systèmes d'exploitation Lynx OS, VxWorks ou encore QnX [LFGGS99, LFGS99]. Enfin, la définition d'un temps global est une nécessité afin d'assurer la qualité de service pour les fonctionnalités du système distribué. Pour RT-CORBA, les services de temps [OMG08] re-

[27] *Groupement de threads*
[28] *Timed Distributed Method Invocations*

commandent l'existence d'une horloge commune ou une synchronisation avec le protocole NTP comme c'était le cas aussi dans le standard DIS (*C.f.* partie 2.3.2, page 18).

Les fonctionnalités pour la gestion des ressources de communication (réseaux) tentent de fournir des supports pour améliorer la qualité de service par rapport au support classique CORBA (approche Best effort). Les nouveautés de RT-CORBA proposent des services pour établir des connexions prioritaires et allouer des connexions privées afin d'éviter le mélange des messages dans un souci de déterminisme. Néanmoins, la mise en œuvre de ces services n'est pas explicite. Nous pouvons citer l'ORB TAO qui utilise le protocole Diffserv pour catégoriser ses connections, d'autres travaux ont utilisé le protocole CAN [LJB03]. RSVP est utilisé dans certain cas pour l'établissement de connexions privées [RML03]. Il est aussi possible pour le concepteur de l'application de choisir et de configurer les protocoles utilisés par l'ORB [OKS+00] comme, par exemple, les travaux basés sur une architecture Ethernet Time-Triggered [LJR02]. En revanche, la pertinence du paramétrage des protocoles réseau choisis n'est pas vérifiée par la norme.

Les fonctionnalités pour la validation de bout en bout fournissent un ensemble de méthodes et de techniques basées sur les principes de l'ordonnancement. Cela permet de valider le comportement temporel d'une application distribuée CORBA sous les hypothèses de conception. En effet, RT CORBA combine un support pour l'ordonnancement statique ainsi que l'ordonnancement dynamique [GLS99]. Les techniques de validation d'une application CORBA en utilisant des méthodes d'ordonnancement statique sont les plus abouties et les plus utilisées. Par exemple, l'article [DWE+01] propose une démarche de validation complète basée sur une utilisation conjointe de l'algorithme Deadline Monotonic avec le protocole de gestion de ressource distribué MPCP[29]. Ce protocole prend en compte des blocages causés par les conflits de ressources distribuées. Le protocole MPCP est par conséquent plus difficile à mettre en œuvre sur les systèmes actuels, et nécessite la notion de priorité globale sur tout le système distribué (priorité globale CORBA allant de 1 à 32767). Le lecteur intéressé pourra trouver une description de l'algorithme particulier MPCP dans le livre de Jane W.S.Liu [Liu00] (de la page 352 à la page 360). L'article [DWE+01] présente aussi une méthode de modification des priorités globales CORBA vers les priorités natives des systèmes d'exploitation appelée *Lowest Overlap First Priority Mapping*. Enfin, les chercheurs ont développé des extensions pour l'outil de validation d'ordonnancement PERTS [LRD+93] afin de l'adapter aux hypothèses d'un environnement CORBA et ainsi avoir un environnement générique de validation. Pour l'ordonnancement dynamique, l'utilisation des threads distribués ou des TDMIs permet de distribuer les contraintes temporelles des différentes requêtes au travers du réseau. Ensuite, l'ORB utilise des politiques d'ordonnancement dynamique classiques telles que EDF et LLF, mais aussi de nouvelles politiques mixtes telles que MUF[30] [KTS08], afin d'attribuer des priorités globales CORBA aux différentes requêtes ou threads. Ainsi, chaque nœud exécute la tâche qui lui est assigné localement et qui a la priorité CORBA la plus forte.

L'intergiciel impliqué dans le standard CORBA est l'ORB. Plusieurs ORBs existent et implémentent une partie des services de RT CORBA. Le plus connu est l'ORB TAO[31] [70, 71] implémenté en C++ qui est à l'heure actuelle l'un des rares ORB qui respecte toutes les spécifications. D'ailleurs la plupart des documents précédemment cités proviennent du groupe de travail TAO. Nous pouvons citer l'ORB ZEN [KKS02, KRK03] qui est un ORB temps réel open source, architecturé autour d'un micro noyau implémenté en Java, et qui s'appuie sur les travaux de TAO. En utilisant Java, les concepteurs de ZEN ont dû résoudre deux inconvénients majeurs

[29] *Multiprocessor Priority Ceiling Protocol*
[30] *Maximum Urgency First*
[31] *The ACE ORB*

pour le temps réel : (1) le modèle préemptif d'ordonnancement de la JVM (Java Virtual Machine) varie d'une plateforme à une autre (tourniquet ou préemption par priorité simple) et (2) le ramasse-miettes (*i.e.* Garbage Collector) de la JVM peut préempter n'importe quel autre thread. Ainsi, afin de s'affranchir de l'indéterminisme de la machine virtuelle Java JVM[32], ZEN s'appuie sur les travaux menés sur les spécifications temps réel pour Java RTSJ [GB00] en utilisant des mécanismes tels que la *scoped memory* et les *no-heap real-time threads*. Enfin, nous pouvons citer un projet open source récent : CARDANOM [Cor05] [8]. CARDANOM est développé et maintenu par le CDO[33] qui est un regroupement d'industriels incluant actuellement THALES et SELEX SI. L'architecture de CARDANOM repose sur différents ORBs dont TAO pour les applications `C++` et JacORB [Bro97] [30] pour les applications Java.

3.2.3 DDS

Le succès du standard RT CORBA, mais aussi la rigueur de son mode de conception orientée distribution de requêtes, a amené l'OMG à s'orienter vers une nouvelle norme basée sur le paradigme de la publication et de l'abonnement [EFGK03] : DDS. La première version de cette norme fut élaborée en 2004 [OMG04] et la version actuellement en vigueur date de 2007 [OMG07]. Cette norme complète les spécifications de certains services de RT CORBA, tels que l'*Event Service* [SO03] pour les étendre à un cadre plus générique. Le rôle est de proposer une technologie d'échanges de données sur le réseau orientée message et, ainsi, DDS est une norme pour la conception d'intergiciels et l'interconnexion d'applications réparties. La spécification DDS offre deux niveaux d'interface [PC03] :

1. un bas niveau d'abstraction nommé DCPS[34] qui est hautement configurable et étroitement lié aux données. DCPS est riche de nombreux paramètres pour gérer la qualité de service afin de déterminer le comportement requis et aboutir à une livraison efficace des messages dans le système distribué DDS. Nous nous intéresserons dans la suite à ce niveau car c'est celui qui permet une configuration efficace des paramètres de qualité de service et ainsi autorise l'interconnexion d'applications avec des contraintes temps réel.
2. un niveau d'abstraction optionnel plus élevé nommé DLRL[35] qui offre une approche amplement simplifiée à l'utilisateur et permet une meilleure intégration au niveau applicatif. DLRL est rarement implémenté dans les middlewares DDS et propose un niveau d'abstraction bien trop élevé pour satisfaire les contraintes de déterminisme temporel.

Le modèle conceptuel de DDS est basé sur l'abstraction d'un espace de données commun qui est accessible à toutes les applications concernées et connectées au middleware (*C.f.* figure 3.7). Les applications qui veulent produire des informations (notées les *topics*) dans cet espace commun déclarent leur(s) intention(s) de devenir des producteurs. De même, les applications qui veulent recevoir des informations de l'espace de données déclarent leur(s) intention(s) de devenir des «abonnés». Chaque fois qu'un producteur ajoute une nouvelle donnée dans l'espace de données commun, le middleware DDS propage cette donnée à tous les abonnés intéressés par celle-ci.

Le middleware DDS fournit un ensemble de paramètres configurables pour répondre aux exigences de qualité de service d'une application. Ces caractéristiques de qualité de service, appelées les polices de qualité de service, représentent des paramètres de contrôle du comportement de l'intergiciel DDS. La norme [OMG07] permet ainsi de contrôler de nombreuses polices de qualités

[32] *Java Virtual Machine*
[33] *Common Development Organisation*
[34] *Data Centric Publish Subscribe*
[35] *Data Local Reconstruction Layer*

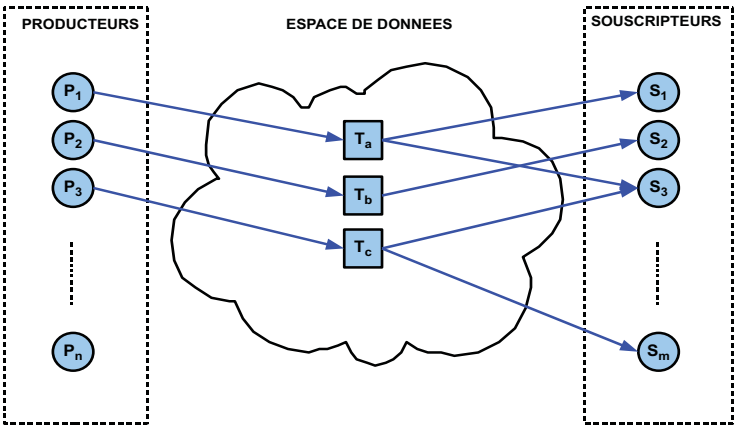


Fig. 3.7 – Espace de données commun dans DDS

de service associées à la production et à la livraison des données. Par exemple, le producteur d'une donnée peut s'engager à écrire une nouvelle valeur périodiquement (au moins une fois à chaque période) par le biais du paramètre `DEADLINE`. De la même façon, ce paramètre permet au consommateur d'exprimer son besoin de recevoir une donnée avec une période donnée. Le paramètre `LATENCY_BUDGET` permet de spécifier le délai maximum acceptable entre le moment où les données sont écrites par le producteur jusqu'au moment où les données sont insérées dans la file d'attente des lecteurs associés. Il faut, bien sûr, préciser que la sémantique de ces différentes polices repose sur la capacité des supports d'exécution et de communication. Les implémentations DDS doivent garantir ces différentes contraintes en se basant sur des systèmes d'exploitation temps réel ou encore des protocoles réseaux spécifiques. La société Real-Time Innovations a proposé en 2002 la création d'un protocole spécifique pour la norme DDS : le protocole RTPS[36] [Inn02]. Ce protocole a ensuite été normalisé par l'OMG [OMG09]. Il est construit sur les protocoles réseaux classiques Ethernet UDP et TCP et propose certaines fonctionnalités supplémentaires afin d'assurer une qualité de service aux échanges de messages dans DDS.

Dans la littérature, peu de travaux sont disponibles sur les techniques d'implémentation et l'utilisation du standard DDS pour des applications temps réel. Nous pouvons citer les travaux du Syscom Laboratory [GRHR07] dans lesquels ils modélisent une application distribuée avec sur chaque nœud un OS Temps réel relié par un réseau temps réel CAN. Une particularité de ce travail est la proposition d'une modélisation formelle de l'application distribuée en utilisant des techniques d'ordonnancement dynamique basées sur l'algorithme EDF[37] afin d'assurer le comportement temporel de leurs applications. Ce modèle permet d'exprimer les tâches du système DDS sous forme de *jobs*, cependant l'article ne précise pas si ces jobs considérés sont des threads ou des processus. Dans d'autres travaux [MH07], ils présentent des critères formels pour l'évaluation des performances d'une application distribuée DDS dans le contexte temps réel mou basés sur le nombre d'échecs en fonction du nombre de connexions. Dans [HSG09], les auteurs utilisent leur implémentation FLEX-MAT[38] de DDS afin de comparer l'utilisation de protocoles réseaux classiques (TCP ou UDP) avec des protocoles plus spécifiques (MultiCast,...) pour assurer le transport de flux audio et video. Un benchmark pour l'utilisation d'un middleware DDS dans

[36] *Real-Time Publish-Subscribe*
[37] *Earliest Deadline First*
[38] *FLEXible Middleware And Transports*

le contexte d'applications temps réel radar peut aussi être trouvé dans [RSP07]. Le tableau 3.2 présente une liste des différentes implémentations existantes du standard DDS (Commerciale ou Open Source).

Nom du logiciel	Distributeur	Licence	Langages supportés	Références
`OpenSlice`	Prismtech	GPL/LGPL	C++	[45]
`RTI DDS`	Real Time Innovations	Commerciale	Java	[61]
`Open DDS`	Object Computing, Inc.	GPL/LGPL	C++	[62]
`Core DX`	Twin Oaks Computing	GPL/LGPL	C++	[13]
`MilSOFT DDS`	MilSOFT Technology	Commerciale	C++/Java	[41]
`InterCOM DDS`	Gallium	Commerciale	C++	[29]

TAB. 3.2 – Liste des intergiciels DDS disponibles

3.2.4 Discussion

Il existe une ambiguïté lorsque l'on parle de la conception d'une norme pour des intergiciels temps réel. En effet, comme nous l'avons explicité précédemment, le rôle de l'intergiciel est d'abstraire la complexité de l'application distribuée aux concepteurs et fournir des primitives aux applications afin qu'elles puissent communiquer dans un environnement distribué (et éventuellement hétérogène) [All03, Pal07]. Inversement, une application temps réel impose au concepteur de rester proche du système. C'est-à-dire qu'il faut s'assurer de l'ordre des tâches s'exécutant sur les processeurs, l'ordre des messages traversant le réseau, dimensionner la mémoire,... Cela implique que le concepteur doit connaître parfaitement les supports d'exécution (*C.f.* partie 3.1.2, page 41) et les supports de communication de son application distribuée (*C.f.* partie 3.1.3, page 43). L'ajout de services pour la gestion d'une qualité de service temporelle (contraintes temps réel) dans les middlewares dépend donc de la mise en œuvre de ses services dans l'implémentation du middleware qui est fonction du langage de programmation choisi, des systèmes d'exploitation supportés ou encore des protocoles réseaux supportés. La mise en œuvre de l'intergiciel sur une architecture donnée doit permettre de vérifier la sémantique des services temps réel telle que le respect d'une deadline dans le transfert d'un message.

La norme RT CORBA est relativement ancienne et a profité de nombreux travaux de différents universitaires et industriels pour s'étendre et être utilisée dans de nombreux domaines. De nombreux intergiciels (ORBs) ont été implémentés en respectant totalement ou partiellement les spécifications du standard et utilisés sur des plate-formes particulières. Depuis sa création, ce standard s'est enrichi en incluant des spécifications de plus en plus complexes. Un point fort de cette norme est l'inclusion de principes basés sur la théorie de l'ordonnancement pour la validation de bout-en-bout d'une application distribuée. Néanmoins, le paradigme de programmation orienté distribution de requêtes était trop spécifique pour s'adapter à tout type d'application. De plus, dans CORBA, il n'existe aucune spécification pour borner les délais de transfert d'une requête, cela peut être très pénalisant pour assurer la validité d'une application temps réel dure, c'est pourquoi le terme qualité de service est plus utilisé que le terme déterminisme et temps réel dans les travaux CORBA.

A l'instar de RT CORBA, la norme DDS, basée sur un paradigme orienté échange de données, propose une solution plus adaptable à des applications hétéroclites. Les spécifications de qualité de service dans le standard ne concernent que les messages et les détails de l'implémentation

sont laissés aux choix du concepteur de l'intergiciel. Par exemple, il est impossible de connaître la structure de l'intergiciel et les détails pour définir une entité ordonnançable (telle que le thread ou le processus) ne sont pas identifiés clairement. Ainsi la définition de la sémantique se limite à assurer certaines garanties sur la livraison des messages mais aucune technique n'est envisagée pour valider de bout en bout l'application. Cependant, cette norme est relativement nouvelle et elle est encore peu utilisée. De plus, moins de travaux sur la mise en œuvre et sur des expérimentations sont disponibles.

La norme DDS est très proche de la norme HLA [JC09] que nous souhaitons utiliser pour nos travaux. Néanmoins, l'absence de spécifications temporelles dans la norme HLA elle-même, et donc dans l'implémentation des services fournis par l'intergiciel HLA, a conduit les concepteurs d'applications temps-réel à bouder cette norme. D'autres réalisent des simulations temps réel en combinant l'utilisation des deux normes [LR11]. Plusieurs travaux se sont succédés dans la littérature afin d'inclure des spécifications supplémentaires dans la norme HLA et l'utiliser pour mettre en œuvre des simulations temps réel. Dans un premier temps, nous expliciterons les lacunes de qualité de service et les évolutions envisagées pour cette norme et ensuite nous présenterons les différentes techniques de mise en œuvre pour des simulations temps réel (*C.f.* partie 3.3). L'objectif de cette réflexion nous permettra ensuite de présenter nos travaux pour assurer l'exécution de simulations toutes entières soumises à des contraintes temps réel.

3.3 Vers une norme HLA temps réel

3.3.1 Les besoins temps réel en simulation

Comme nous l'avons expliqué dans le chapitre précédent (*C.f.* partie 2.3.2, page 18), la norme HLA fut censée remplacer la norme DIS spécifiquement conçue pour des simulations "temps réel". Les contraintes temporelles soumises aux applications DIS étaient dues à l'intégration d'un opérateur humain dans la boucle de simulation. En effet, la technologie DIS était dédiée à la simulation et la réalisation de théâtres de guerre c'est-à-dire une réalité synthétisée [Ell94b] (telle qu'un champ de bataille) partageable avec d'autres entités et avec laquelle l'opérateur humain pouvait interagir par l'intermédiaire de technologies diverses. Cela impliquait une immersion sensorielle de l'opérateur humain (vision, audition, ...). Il était donc primordiale d'assurer que cet environnement virtuel paraisse le plus réaliste possible pour l'opérateur en fonction de ses capacités physiologiques. En ce sens, tout le système DIS devait assurer le respect de certaines contraintes temporelles [Sym99] :

1. **100 ms** entre l'input et l'output d'une PDU (*C.f.* tableau 2.1, page 19) entre deux simulateurs pour les interactions fortement couplées et **300 ms** entre l'input et l'output d'une PDU entre deux simulateurs pour les interactions faiblement couplées ;
2. La prise en compte de la latence et de la gigue dus au réseau doit permettre aux simulateurs en jeu dans la simulation de respecter les contraintes pour les interactions faiblement et fortement couplées.

De façon générale, dans un environnement virtuel, l'opérateur humain tolère une latence de l'ordre de 250 ms mais tout délai supérieur à cette borne peut dégrader la qualité de l'environnement [ABW93, LHM+97, WO00]. Au-delà de ces délais, l'opérateur pouvait être impacté et perturbé lors de l'utilisation de cette application pour le respect des contraintes de temps et l'homogénéité du temps de réaction du système Les mécanismes de DIS tels que l'utilisation des PDUs et les synchronisations des sites par protocole NTP associés aux techniques de prédictions (algorithme de *dead-reckoning*) permettaient d'assurer l'homogénéité des simulations DIS avec

ces contraintes de temps de l'ordre de la centaine de millisecondes. De la même façon, la qualité des flux vidéo et audio (utilisés aussi dans le cadre des environnements virtuels) est aussi dépendante des capacités physiologiques de perception de l'être humain et doivent respecter des contraintes de l'ordre de la centaine de millisecondes [HFP+00]. En revanche, ces constantes de temps ne sont plus vraies dans le cas des systèmes embarqués. Les matériels réels tels les actuateurs, les capteurs ou les calculateurs embarqués sont beaucoup plus exigeants. Ils sont soumis à des contraintes de temps extrêmement petites peuvent être de l'ordre de la milliseconde voir de la microseconde. De plus, ces systèmes ne tolèrent aucune flexibilité dans le respect de leurs échéances temporelles. Une architecture de simulation utilisable pour simuler des systèmes embarqués pourra ensuite aisément s'adapter aux contraintes des environnements virtuels. La figure 3.8 récapitule les besoins temps réel en simulation.

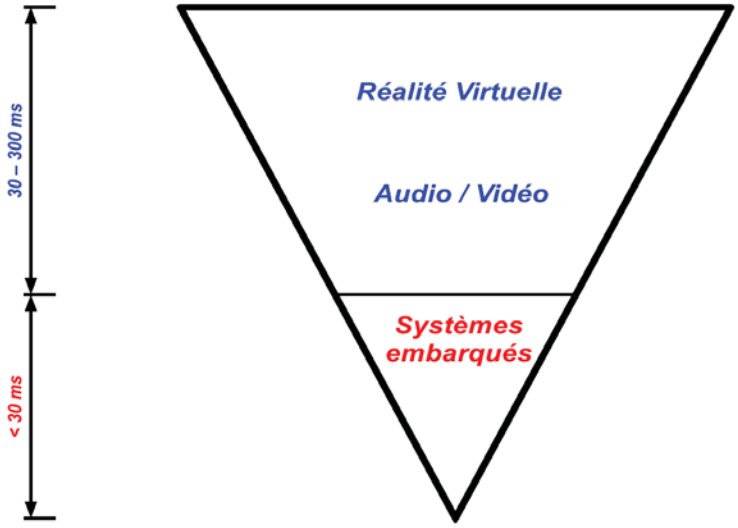


FIG. 3.8 – Les besoins temps réel en simulation

HLA reste une voie de recherche à envisager pour interconnecter des systèmes embarqués [Pea03] et, dans ce travail, nous allons l'utiliser pour l'interconnexion de nos simulations temps réel. Cependant, la norme a des lacunes dans les spécifications relatives à la qualité de service. Par exemple, les spécifications d'interface de la norme HLA [DoD98c, IEE00d] n'offrent aucune directive sur les comportements temporels des services sauf en ce qui concerne le `tick()`. En effet, ce service dans sa version paramétrée, permet à l'utilisateur de préciser des durées minimales et maximales que la RTI peut utiliser avant de rendre la main au fédéré appelant. De la même façon, l'OMT [DoD98b, IEE00b] ne fournit aucune police pour la prise en compte de paramètres sur le comportement temporel pour les données échangées entre les fédérés d'une fédération. L'OMT fournit simplement des critères simples pour choisir le type de transmission et l'utilisation des mécanismes de gestion du temps :

1. La spécification du mode de transport des données échangées par la fédération selon le mode `BEST_EFFORT` (protocole UDP) ou le mode `RELIABLE` (protocole TCP). Le mode `BEST_EFFORT` assure que la donnée arrivera le plus vite possible mais il n'y a aucune assurance sur l'arrivée sans perte du message. Le mode `RELIABLE` assure la mise en place de mécanismes de contrôle qui permettront de s'assurer de la réception de la donnée. Cependant, ces mécanismes de contrôle rendront le transfert du message plus long que le mode `BEST_EFFORT`.
2. La spécification de l'horodatage ou non de la donnée sur le temps simulé (mode `TIME STAMP` ou mode `RECEIVE`). C'est l'intergiciel HLA (la RTI) qui assurera ensuite la livraison des

données avec le respect des mécanismes de gestion du temps (*C.f.* partie 2.5, page 31).

3.3.2 L'ajout de paramètres de QoS

Hui Zhao [ZG01a, Zha01] fut le premier à s'intéresser aux lacunes de la norme HLA en termes de qualité de service et à proposer des extensions. Il propose 9 paramètres (regroupés dans le tableau 3.3) qui permettraient au concepteur d'une fédération HLA de définir explicitement ses besoins en qualité de service (au niveau des fédérés et des données).

Indice	Paramètres	Description
1	`Taille de la Donnée (Bytes)`	Taille des attributs de l'objet ou des paramètres de l'interaction
2	`Fréquence moyenne de production (Bytes/Sec)`	Fréquence moyenne de production de données pour le fédéré
3	`Fréquence maximale de production (Bytes/Sec)`	Fréquence maximale de production de données pour le fédéré
4	`Temps de production de la donnée (Milliseccondes)`	Temps de calcul de la donnée
5	`Delai maximal (Milliseccondes)`	Temps de transfert entre la production de la donnée et sa consommation par un autre fédéré
6	`Pourcentage de pertes (%)`	Pourcentage de pertes de paquets toléré par l'application pour la donnée
7	`Niveau de priorité [1-32767]`	Importance de la donnée dans une fédération
8	`Partage de canal`	La donnée a t'elle besoin d'un canal privé ou peut elle utiliser un canal partagé
9	`Niveau de sécurité`	Niveau de sécurité de la donnée dans une fédération

Tab. 3.3 – Paramètres de qualité de service HLA

Le paramètre 1 peut être directement dérivé des informations de la FOM. En effet, l'OMT définit des types classiques pour les attributs d'objets ou les classes d'interaction (`double`, `integer`, ...). Les paramètres 2 et 3 traduisant la fréquence moyenne et maximale de production ont été repris et intégrés dans la nouvelle norme IEEE 2010 de HLA et sont intégrés dans un nouveau service décrit dans la partie suivante (partie 3.3.3). Le temps de production de la donnée (paramètre 4) est dépendant des algorithmes implémentés dans le fédéré publicateur et du support d'exécution sur lequel se trouve le fédéré (système d'exploitation, langage de programmation, allocation mémoire, ...). On peut l'assimiler au pire temps de calcul nécessaire pour produire cette donnée. Les paramètres 5 et 6 sont clairement dépendants de la ressource de communication choisie pour acheminer la donnée. La technologie réseau utilisée, la topologie du réseau ainsi que le protocole réseau utilisé auront un impact sur ces paramètres. Hui Zao envisage l'ajout de nouveaux types de transport aux deux types existants (`BEST EFFORT` ou `RELIABLE`). Ces nouveaux types de transport sont principalement basés sur l'inclusion d'un support multicast en se basant sur les travaux des protocoles pour DIS [Pul99]. La prise en compte du paramètre 7 est plus complexe à mettre en œuvre. Cela repose sur la notion de priorité de la donnée et cela suppose qu'une telle donnée serait prioritaire aussi bien lors des transferts réseaux que dans les

exécutions de la RTI. La définition d'une plage de priorité globale allant de 1 à 32767 est directement dérivée des travaux sur RT CORBA (priorités globales des tâches CORBA). Cependant, une telle plage n'est pas très réaliste car la priorité de la donnée transmise doit être prise en compte aussi bien au niveau des transports sur le réseau que des exécutions de la RTI. De plus, les systèmes d'exploitation ne tolèrent pas des plages de priorités si grandes et il en est de même pour les réseaux. La notion de partage de Canal (paramètre 8) est utile pour déterminer si la donnée peut partager un bus réseau ou bien si elle a besoin d'un canal privé. Ainsi la déclaration de l'intention de publier un attribut par un canal privé devrait par exemple générer la création de ce canal privé grâce à l'utilisation d'un protocole de réservation de ressource réseau tel que le protocole RSVP. Le dernier paramètre (paramètre 9) de qualité de service ne répond pas à des contraintes temporelles mais à des contraintes de sécurité de la donnée. Cela peut être important dans certains types de simulation, mais ce n'est pas le cas pour la problématique qui nous concerne. Toutefois, les extensions de sécurité dans la norme HLA ont aussi été envisagées par d'autres auteurs [BCSZ98, KK04].

La prise en compte de ces différents paramètres et des nouveaux types de transport doit se faire par une mise à jour des tables de l'OMT (tables d'attributs et de paramètres relatifs aux objets et aux interactions) ainsi qu'une mise à jour de la table de transport, mais surtout par la mise à jour des services HLA existants ou la création de nouveaux services. Nous détaillons ce point dans la partie suivante.

3.3.3 La mise à jour et l'ajout de services

Les fédérés communiquant au sein d'une fédération utilisent les services de publication et d'abonnement afin de déclarer leurs intentions pour générer de l'information ou bien d'en recevoir. L'appel à ces services ne permet pas de prendre en compte des paramètres de qualité de service mais il serait possible de les adapter. L'utilisateur pourrait ainsi choisir un nouveau type de transport s'ajoutant aux deux types de transports classiques de HLA. Ces services devraient aussi permettre d'intégrer la création d'un canal privé pour la donnée. Ainsi la déclaration de l'intention de publier un attribut par un canal privé devrait par exemple générer la création de ce canal privé. Et inversement, les services de résiliation relatifs à la publication et l'abonnement devraient fournir les services inverses permettant de libérer ces canaux privés. Hui Zhao [Zha01] propose aussi l'ajout de nouveaux services `SetPriority()` et `GetPriority()` pour affecter une priorité à un attribut d'une classe d'objet ou d'interaction. Cette priorité devrait ensuite influer sur la façon dont l'attribut ou le paramètre sera traité par la RTI. Cependant, comme nous l'avons expliqué précédemment la mise en œuvre serait assez difficile. Dans son document de thèse, Hui Zhao décrit ces services mais ne les implémente pas dans une RTI donc nous ne connaissons pas les techniques sous-jacentes de mise en œuvre de ces nouveaux services ou des mises à jour.

L'exemple de plus concret de l'ajout de nouveaux services est basé sur l'idée de la fréquence de production d'une donnée. En effet, ce principe a été repris par les concepteurs du RTI pitch [Mö05]. Dans leurs travaux, cela est vu du côté du fédéré souscripteur qui peut utiliser une option (`SmartUpdateReduction`) pour souscrire à certaines mises à jour d'attribut avec une fréquence maximale qu'il peut spécifier. Le producteur des données reçoit une notification de la vitesse maximale à laquelle un attribut est souscrit. Ces principes ont été repris dans les propositions d'extensions de la norme (nommé HLA *Evolved*) [Mö08] puis acceptés dans la mise à jour 2010 de la norme IEEE. La figure 3.9 donne une illustration du principe de fonctionnement de cette nouvelle fonctionnalité.

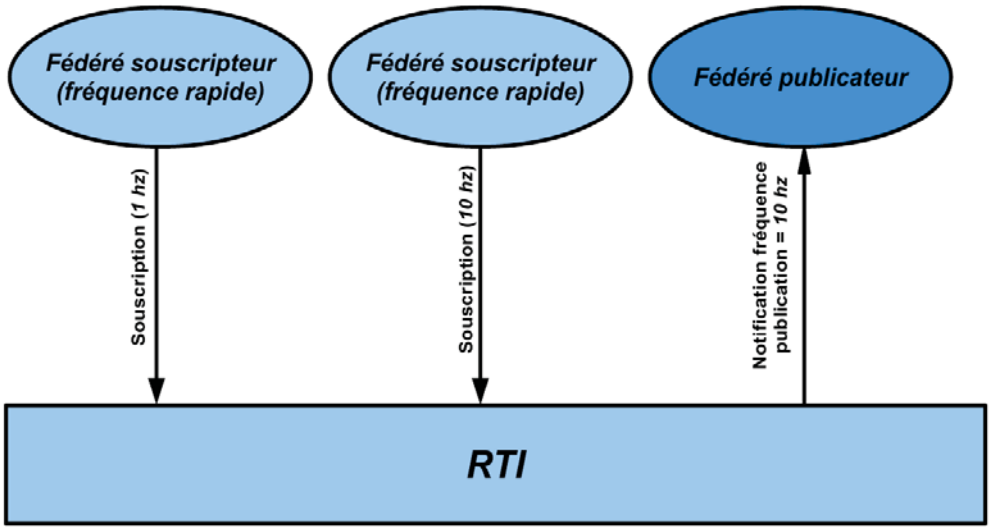


FIG. 3.9 – Illustration du fonctionnement de l'option `SmartUpdateReduction`

Tom McLean et Richard Fujimoto se sont aussi intéressés aux simulations temps réel dans le contexte des environnements virtuels et des simulations de théâtre de guerre [Fuj97, MF00, McL01, Mcl02]. Leurs travaux sont principalement basés sur l'utilisation des mécanismes de gestion du temps spécifiques à la norme HLA et nous les détaillerons plus précisément dans la partie 3.4.1 et 3.4.2. Afin d'être complètement compatible avec leur approche basée sur la gestion du temps, dans leur article [MF00] (*C.f.* section 5.5 de cet article), ils proposent aussi un ensemble de mises à jour de services existants (`joinFederationExecution()`, `updateAttributeValues()`, ..) ainsi que des nouveaux services (`advanceRunTimeClock()`, `setRunTimeClock()`, ..). Par exemple, le service `joinFederationExecution()` a été complété afin d'assurer que chaque fédéré qui utilisera ce service pour rejoindre la fédération aura, en même temps, ses valeurs de temps logique local et de lookahead initialisées par défaut. Ces services ont été intégrés dans leur RTI basée sur la FDK[39] [MF01] afin de mettre en œuvre des mécanismes de gestion du temps spécifiques à leurs simulations temps réel.

3.3.4 Indéterminisme dans les simulations

Comme pour l'utilisation de tous les standards pour les middlewares, il faut distinguer la partie spécifications du standard que nous venons de décrire avec les mises en œuvre que nous décrirons dans la partie suivante (*C.f.* partie 3.4). Les différentes techniques de mise en œuvre de simulations distribuées HLA doivent permettre de s'affranchir des sources d'indéterminisme inhérentes des systèmes distribués temps réel. En effet, la conception d'une simulation distribuée temps réel dépend de trois points :

1. Les contraintes temporelles soumises aux applications interconnectées (les fédérés HLA). Comme nous l'avons vu dans la partie 3.3.1, les simulations incluant un opérateur humain (simulation *Human-in-the-loop*) telles que les environnements virtuels ou la diffusion de flux audio et video sont soumises aux capacités physiologiques de l'humain. Ces systèmes sont considérés comme temps réel souple et peuvent admettre des failles car cela ne mènera pas à des situations catastrophiques mais peut occasionner de la gêne pour l'utilisateur. Les simulations de systèmes embarqués ou incluant des systèmes embarqués réel (simulation *Hardware-in-the-loop*) dans la boucle de simulation sont plus restrictives car elles ne tolèrent

[39] *Federation Developers Kit*

aucune erreur du système et les contraintes temporelles sont très petites. De plus, dans les deux cas, les événements du monde extérieur sur la simulation tels que les actions de l'opérateur ou encore la prise en compte de systèmes réels de perception (capteur) ou d'action (actuateurs) dans les simulations ne sont pas garantis de pouvoir être prédites par une analyse avant l'exécution de la simulation.

2. Les modes de fonctionnement et les principes d'exécution des différents simulateurs dépendent principalement des fonctionnalités implémentées et du support d'exécution du simulateur. Par exemple, les mécanismes de gestion de la mémoire ou les interruptions systèmes sont des sources d'indéterminisme qui doivent être pris en compte et qui peuvent influer sur le comportement global de la simulation.
3. Les principes d'échange de messages entre les simulateurs doivent garantir des livraisons de messages dans le bon ordre, dans le bon temps et cela sans perte (ou alors avec un pourcentage de perte connu et maîtrisé). Cela dépend du support de communication (réseau) mais aussi des traitements réalisés par le middleware ainsi que des méthodes de synchronisation entre les différents sites distribués.

3.4 Mise en oeuvre de simulations HLA temps réel

3.4.1 Définitions du mode d'exécution pour les simulateurs

Fujimoto et McLean furent les premiers à définir des modes d'exécution pour les simulateurs temps réel [Fuj97, MF00, McL01]. Dans leurs études, ils introduisent le concept de la répétabilité dans les simulations pour traduire des exécutions périodiques. Ainsi, chaque simulateur (fédéré HLA) engagé dans ce type de simulation répète, périodiquement, le même schéma d'exécution avec un pas de temps noté Δt. Ce schéma d'exécution est composé de quatre phases successives (1) une phase de réception, (2) une phase de calcul, (3) une phase de transmission et (4) une phase de temps libre (une illustration est donnée dans la figure 3.10). Lorsque le simulateur ne respecte pas son pas de temps Δt cela mène à une erreur globale du système. C'est pourquoi les détails de chaque phase doivent permettre de garantir le déterminisme assurant une exécution périodique respectée pour tous les fédérés de la simulation HLA.

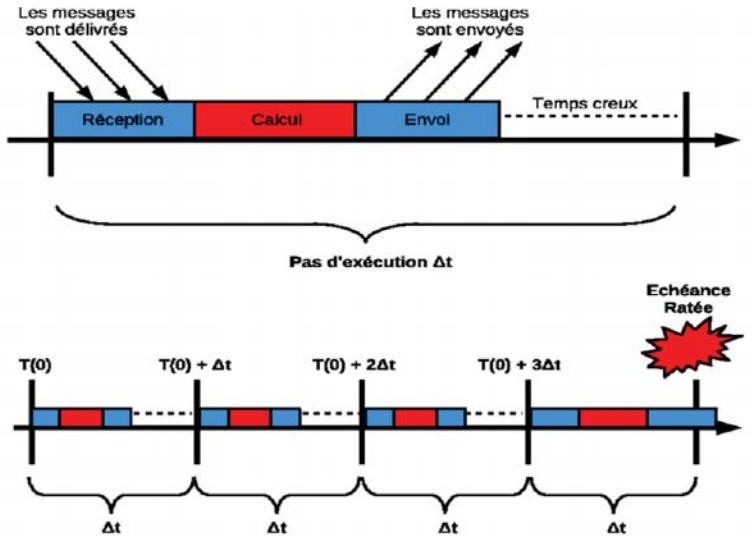


FIG. 3.10 – Illustration de l'exécution périodique d'un fédéré

Dans une fédération HLA composée de fédérés ayant ce modèle d'exécution, le fonctionnement de l'application est basé sur la technique de synchronisation utilisée. Les fédérés peuvent partager une même référence temporelle ou horloge globale (WCT[40]). Cette technique permet d'assurer la cohérence de l'ensemble de la simulation et les messages reçus par le fédéré au début de chacun de ses cycles sont supposés bons et sûrs. En revanche, cette technique dépend de la précision de l'horloge. Une horloge GPS ou une technique de synchronisation d'horloge par protocole NTP (jusqu'à la version 3) permet de respecter des contraintes de temps de l'ordre de la centaine voire de la dizaine de millisecondes selon l'étendue du réseau (*C.f.* partie annexe B.4.1, page 170). De telles horloges distribuées sont plus délicates à utiliser dans le cas de simulations de systèmes embarqués soumis à des contraintes de temps très petites. Fujimoto et McLean utilisent les mécanismes de gestion du temps spécifiques à la norme HLA pour synchroniser les simulateurs temps réel et s'assurer de la livraison cohérente des messages dans toute la simulation. Ils proposent différentes techniques de gestion du temps afin de garantir des exécutions périodiques déterministes pour les simulateurs (nous détaillerons ces techniques dans la partie 3.4.2).

Hui Zhao propose une approche similaire pour la transmission d'un flux audio dans une fédération HLA (*C.f.* chapitre 3 de son document de thèse [Zha01]). Sa fédération est composée de deux fédérés périodiques (*C.f.* figure 3.11) : un fédéré émetteur (a) et un fédéré récepteur (b). L'émetteur du flux audio lit le flux sur le disque dur et le transmet sur le réseau en le découpant en paquet de tailles prédéfinis. Le récepteur reçoit la donnée et reconstruit le flux à l'aide d'un buffer pour, ensuite, le faire écouter sur la sortie audio. Les performances sont évaluées de façon auditive pour voir si le flux ne subit aucune altération lors de la transmission. En revanche, il ne précise pas la technique de synchronisation utilisée entre de ces deux fédérés.

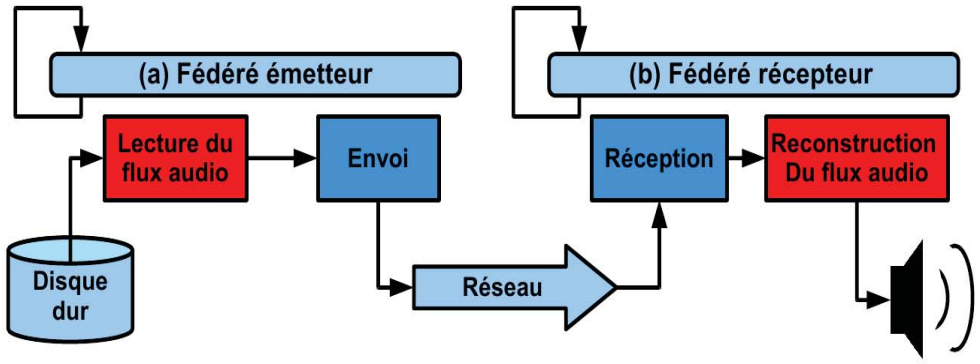


FIG. 3.11 – Transmission du flux audio entre entre deux fédérés périodiques

Les benchmarks classiques du DMSO pour les simulations HLA [Fit02] ainsi que des benchmarks plus spécifiques aux simulations HLA temps réel [Tie03] reposent aussi sur des modes d'exécutifs répétitifs. De plus, ce mode d'exécution périodique pour les fédérés convient parfaitement pour décrire des simulations de systèmes embarqués. Par exemple, dans [LKV02], un simulateur de vol basé sur des simulateurs périodiques échangeant des données à différentes fréquences est implémenté et testé. Le simulateur RMUS[41] [GNRS03] reprend ce principe avec différents modules qui communiquent et s'exécutent à des fréquences données. Dans [LKV02], la technique de synchronisation des différents composants du simulateur de vol n'est pas précisée et dans [GNRS03] les différents composants sont synchronisés en utilisant une horloge commune supposée assez précise pour garantir la validité du système global.

[40] *Wall Clock Time*
[41] *Real-time Multi Uav Simulator*

3.4.2 Utilisation de services spécifiques

HLA est une norme dédiée à la simulation distribuée et, malgré son manque de spécifications de qualité de service, elle propose un certain nombre de services intéressants pour des simulations temps réel. En effet, comme nous l'avons précisé dans la partie 2.5, un des atouts majeurs de la norme HLA repose sur ses spécifications de mécanismes de gestion du temps (nous parlons ici de temps logique ou temps simulé). L'objectif est de contrôler les instants où les différents fédérés effectuent des actions afin de déterminer un ordre sur l'échelle de temps logique. HLA supporte plusieurs politiques de gestion du temps qui influeront sur la manière dont les fédérés avanceront dans le temps simulé. Le respect de la sémantique de ces services est assuré par l'implémentation de différents algorithmes issus de la communauté de la simulation à événements discrets (*C.f.* partie 2.2.3, page 15).

Dans [MF00, McL01], Fujimoto et McLean définissent des fédérés régulateurs et contraints utilisant des mécanismes d'avancée par pas de temps. Ces mécanismes sont basés sur l'utilisation du service `timeAdvanceRequest(`t`)` (ou `TAR(`t`)`) pour avancer dans le temps simulé. Ainsi, en respectant le mode de fonctionnement de ce mécanisme (*C.f.* partie 2.5.2, page 33), ils recevront tous les messages attendus pour le temps logique demandé avant de recevoir l'acquittement de la RTI par la callback `timeAdvanceGrant(`t`)` (ou `TAG(`t`)`) pour pouvoir commencer leur phase de calcul. La figure 3.12 illustre ce principe de fonctionnement.

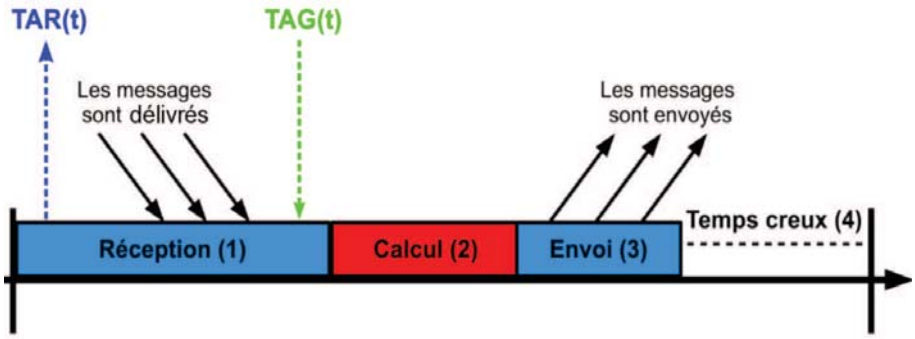


FIG. 3.12 – Fédérés périodiques utilisant le service `TAR()`

Ce mécanisme est associé à une technique d'assignation des estampilles temporelles afin de prendre en compte la latence du réseau ainsi qu'un ensemble de services supplémentaires (décrits brièvement dans la partie 3.3.3). En revanche, l'algorithme utilisé afin d'assurer le fonctionnement correct de l'avancée par pas de temps n'est pas décrit explicitement. Nous supposons qu'il s'agit d'une implémentation de l'algorithme de Mattern (algorithme de seconde génération) car la notion de *lookahead* (propre à l'algorithme CMB de première génération) n'est jamais citée. De plus, dans un article plus récent [MF03], ils décrivent explicitement l'utilisation de l'algorithme de Mattern dans leurs implémentations. Cet algorithme est basé sur la notion d'état des processus et la RTI joue le rôle du processus de contrôle qui connaît l'état global de l'ensemble de la fédération. En revanche, avec cet algorithme, le calcul du $LBTS$[42] (donc de l'état global) ne peut pas être borné car la prise en compte des messages *transiants* (message en transit sur le réseau lors du calcul du LBTS) peut imposer d'interrompre le calcul puis de le recommencer. Cela pose un problème de déterminisme pour les simulations temps réel car on ne peut pas borner le temps nécessaire pour accomplir la phase de réception des messages (entre le `TAR(`t`)` et le `TAG(`t`)`). Ainsi, dans [MF03], ils proposent une technique appelée *offset-epoch method* afin d'assurer un

[42] *Lower Bound Time Stamp*

calcul de LBTS déterministe. Cependant, cette technique repose sur l'hypothèse d'une horloge commune à tous les simulateurs ce qui nous ramène aux problèmes de précision de l'horloge commune afin d'assurer le respect des échéances temporelles. Nous pouvons aussi citer d'autres travaux basés sur des fédérés utilisant les mécanismes de gestion du temps et une avancée par pas de temps [Jan04, Phi05].

Un autre type de service est aussi utilisé dans le contexte des simulations temps réel. Il s'agit des mécanismes de gestion de la distribution des données qui permettent de filtrer les échanges de données en définissant des zones d'intérêt et ainsi d'optimiser les échanges de données au sein de la fédération. Ces services sont particulièrement utilisés dans la simulation temps réel d'environnements virtuels tels que la simulation STOW d'un théâtre de guerre à grande échelle [RSM97]. Nous pouvons aussi citer les travaux de Richard Fujimoto [TF98] et de Victor Skowronski [SD01] sur la synchronisation des mécanismes de gestion de distribution de données avec ceux de gestion du temps afin d'optimiser la circulation des données et leur synchronisation. Azzedine Boukerche [BL05], quant à lui, propose des services de gestion de la distribution des données optimisés et intégrés dans sa RTI. Ces services reposent sur une méthode hybride, dynamique et optimisée pour déterminer les régions d'intérêt, les détails sur cette nouvelle méthode peuvent être trouvés dans [BR02]. Cependant, les services de gestion de la distribution des données paraissent plus délicats à utiliser pour les simulations de systèmes embarqués qui doivent être beaucoup plus rigoureuses en terme de déterminisme. En effet, ces services optionnels, bien que très utiles pour les simulations à grandes échelles pour améliorer le routage des données, fournissent des mécanismes trop complexes pour une utilisation dans le contexte des simulations temps réel de systèmes embarqués.

3.4.3 Techniques d'implémentation pour les RTIs

Comme nous l'avons décrit dans la partie 2.4.5, la RTI est l'intergiciel (ou middleware) servant d'intermédiaire entre les différents simulateurs et les couches basses du système (protocole réseau et système d'exploitation). C'est ce logiciel particulier qui fournit les différents services décrits dans la norme HLA et implémente les algorithmes correspondants afin d'assurer le respect de la sémantique de ces services. Les spécificités de l'implémentation et les algorithmes de la RTI sont déterminants pour assurer les critères de performance et de déterminisme de la simulation globale. Le déterminisme d'une RTI repose sur le choix du langage de programmation, la structure de programmation ainsi que sur les supports d'exécution et de communication.

Les deux langages de programmation les plus utilisés pour l'implémentation d'un RTI sont les langages objet Java et `C++`. Le langage `C++` est utilisé pour l'implémentation du RTI du DMSO [HNB97, BFMT98], la RTI-KIT [FMPT00, ZG01b, MFF04] et la FDK [MF01] de Fujimoto et McLean, la RTI du TNO [Jan04], la MAK-RTI [WG01][42] ainsi que pour le CERTI (*C.f.* partie 2.6, page 35). L'utilisation du Java pour implémenter une RTI est aussi répandue et a été envisagée par les concepteurs de la PITCH-RTI dès les débuts de l'existence du standard [Kar98]. La même année, un prototype pour une RTI temps réel a été implémentée par la société Virtual Workshop [MS98]. Cependant, il n'y a aucun détail sur la structure de l'implémentation et les auteurs n'évoquent pas les limitations inhérentes à Java pour le temps réel telles que l'indéterminisme de la machine virtuelle Java JVM[43]. Maintenant, les deux RTIs JAVA les plus connus sont la RTI commerciale de PITCH [KO01][47] et la RTI Open Source PORTICO [48].

De nombreuses autres implémentations existent mais souvent les architectures utilisées ne sont pas clairement décrites. Hui Zhao [ZG01b] et Azzedine Boukerche [BL05, BSZ07] recom-

[43] *Java Virtual Machine*

mandent une éventuelle architecture pour une RTI temps réel (*C.f.* figure 3.13). Cette architecture est basée sur les pools de threads (comme dans RT CORBA), les files de messages prioritisées ainsi qu'un module d'ordonnancement global à toute la RTI.

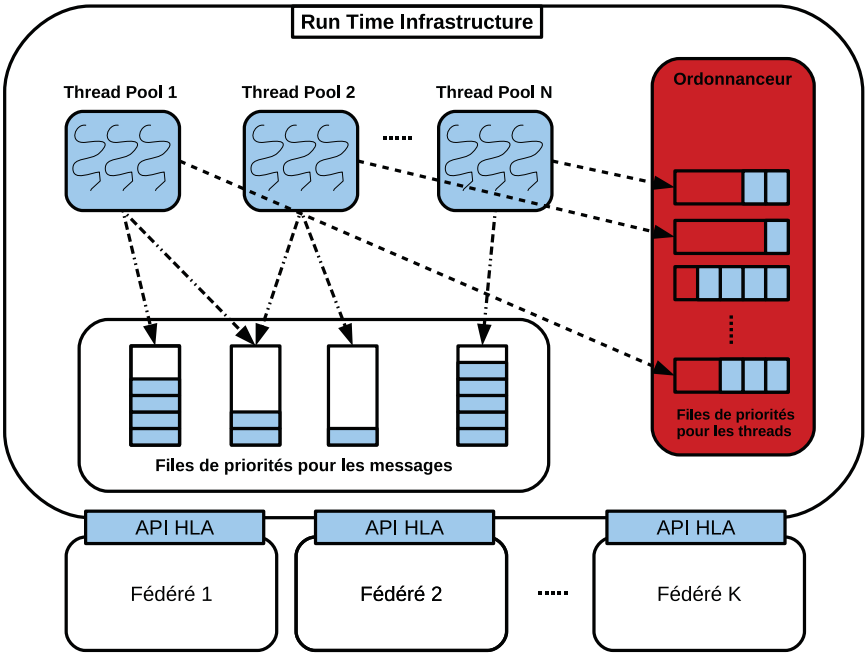


FIG. 3.13 – Architecture possible pour une RTI temps réel

Les travaux de Hui Zhao sont uniquement basés sur les connaissances du RTI-NG. Il n'y pas eu d'implémentation car le logiciel RTI NG etait téléchargeable uniquement en format binaire (exécutables) et les utilisateurs ne pouvaient en aucun cas modifier les sources de l'implémentation. Il n'a donc pas été confronté aux problèmes de gestion de ressources des architectures multi-threads ainsi qu'à la complexité de définir un ordonnanceur global pour une application distribuée. La structure de la RTI proposée par Azzedine Boukerche n'est pas clairement décrite aussi. Cependant, il propose, pour sa RTI, une notion d'équilibrage de charge qui ressemble à un ordonnanceur global permettant un placement et une exécution des différents traitements dans la RTI. Il évoque aussi l'utilisation de différentes techniques d'ordonnancement statique mais il ne précise pas lesquelles, ni les techniques de mises en œuvre de ces techniques dans les threads pools de la RTI. Les supports d'exécution utilisés ne sont pas précisés, cependant les auteurs recommandent l'utilisation de systèmes d'exploitation temps réel compatibles avec le standard POSIX. La RTI du TNO a été testée sur le système d'exploitation Linux Red Hawk [BKBH08] ; c'est aussi le système que nous avons choisi pour notre plate-forme (*C.f.* chapitre 4, page 69).

Au niveau des supports de communication, Hui Zhao utilise le protocole RSVP pour la réservation, les supports DiffServ et IntServ ainsi que les technologies multicast pour s'assurer des communications performantes ; ces protocoles sont décrits en annexe (*C.f.* partie B.3.2, page 169). Les protocoles et les résultats des expérimentations réalisées par Hui Zhao sont disponibles dans le chapitre 9 de son manuscrit de thèse [Zha01]. Ning Li et al., quant à eux, utilisent le protocole RTP (construit au-dessus du protocole UDP) pour s'assurer de la qualité de service dans le transport des flux audio et vidéo dans la RTI [LPZ+06]. La même idée avait été initiée

par Bruzman et Zyda avec la définition du protocole VRTP[44] construit au dessus du protocole RTP dans le contexte des simulations d'environnements virtuels. Les travaux de Fujimoto [FF99, ZG01b] et de Christensen et Van Hook [FF99] se sont intéressés aux communications par mémoire partagée pour les échanges de données sur le même nœud. Ces travaux ont aussi été envisagés dans le CERTI et les résultats des expérimentations avaient été encourageants [ABR+04].

[44] *Virtual Reality Transfert Protocol*

Deuxième partie

CONTRIBUTIONS

Chapitre 4

Supports et Modes d'exécution pour des simulations temps réel

Préambule

L'état de l'art ayant été présenté dans la partie précédente, nous allons maintenant poser les premières briques de notre travail. Ce chapitre introduit, dans un premier temps, la démarche envisagée et les différents choix matériels et logiciels sur lesquels repose notre travail. Ensuite, dans un deuxième temps, nous décrirons les deux principaux cas tests envisagés pour des études de simulations de systèmes embarqués. Le premier cas test vient d'une problématique du domaine spatial et le second d'une problématique du milieu aéronautique ; nous détaillerons les caractéristiques et les spécificités de ces deux applications. Enfin, en nous basant sur les caractéristiques des cas tests, nous présentons les différents modes d'exécution définis et envisagés pour des fédérés temps réel. Ces modes d'exécution traduisent des calculs périodiques et se distinguent par les différentes techniques de synchronisation utilisées pour conserver une concordance temporelle dans la fédération.

4.1 Démarche envisagée

4.1.1 Les différents niveaux d'actions

Les propriétés temporelles d'un système temps réel sont obtenues à partir d'une combinaison complexe des besoins de l'application, de l'intergiciel utilisé, de l'infrastructure logicielle d'exécution (système d'exploitation et protocoles de communication) et enfin de l'infrastructure matérielle d'exécution (type d'ordinateur, type de réseau et topologie de distribution). Ainsi la conception d'un système distribué temps réel utilisant les services d'un intergiciel implique de répondre à différentes questions relatives aux différents niveaux d'actions :

1. **Le niveau matériel** : Quel matériel est à notre disposition ? Quelle est la topologie du réseau ? Ces ressources matérielles sont elles assez performantes pour répondre aux attentes de l'application considérée ? ...
2. **Le niveau logiciel** : Quelles sont les caractéristiques du langage de programmation choisi ? Quel système d'exploitation convient le mieux ? Quel protocole réseau est le plus adapté ? ...
3. **Le niveau intergiciel** : Quel type d'intergiciel souhaitons-nous utiliser ? Quels mécanismes

opératoires et quels algorithmes sont mis en jeu ? Quels services doit-il offrir aux différentes applications interconnectées ? ...

4. **Le niveau application** : Qu'est ce que l'on souhaite modéliser ? Quels types de tâches sont mises en jeu ? ...
5. **Le niveau formel** : Quelles méthodes formelles peuvent être appliquées afin de vérifier que le système se comportera conformément aux exigences du concepteur ? ...

Les supports matériels et logiciels utilisés pour supporter les expérimentations sur nos simulations de systèmes embarqués sont décrits dans la partie suivante 4.1.2. Le niveau application relève des cas tests envisagés et utilisés lors des expérimentations et ces cas tests sont décrits dans la partie 4.2. Les modes d'exécutions mis en œuvre pour les simulateurs de ces cas tests sont décrits plus précisément dans la partie 4.3. Comme nous l'avons vu dans la partie sur la norme RT CORBA (*C.f.* partie 3.2.2, page 50), un modèle formel est utile voir indispensable pour valider une application temps réel en fonction des modes d'exécution. Les différents modèles formels utilisés ainsi que les méthodes de validation correspondantes constituent une autre contribution de ce travail de thèse et cela est décrit dans la suite de ce manuscrit (*C.f.* chapitre 6, page 113).

4.1.2 Choix de la plate-forme PRISE

Du point de vue matériel, la plate-forme PRISE est constituée de 4 nœuds temps réel (un serveur et trois clients) équipés de processeurs AMD Opteron à 6 cœurs. Il y a aussi 2 stations graphiques HP équipées avec des processeurs Intel Xeon ainsi que des cartes graphiques GP-GPU de haute performance. Cet ensemble de matériels informatiques est interconnecté par un réseau dédié avec un commutateur Ethernet Gigabit (HP Procurve Switch 1800 Series). La plate-forme PRISE comprend aussi deux organes d'entrée (Yoke/Throttle/Pedals Joysticks) qui permettent à l'utilisateur d'interagir avec les équipements dans le cas des simulations *Human-in-the-loop*.

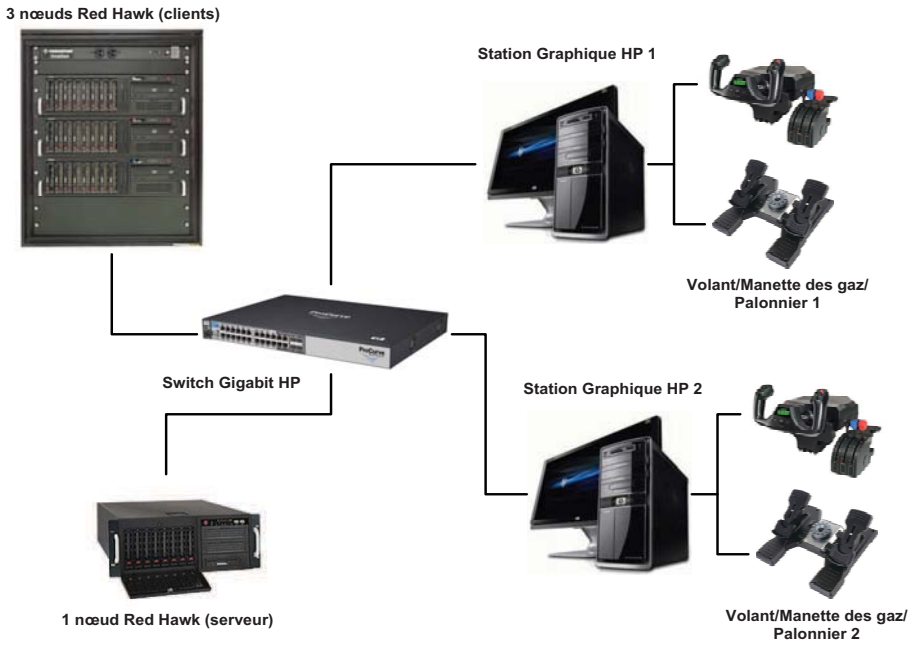


FIG. 4.1 – Plate-forme PRISE

Du point de vue logiciel, les 4 nœuds temps réel sont équipés du système d'exploitation Linux Red Hawk [BKBH08] respectant la norme Posix 1003.1.B [IEE93b, IEE95c]. Rappelons que ce système d'exploitation a aussi été choisi par le laboratoire hollandais TNO[1] pour leurs expérimentations sur des simulations HLA temps réel [Jan04]. Un des avantages de ce système d'exploitation est que le distributeur fournit aussi un ensemble de logiciels nommé Nightstar tools [Con03] très utile pour la configuration, la gestion et la vérification du comportement du système Red Hawk (assignation de priorité, surveillance de l'exécution d'un processus, ...). Les stations HP, quant à elles, sont équipées de systèmes d'exploitation Linux Fedora et Windows XP. L'utilisateur peut choisir l'un ou l'autre des systèmes d'exploitation selon les caractéristiques de son application. Les communications seront assurées par les protocoles classiques UDP et TCP sur le réseau Ethernet dédié. Nous supposons que ces protocoles sont assez performants pour les besoins de nos simulations. En effet, la faible latence de ces protocoles sur notre réseau LAN lié à l'utilisation d'un switch permet de négliger les éventuels problèmes de collisions du protocole MAC CSMA/CD (et donc le non déterminisme du dit protocole).

Le principal avantage des systèmes Red Hawk est une technologie de distribution d'horloge nommé RCIM[2] [Con01]. C'est une carte PCI conçue spécialement pour des applications temps-critiques qui permet à des systèmes distribués Red Hawk d'être synchronisés par une horloge commune. La technologie RCIM distribue cette horloge commune à tous les nœuds Red Hawk par le biais d'une connexion par fibre optique. Cela permet d'avoir une précision d'horloge de l'ordre de la nanoseconde qui est, par conséquent, très adaptée au contexte des études sur les systèmes embarqués.

4.1.3 Utilisation du CERTI pour le temps réel

Sur la plate-forme PRISE, l'intergiciel choisi pour l'interconnexion des différents systèmes informatiques au sein d'une simulation globale est le CERTI (*C.f.* partie 2.6, page 35). Le CERTI est une implémentation des spécifications de la norme HLA qui répond aux spécifications de la norme 1.3 et partiellement à celles de la norme 1516. Comme nous l'avons vu précédemment, la norme HLA n'a pas été initialement conçue pour répondre aux spécificités des systèmes temps réel. En conséquence, dans sa version originale, le CERTI ne possède pas de mécanisme pour la prise en compte de paramètres de qualité de service et aucun outil pour évaluer une prédictibilité de bout en bout. Par exemple, il ne permet pas de considérer certaines applications (certains fédérés) selon une priorité donnée au niveau du système d'exploitation. De plus, le CERTI est implémenté en `C++` et, bien que ce langage soit souvent utilisé dans le contexte d'applications temps réel pour des raisons de performance, il possède néanmoins certaines limitations telles que les mécanismes d'allocation dynamique de mémoire qui sont indéterministes. En revanche, nous avons un contrôle complet sur l'implémentation du CERTI. Nous maîtrisons le code source de cet intergiciel et nous pouvons intégrer des modifications. Ainsi, nous pouvons faire évoluer ses principes de fonctionnement, mettre à jour certains services existants et ajouter des nouveaux services utilisables par une application temps réel. Les extensions de l'intergiciel CERTI, proposées dans le cadre de cette thèse, sont présentées un peu plus loin dans ce manuscrit (*C.f.* partie 5.1, page 89).

Néanmoins, tous les services fournis par l'intergiciel HLA ne doivent pas nécessairement respecter des contraintes de déterminisme. En effet, les phases de création, d'initialisation, de terminaison ainsi que de destruction (décrites dans le tableau 2.8 de la partie 2.4.4.7 page 29) n'ont pas besoin de vérifier de contraintes temporelles. En considérant qu'aucun fédéré ne

[1] *Netherlands Physics and Electronics Laboratory*
[2] *Real-time Clock & Interruption Module*

peut se rajouter durant l'exécution de la simulation et que l'ensemble des fédérés sont pris en compte durant les phases de création et d'initialisation, nous n'avons pas besoin de vérifier le déterminisme temporel des services sous-jacents. Dans notre cas, nous considérons donc qu'une simulation HLA temps réel ne doit vérifier des contraintes de prédictibilité que lors de la phase de la boucle de simulation. Les services concernés par la phase de simulation et appelés par le fédéré sont `updateAttributesValues()`, `sendInteraction()` et les callbacks appelés par la RTI `reflectAttributesValues()`, `receiveInteraction()` pour la famille des services de gestion des objets (*C.f.* partie 2.4.4.4, page 28) ; les services `timeAdvanceRequest()`, `timeAdvanceRequestAvailable()` `nextEventRequest()`, `nextEventRequestAvailable()` et le callback `timeAdvanceGrant()` pour la famille des services de gestion du temps (*C.f.* partie 2.4.4.4, page 28). Les services de gestion de la distribution des données (*C.f.* partie 2.4.4.6, page 29) ainsi que les services de gestion de la propriété de données (*C.f.* partie 2.4.4.5, page 29) sont des services optionnels mis en œuvre dans le CERTI, mais que nous n'utiliserons pas pour nos simulations temps réel.

Le service `tick()` constitue un cas particulier qui permet au fédéré utilisant ce service de rendre la main à la RTI afin de lui permettre d'invoquer les callbacks attendues. Les deux versions de ce service disponibles dans la norme posent des problèmes pour les simulations de systèmes embarqués. Ce problème est expliqué plus en détail dans la suite du manuscrit (*C.f.* partie 5.1.4, page 96). Nous présenterons aussi la solution envisagée avec la création d'un nouveau service `tick2()` bloquant et disponible dans la librairie du CERTI.

4.1.4 Bilan

A ce stade, nous avons donc les premiers éléments de réponse pour plusieurs niveaux d'actions : matériel, logiciel et intergiciel. En effet, la plate-forme dédiée PRISE est composée d'un ensemble de matériels informatiques de haute performance sur lesquels sont installés des supports logiciels adaptés. Il y a aussi des équipements permettant de faire intervenir un opérateur humain dans la simulation ainsi que des technologies de distribution d'horloge pour assurer des synchronisations optimales. Cette plate-forme propose donc un support modulaire et adapté pour utiliser le CERTI afin d'interconnecter différents systèmes embarqués réels et/ou simulés. La figure 4.2 résume les acquis de ces différents niveaux.

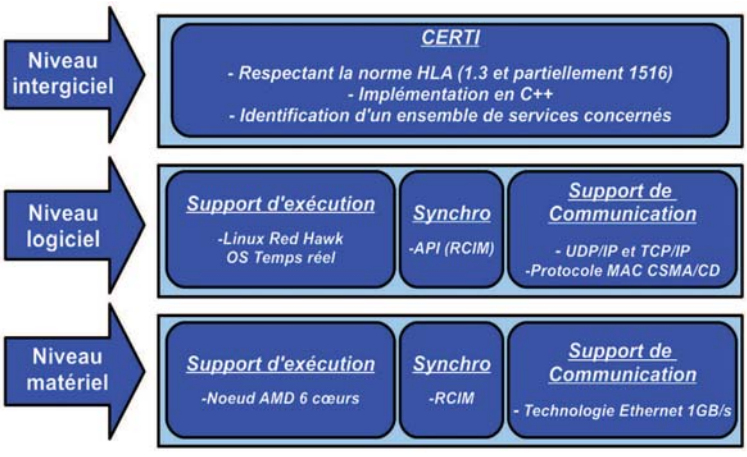


FIG. 4.2 – Réponses des niveaux matériel, logiciel et intergiciel

4.2 Description des cas tests

4.2.1 Vol en formation de satellites

Les premiers travaux d'investigation sur l'utilisation du CERTI en tant qu'intergiciel dans le contexte de l'interconnexion des systèmes embarqués proviennent d'une étude collaborative entre le laboratoire du CNES[3] et l'ONERA [NdS08]. Le but était d'utiliser le CERTI en tant que système de communication haute fréquence afin d'assurer le vol en formation de deux satellites. Les spécifications décrites dans le document [Gui05] envisagent deux fédérations HLA, une fédération simple et une autre plus complexe. La fédération dite *cas simple* comporte quatre fédérés représentant chacun un simulateur de système embarqué d'un des deux satellites.

- Le fédéré 1 est le modèle du simulateur du calculateur de bord 1 ;
- Le fédéré 2 est le modèle du simulateur de la dynamique du satellite 1 ;
- Le fédéré 3 est le modèle du simulateur de la dynamique du satellite 2 ;
- Le fédéré 4 est le modèle du simulateur du calculateur de bord 2.

Les fédérés 2 et 3 simulant la dynamique du vol répètent un schéma d'exécution selon une fréquence de 100 Hz et mettent donc en œuvre des cycles théoriques de 10 millisecondes. Les fédérés 1 et 4 représentent les calculateurs de bord des deux satellites qui s'exécutent selon une fréquence de 20 Hz et mettent en œuvre des cycles de 50 millisecondes. Les échanges de données entre les fédérés, la taille des données échangées ainsi que la fréquence à laquelle s'effectuent ces échanges sont représentées sur la figure 4.3.

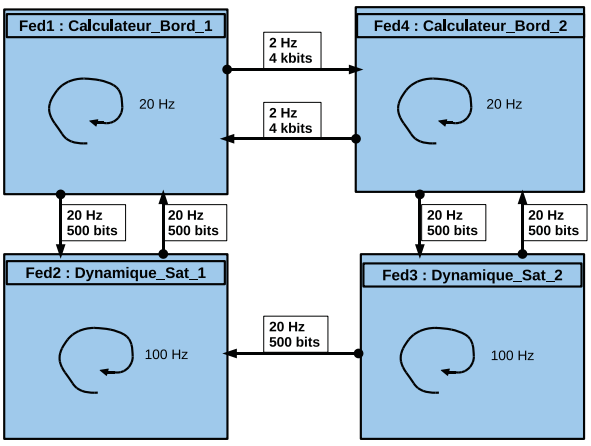


FIG. 4.3 – Fédération CNES (cas simple)

La fédération dite *cas complexe* est comme son nom l'indique un peu plus complexe et comporte, quant à elle, 6 fédérés. Les quatre premiers fédérés conservent le même mode de fonctionnement que les fédérés relatifs à la fédération *cas simple*. Cependant, on accélère la fréquence des communications entre les deux fédérés simulant les comportements des calculateurs de bord qui passe de 2 Hz à 10 Hz. Deux fédérés supplémentaires sont ajoutés :

- Le fédéré 5 est le modèle du simulateur du calculateur de charge utile ;
- Le fédéré 6 est le modèle du simulateur des composants de charge utile.

La fréquence de calcul de ces deux nouveaux fédérés est de 200Hz et ils mettent donc en œuvre des cycles de 5 millisecondes. De la même manière que précédemment, les différents échanges de données entre les fédérés, la taille des données échangées ainsi que la fréquence à laquelle s'effectuent ces échanges sont représentées sur la figure 4.4.

[3] *Centre National d'Etudes Spatiales*

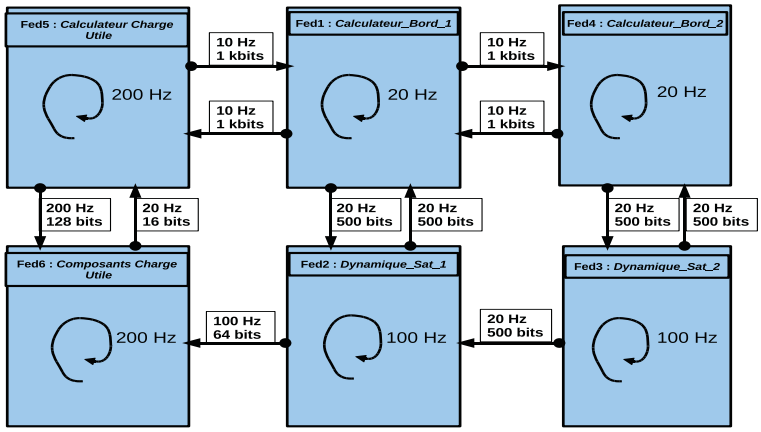


FIG. 4.4 – Fédération CNES (cas complexe)

Cette fédération a initié les études sur les simulations de systèmes embarqués et établit la performance du CERTI dans la tenue de contraintes temporelles. Les premières expérimentations sur ces cas tests ont été réalisées sur des architectures matérielles standards de type PC pour les calculs et des connexions Ethernet simples pour la communication réseau [NdS08]. Cela a permis de mettre en avant différents points clés tels que l'impact des mécanismes opératoires du CERTI dans la tenue d'échéances temporelles très petites (de l'ordre de la dizaine de millisecondes).

En revanche, les spécifications de cette simulation ne contenaient qu'un canevas de conception. C'est-à-dire que nous connaissions les fréquences de calcul des fédérés ainsi que leurs temps de calcul. Nous connaissions aussi les tailles et les fréquences des messages échangés entre ces fédérés au travers du CERTI. Cependant, nous n'avions aucune connaissance sur les différentes spécificités sémantiques des fédérés mis en œuvre pour le vol en formation de satellites. Par exemple, nous ne savions pas quels algorithmes étaient implémentés pour la dynamique du vol. Ainsi, ces fédération permettaient de tester si le respect de contraintes de temps était possible entre des processus s'exécutant et communiquant à haute-fréquence par l'intermédiaire du CERTI. Mais, nous ne pouvions pas vérifier la pertinence des résultats produits et si les erreurs temporelles commises avaient un impact sur la fiabilité des simulateurs. Cette simulation contient uniquement des systèmes embarqués simulés et, par conséquent, ce n'est pas une simulation *Hardware-in-the-loop*. De plus, aucun opérateur humain n'est intégré pour participer à la boucle de simulation globale et donc ce n'est pas une simulation *Human-in-the-loop* non plus.

4.2.2 Une fédération contrôle commande aéronautique

En collaboration avec David Saussié [Sau10], effectuant son contrat post-doctoral sur la plateforme PRISE, nous avons mis en œuvre et testé un simulateur de vol. Il s'agit d'une fédération HLA découpée en différents simulateurs représentant chacun une partie spécifique de l'avion ou de son environnement. Cette première version du simulateur est basée sur les paramètres d'un A320 mais nous souhaitons intégrer un grand nombre d'aéronefs. Au final, cette fédération, illustrée dans la figure 4.5, est composée de neuf fédérés spécifiques :

1. un fédéré Joystick ;
2. un fédéré Contrôleur ;
3. un fédéré Actuateur MecaVol ;
4. un fédéré Actuateur Moteur ;

5. un fédéré Dynamique du Vol ;
6. un fédéré Capteurs ;
7. un fédéré Visu 3D ;
8. un fédéré Visu PFD[4] ;
9. un fédéré Environnement.

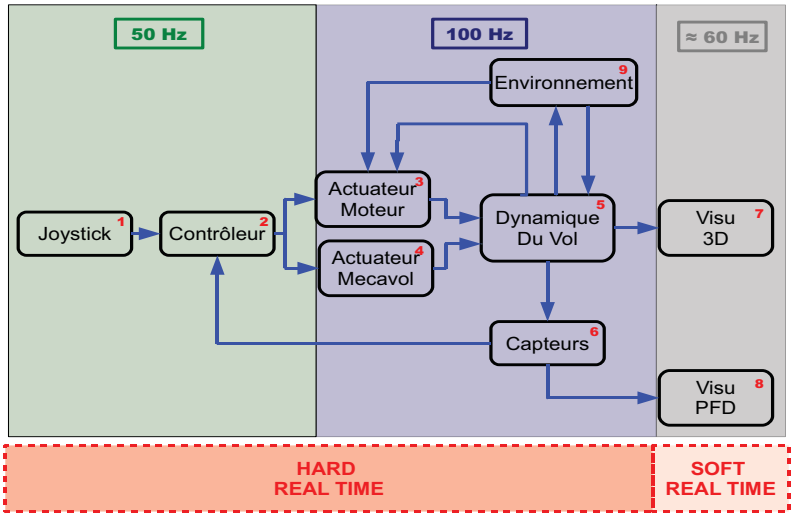


FIG. 4.5 – Fédération Contrôle-Commande

La fédération est composée de deux types d'applications selon les exigences des contraintes temps réel :

1. La première partie est concernée par des contraintes de temps réel dur (*Hard real time*) et doit s'assurer que toutes les contraintes temporelles seront respectées par chaque fédéré. Le fédéré récupérant les ordres du pilote ainsi que le fédéré du contrôleur de vol fonctionnent à une fréquence de 50 Hz (une période de 20 millisecondes). Cela correspond à une fréquence moyenne usuelle dans le monde des systèmes avioniques. Les autres fédérés (moteurs, actionneurs, capteurs et dynamique du vol) travaillent à une fréquence de 100 Hz (soit une période correspondante de 10 millisecondes). Ces fédérés simulent des systèmes en temps continu qui sont modélisés par des équations différentielles. Ces équations différentielles sont résolues par des méthodes numériques durant l'exécution de la simulation. Une fréquence de 100 Hz nous paraît suffisante pour fournir des résultats fiables. Nous pourrions éventuellement augmenter cette fréquence tout en prenant soin de respecter les nouvelles contraintes temps-réel.
2. La deuxième partie est concernée par des contraintes temps réel mou (*Soft real time*). Ici, l'objectif est de répondre à un sous-ensemble de contraintes en respectant un certain taux de réussite. Dans notre cas, le fédéré de visualisation 3D ainsi que le fédéré PFD travaillent avec une fréquence de 60 Hz afin d'assurer un taux moyen de rafraîchissement de l'image cohérent pour les limites physiologiques des yeux humains (cette problématique a été traitée dans la partie 3.3.1, page 55).

[4] *Primary Fligh Display*

4.2.2.1 Le fédéré de dynamique du vol

Le fédéré chargé des calculs relatifs à la dynamique du vol de l'avion représente le cœur du modèle pour le simulateur de vol. Il implémente les différentes équations du mouvement et fait évoluer l'état de l'avion sous l'action des forces aérodynamiques, du poids et de la poussée des moteurs. Les équations du mouvement [Nel98, SL03] sont 12 équations différentielles ODE[5] séparées classiquement : 3 pour la position, 3 pour l'orientation, 3 pour la vitesse de translation et enfin 3 pour les vitesses angulaires. Ce sont ces équations qui sont résolues pendant l'exécution de la simulation et peuvent être décrites sous la forme :

$$\begin{cases} \dot{x}(t) &= f\big(x(t), u(t), t\big) \\ x(0) &= x_0 \\ y(t) &= g\big(x(t), u(t), t\big) \end{cases}$$

La première équation est l'équation d'état du système, la troisième équation est l'équation de sortie avec comme variables le vecteur d'état $x \in \mathbb{R}^n$ et $x(0)$ la condition initiale sur cet état, le vecteur d'entrée $u \in \mathbb{R}^m$, le vecteur de sortie $y \in \mathbb{R}^p$ et le temps continu t. Dans le cas général, les fonctions $f(.)$ et $g(.)$ désignent des fonctions vectorielles non linéaires. Fondamentalement, l'équation d'état décrit le comportement dynamique du système et donc sa description interne c'est-à-dire les évolutions du vecteur d'état en fonction des entrées. L'équation de sortie permet de produire les variables de sortie (c'est-à-dire les attributs qui seront publiés au reste de la fédération) en se basant sur les états du système et, éventuellement, sur les entrées. Afin de résoudre l'équation d'état, nous pouvons utiliser des méthodes d'intégration différentes telles que les méthodes explicites d'Euler ou encore des méthodes prédicteur/correcteur de Heun. Ces différentes méthodes sont implémentées directement dans le code du fédéré. Par exemple, si on utilise une discrétisation selon la méthode d'Euler, on obtient les équations discrètes de la forme :

$$\begin{cases} x_{n+1} &= x_n + \Delta t * f\big(x_n, u_n, n\Delta t\big) \\ x_0 &= ConditionInitiale \\ y_n &= g\big(x_n, u_n, n\Delta t\big) \end{cases}$$

La qualité des résultats produits dépend donc du choix de la méthode de résolution ainsi que du pas de temps utilisé pour la discrétisation (le paramètre Δt). Ainsi, en fonction de l'ordre de la méthode et de la valeur du pas de temps, les temps de calcul nécessaires aux différents calculs peuvent être considérablement augmentés. Bien entendu, plus on choisit un pas de temps Δt petit et meilleure est la précision de la résolution du système d'équations. Il faut donc prendre soin de déterminer correctement ces paramètres afin d'obtenir une solution adéquate tout en assurant le respect des contraintes temps réel dur auxquelles est soumis ce fédéré.

[5] *Ordinary Differential Equations*

4.2.2.2 Les fédérés actuateurs

Le fédéré *Actuateur MecaVol* rassemble des simulateurs pour tous les actionneurs des surfaces de contrôle qui influencent le mouvement de l'avion. Nous considérons ici des actionneurs pour les ailerons à gauche et à droite, des actionneurs pour les gouvernes de profondeur à gauche et à droite ainsi qu'un actionneur pour la gouverne de direction. La dynamique de chaque actionneur d'une surface de contrôle est modélisée par un système du second ordre incluant une saturation en position et en vitesse pour assurer le réalisme du modèle. D'autres phénomènes tels que le retard ou l'hystérésis[6] peuvent également être activés pour accentuer le réalisme du modèle. Ce fédéré reçoit les ordres de déviation angulaire publiés par le fédéré contrôleur et il publie ensuite ses données pour le fédéré de dynamique du vol.

Le fédéré *Actuateur Moteur* simule deux turboréacteurs (gauche et droite) dont le modèle a été repris d'un modèle Matlab/Simulink et ré-implémenté en `C++`. C'est un modèle simple car la modélisation d'un moteur reste un problème très complexe. Chaque avion que nous souhaitons simuler nécessitera l'élaboration d'un modèle particulier pour ses moteurs. Ce fédéré reçoit les ordres de commandes de gaz publiés par le fédéré contrôleur, les conditions atmosphériques (pression, température et densité) du fédéré environnement ainsi que le nombre de Mach du fédéré de dynamique du vol. Il réalise ensuite ses calculs internes pour publier ses données pour le fédéré de dynamique du vol.

4.2.2.3 Le fédéré capteurs

Pour effectuer sa tâche et donc assurer le vol de l'avion, le contrôleur de vol a besoin de différentes mesures sur l'état de l'avion. Ces différentes mesures sont réalisées par des capteurs qui permettent de rendre ces informations disponibles pour le contrôleur. Le fédéré capteurs simule donc différents capteurs tels que des IRU[7], IMU[8] ou encore ADIRU[9]. Les capteurs ayant leur propre dynamique, ils sont modélisés et simulés comme des filtres de Butterworth [But30] selon un ordre et une fréquence de coupure donnés. Des phénomènes additionnels tels que le délai, la dérive, le bruit ou la quantification sont également implémentés et peuvent aussi être mis en œuvre. Ce fédéré souscrit à certains attributs du fédéré de dynamique du vol et publie des données pour le fédéré de visualisation PFD ainsi que pour le fédéré contrôleur.

4.2.2.4 Le fédéré contrôleur

Le fédéré contrôleur est en charge du contrôle de l'aéronef selon des lois de commande précises. Les fonctions classiques du pilote automatique, telles que le maintien d'une vitesse ou d'une altitude, ont été implémentées. D'autres fonctionnalités de commande de vol, telles que les systèmes de contrôle de la stabilité, sont aussi implémentées. L'avion peut alors fonctionner en mode manuel et un utilisateur peut interagir avec la simulation en actionnant le Joystick Yoke/Throttle. Ou encore, il peut être basculé en mode automatique où aucune intervention du pilote n'est nécessaire au déroulement de la simulation. Contrairement aux autres fédérés de la fédération, qui simulent des processus physique en temps continu, le contrôleur de vol est prévu pour fonctionner sur une plate-forme numérique avec une période d'exécution connue a priori. L'ingénieur doit alors choisir une période de temps compatible avec le matériel et le

[6] *L'hystérésis est le retard de l'effet sur la cause, la propriété d'un système qui tend à demeurer dans un certain état quand la cause extérieure qui a produit ce changement d'état a cessé*

[7] *Inertial Reference Unit*

[8] *Inertial Measurement Unit*

[9] *Air Data Inertial Reference Unit*

logiciel et assurer que le contrôleur numérique garantissent la stabilité et la performance de la boucle fermée. Les transformations des représentations du système en temps continu à temps discret sont assurées par des transformations de Tustin couramment utilisées pour implémenter les contrôleurs discrets [Oga94]. Le fédéré contrôleur reçoit les commandes du pilote envoyées par le fédéré Joystick ainsi que les mesures émises par le fédéré Capteurs. Il réalise ensuite ses différents calculs internes et peut envoyer l'ordre de déflection (ordre de braquage) pour le fédéré Actuateur MecaVol (gérant les surfaces de contrôle) ainsi que les ordres de commandes de gaz pour le fédéré Actuateur Moteur.

4.2.2.5 Le fédéré environnement

Le fédéré environnement modélise différents paramètres relatifs à l'environnement dans lequel se trouve l'avion durant son vol (atmosphère, vent,...). Le modèle de l'atmosphère est basé sur le modèle US Standard Atmosphere de 1976 [Nat76] qui est valable pour des altitudes comprises entre 0 et 85 kilomètres. En se basant sur la donnée d'altitude qui lui est fournie par le fédéré de dynamique du vol, il calcule les variables d'atmosphère correspondantes telles que la température, la pression, la densité de l'air ainsi que la vitesse du son. Différentes caractéristiques du vent sont implémentées (cisaillements et rafales) ainsi que des modèles de turbulences tels que ceux de Dryden [DA49, Yea98] ou de Von Karman [VK48]. Des descriptions complètes des deux types de turbulences sont disponibles sur le site de Mathworks (le distributeur de MATLAB) [16, 79]. Le fédéré environnement souscrit aux données publiées par le fédéré de dynamique du vol pour récupérer l'altitude, il peut ensuite fournir les données atmosphériques correspondantes à cette altitude à ce même fédéré de dynamique du vol ainsi qu'au fédéré Actuateur Moteur.

4.2.2.6 Les fédérés de visualisation

Les paramètres calculés par les différents capteurs sont utilisés par le fédéré PFD[10]. Ce fédéré permet l'affichage des informations essentielles à un pilote lors du vol de l'avion : la vitesse, la vitesse verticale, l'attitude ou encore le cap. Ainsi, comme s'il se trouvait dans un véritable cockpit (*C.f.* figure 4.6), l'utilisateur peut avoir une vision du comportement de l'avion sur différentes jauges programmées en utilisant la librairie Qt [55].

FIG. 4.6 – Illustration des différents instruments présents dans un cockpit d'avion

[10] *Primary Fligh Display*

Le fédéré de visualisation 3D permet à l'utilisateur de visualiser le comportement de l'aéronef (position et orientation) dans un environnement graphique virtuel 3D. Le projet VirtualAir [77] fournit des plug-ins pour intégrer différents simulateurs comme des composants logiciels d'une fédération HLA. Ansi, nous pouvons utiliser des simulateurs tels que Flight Simulator [22], FlightGear [21] et X-plane [86] pour visualiser l'avion dans un environnement virtuel.

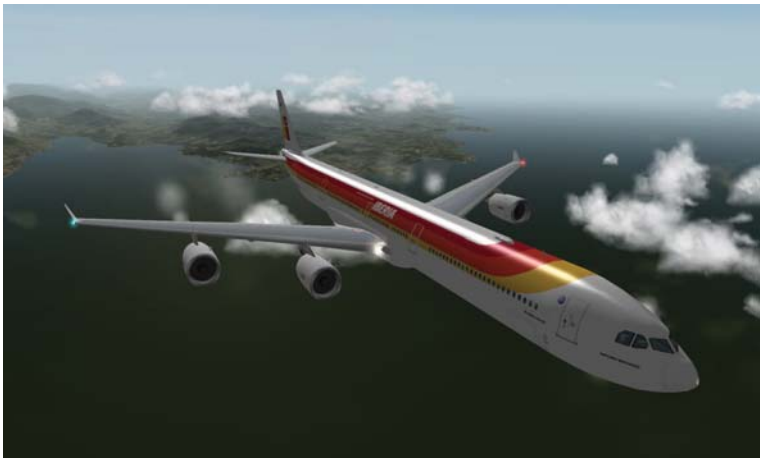


FIG. 4.7 – Visualisation d'un airbus A340 en vol avec le logiciel Xplane

4.2.2.7 Le fédéré joystick

Enfin, le pilote interagit avec la simulation en utilisant des joysticks particuliers permettant de commander un volant, une manette des gaz et un palonnier *i.e.* Yoke/Throttle/Pedals (*C.f.* figure 4.8). Les actions de l'utilisateur sont ensuite transmises par le fédéré pour le fédéré contrôleur. Actuellement, l'utilisateur peut piloter l'avion selon les trois axes (tangage, lacet et roulis), il peut aussi agir sur les moteurs par le biais de la manette de gaz. La poussée du moteur n'est cependant pas directement contrôlée par le pilote car c'est le contrôleur qui va assurer le respect des ordres du pilote en fonction des lois de commande. Le pilote peut aussi activer le pilote automatique en appuyant sur un bouton spécifique du joystick.

FIG. 4.8 – Systèmes de commande Volant/Manette des gaz/Palonnier

4.2.2.8 Bilan et perspectives

Ce simulateur de vol est un premier pas qui ouvre la voie vers de nombreuses études. En effet, ici, nous avons la maîtrise du code source du point de vue des spécificités sémantiques de

l'application (équations de la dynamique du vol, lois de commande, ...). Et, nous avons aussi la maîtrise du code en ce qui concerne la programmation système avec un découpage de l'application sous forme de fédérés HLA et la définition d'un modèle objet commun pour la communication. Nous pouvons donc faire évoluer cette simulation à notre guise en enrichissant les fédérés existants ou en intégrant des nouveaux fédérés (fédéré de configuration, fédéré pour simuler des pannes, ...). Nous pouvons aussi remplacer un fédéré existant par un fédéré extérieur qui remplit la même fonction dans le cas d'une collaboration avec un partenaire extérieur. Par exemple, nous pourrions tester d'autres types de moteurs venant d'un industriel. L'avantage de l'utilisation de la norme HLA est que l'industriel ne devra pas expliciter les spécificités techniques de son code mais il devra simplement spécifier les données qu'il souhaite publier et recevoir. Cela lui permettra ainsi de participer à un projet de recherche tout en conservant la propriété intellectuelle sur ses codes de calculs.

4.3 Description des modes d'exécution

4.3.1 Vers des modes d'exécution périodiques

Comme nous l'avons vu, les fédérés mis en œuvre dans les cas tests sont basés sur un mode de fonctionnement répétitif selon une fréquence donnée. Ainsi, nous allons nous baser sur le mode d'exécution périodique issu des travaux de Fujimoto et McLean (*C.f.* la partie 3.4.1, page 60). Dans ce cas, chaque simulateur (un fédéré HLA) engagé dans ce type de simulation répète, périodiquement, le même schéma d'exécution avec un pas de temps noté Δ_t. Le modèle décrit par Fujimoto et McLean est basé sur l'exécution de quatre phases successives : une phase de réception, une phase de calcul, une phase de transmission et une phase de temps libre. Les travaux conjoints de l'ONERA et du CNES ont mis en évidence la nécessité d'ajouter explicitement une phase de synchronisation aux autres phases pour chaque étape de l'exécution du fédéré afin de maintenir la cohérence entre les cycles de chaque simulateur. Ainsi, pour nous, un mode d'exécution de fédéré périodique est composé de cinq phases successives : (1) une phase de synchronisation, (2) une phase de réception, (3) une phase de calcul, (4) une phase de transmission et (5) une phase de temps libre. Ce mode d'exécution est illustré sur la figure 4.9.

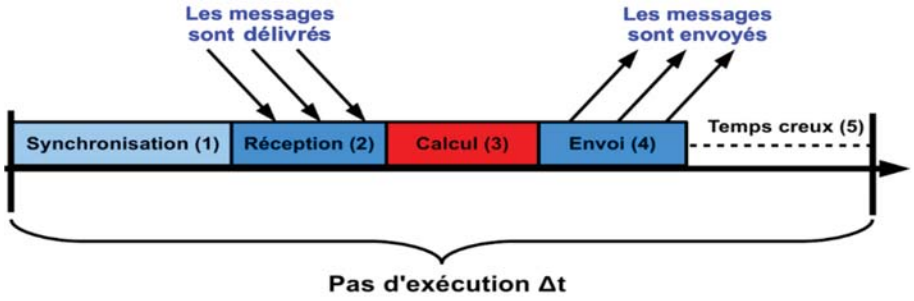


FIG. 4.9 – Les différentes phases d'un fédéré périodique

Les modes d'exécution des simulateurs périodiques se distinguent donc par la technique utilisée afin de réaliser cette synchronisation. Historiquement, dans la norme DIS (*C.f.* partie 2.3.2, page 18), cette phase de synchronisation était faite (pour chaque simulateur) en consultant une horloge globale commune à tous les simulateurs (une horloge GPS ou une horloge synchronisée par le protocole NTP). Dans les travaux de Fujimoto et McLean, cette synchronisation est implicitement faite par l'utilisation des mécanismes de gestion du temps avec lesquels la phase de synchronisation et la phase de réception des données sont réalisées en même temps (*C.f.* partie

3.4.2, page 62). L'étude CNES présente aussi un autre mécanisme de synchronisation original par l'envoi d'une interaction, appelée *pulsation*, par le fédéré le plus rapide qui rythme ainsi les autres fédérés de la simulation. Pour résumer, la phase de synchronisation peut se faire par trois méthodes différentes qui distinguent les différents modes d'exécution des fédérés :

1. Par consultation de l'horloge commune (WCT) telle que l'horloge matérielle si tous les fédérés sont situés sur un seul nœud (mono ou multi-processeur) ou encore la technologie RCIM pour un système distribué tel que l'ensemble des nœuds Red Hawk de la plate-forme PRISE. Ce mode d'exécution, nommé *Data Flow*, est décrit dans la partie 4.3.2.
2. Le fédéré qui a le cycle le plus court peut envoyer une interaction à tous les autres afin de rythmer l'exécution de tous les fédérés impliqués dans la fédération. Ce mode d'exécution est décrit dans la partie 4.3.3.
3. L'utilisation des mécanismes de gestion du temps HLA pour assurer la remise des messages et la synchronisation des cycles entre tous les fédérés. Nous avons distingué plusieurs types de modes d'exécution basés sur l'utilisation des services de gestion du temps :
   - le mode d'exécution *Time-Stepped* est décrit dans la partie 4.3.4 ;
   - le mode d'exécution *Event-Driven* est décrit dans la partie 4.3.5 ;
   - un mode mixte mis en exergue par la résolution explicite d'équations différentielles ; ce mode particulier est présenté dans la partie 4.3.6.

4.3.2 Le modèle *Data Flow*

Ce type de fédéré communique en utilisant les fonctionnalités de base de la norme HLA à savoir les services de publication et souscription. Pour cela, le fédéré fait des appels aux services fournis par la RTI comme `updateAttributeValues()` pour publier des données. Inversement, il recevra les données attendues par l'utilisation de la callback `reflectAttributeValues()` dans la phase de réception. Il devra préalablement demander explicitement à la RTI de recevoir cette callback en utilisant le service `tick()`. Une fois que les données attendues sont reçues, chaque fédéré exécute alors son algorithme local (sa phase de calcul) et il peut publier ensuite sa nouvelle donnée. Cependant, afin d'assurer la cohérence de l'exécution de ses cycles avec l'exécution du reste de la fédération, il doit se synchroniser en consultant l'horloge globale (phase de synchronisation). Ce mode d'exécution périodique est illustré sur la figure 4.10.

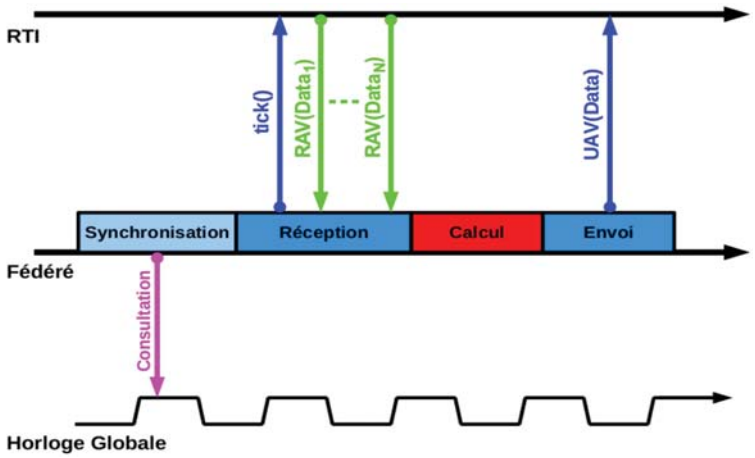


FIG. 4.10 – Modèle d'exécution par consultation d'une horloge globale

Le principal intérêt de ce mode d'exécution est sa simplicité de fonctionnement. De cette façon, la modélisation de son comportement avec un modèle formel compatible avec les politiques

et les techniques d'ordonnancement temps-réel classiques est plus évidente (*C.f.* partie 6.2, page 117). En revanche, les développeurs doivent assurer une programmation cohérente afin que les fédérés attendent et reçoivent les données relatives aux différents cycles au bon moment. Une erreur de conception peut mener à différents types de problème tels que la réception de mauvaises données pour le cycle en cours ou encore l'inter-blocage de la fédération globale. De plus, cette approche est plus difficile à mettre en œuvre pour l'ajout de nouveaux fédérés à une fédération car il faut garantir une horloge globale commune précise à tous les fédérés de la fédération ayant ce mode de fonctionnement. Si l'on souhaite ajouter des fédérés venant de l'extérieur (collaboration avec un industriel par exemple), il faudra conserver la cohérence temporelle entre la partie de la fédération locale et les nouveaux fédérés extérieurs et on peut être confronté à des phénomènes de divergence des horloges. En ce sens, la performance de l'application dépendra fortement de la précision de l'horloge globale établie entre les différents nœuds du système distribué. Par exemple, les synchronisations par le protocole NTP (dans sa version 3) ayant une précision de l'ordre de 30 millisecondes paraissent inadaptées pour synchroniser des simulateurs de systèmes embarqués avec des contraintes temporelles petites (de l'ordre de la dizaine de millisecondes). Il serait, dans ce cas, difficile de garantir les échéances pour des fédérés périodiques avec des cycles de 10ms ou 5ms tels que ceux de l'application CNES. En revanche, la technologie RCIM propose des latences de synchronisation de l'ordre de la nanosecondes adaptées aux contraintes de nos simulations. Cependant, l'horloge commune RCIM est uniquement disponible entre des nœuds Red Hawk peu distants (la distance entre deux nœuds doit être inférieure à 30 mètres). Actuellement, les protocoles de synchronisation des horloges locales tels que NTP ou PTP deviennent de plus en plus performants (*C.f.* partie annexe B.4, page 170). Des études connexes à cette thèse pourraient être menées pour utiliser ce mode d'exécution dans des simulations distribuées dirigées par le temps (*C.f.* partie B.4, page 170)

Ce mode de fonctionnement a été mis en œuvre et testé lors d'expérimentations récentes de l'application CNES sur la plateforme PRISE [CSAN11]. Il a aussi été utilisé pour l'implémentation du simulateur de vol toujours sur la plateforme PRISE [CSSA11, CSSA12]. Ce mode de fonctionnement est très efficace en terme de performance. Nous présenterons ces différents résultats de performance dans la suite du document (*C.f.* chapitre 7.2.3, page 147).

4.3.3 Le modèle synchronisé par interaction

Dans le cas d'une synchronisation par interactions, c'est le fédéré avec la période la plus petite qui rythme l'exécution globale de la fédération en servant d'horloge commune. Pour cela, il envoie une interaction `TOP`, par l'appel au service `sendInteraction()`, à chacun de ses cycles. Son cycle sera alors considéré comme le cycle de base de la fédération. Cette interaction est reçue par les autres fédérés de la fédération par la callback `receiveInteraction()` et ainsi chaque fédéré connait le cycle de base en cours. On suppose donc pour simplifier que tous les cycles des fédérés sont des multiples du cycle de base. Nous pouvons aussi imaginer l'utilisation d'un fédéré spécialisé uniquement dédié à la production de cette pulsation pour l'ensemble de la fédération. Les fédérés qui reçoivent une interaction `TOP` pour la synchronisation peuvent aussi consulter leurs horloges locales afin d'éviter des divergences de synchronisation entre le cycle de base et leurs cycles de calcul. Ce mode d'exécution périodique est illustré sur la figure 4.11.

Ce modèle de synchronisation peut être utilisé dans des simulations qui ne possèdent pas une horloge commune partagée par tous les fédérés participant à l'exécution de la fédération. Cela permet de synchroniser une fédération distribuée sans horloge commune et sans mettre en œuvre les mécanismes de gestion du temps. Par exemple, dans certains RTIs commerciaux l'utilisation de chaque type de service est payante. Le concepteur d'une fédération composée

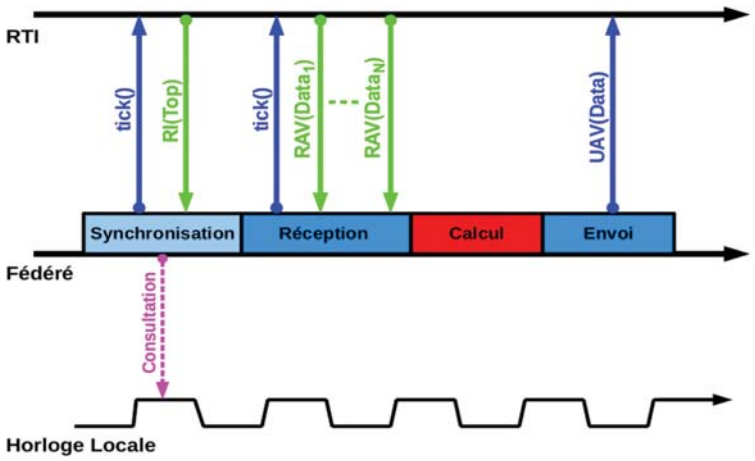


FIG. 4.11 – Modèle d'exécution par envoi d'une interaction `TOP`

de fédérés périodiques pourrait utiliser cette technique qui implique uniquement les services de gestion des objets. En revanche, cette technique a aussi ses inconvénients. Par exemple, l'impact de la distribution topologique des fédérés implique qu'un fédéré placé sur la même machine que le fédéré *pulsation* recevra l'interaction avant les fédérés situés plus loin topologiquement. De même, les fédérés d'un même nœud recevront les interactions de synchronisation en même temps et cela peut poser un problème sur la façon de séquencer les calculs ensuite car tous ces fédérés souhaiteront accéder aux ressources en même temps (le processeur ou les processeurs du nœud).

Ce mode d'exécution a été mis en œuvre et testé lors des expérimentations initiales de l'application CNES sur des composants matériels et logiciels standards. Les résultats avaient été encourageants mais néanmoins ce mode de fonctionnement paraissait moins bien séquencer les différents calculs que l'utilisation des mécanismes de gestion du temps (selon le mode d'exécution *Time-stepped* vu dans la partie 4.3.4 suivante). Le lecteur intéressé pourra se référer à l'article [NdS08] pour plus de précisions notamment sur les différentes mesures de performance. Nous n'utiliserons pas ce mode de fonctionnement dans la suite de nos travaux, ni pour nos expérimentations.

4.3.4 Le modèle *Time-stepped*

L'utilisation des mécanismes de gestion du temps met en place une échelle de temps logique (ou temps simulé) qui représente une sorte d'échéancier sur lequel on peut avoir une chronologie des différents événements de la simulation. Cette échelle de temps est partagée par tous les fédérés afin qu'ils puissent déterminer leur temps local (l'instant du temps simulé dans lequel le fédéré se trouve) et le temps des événements qu'ils produisent (la date de validité de l'événement). Ici, le fédéré reçoit et publie de l'information donc, par conséquent, il est régulateur et contraint selon les principes de la norme HLA (ces concepts ont été expliqués précédemment dans la partie 2.5.1, page 31). Dans ce cas, le fédéré demande une avance à l'instant ts du temps simulé en appelant le service `timeAdvanceRequest(`ts`)`. En retour, lorsque l'avancement des autres fédérés participant à la fédération sera sûr (garantie du respect de la causalité), la RTI délivrera tous les messages avec une estampille inférieure à ce temps ts (utilisation de la callback `reflectAttributesValues()` pour chaque événement estampillé avec un $t_i < ts$). Ensuite, la RTI accordera explicitement au fédéré cette avance dans le temps simulé ts par la callback `timeAdvanceGrant(`ts`)`. Dans ce cas, la phase de synchronisation et la phase de réception sont donc réalisées dans le même temps. Ici, le temps simulé reste une représentation du temps réel. Le

fédéré s'exécutant avec une période de temps réel de 10 ms demandera des avancées dans le temps par incrément de 10 unités de temps logique (ou simulé). Le fédéré peut éventuellement vérifier la correspondance entre ce temps logique et son horloge locale afin d'éviter des divergences. Ce mode d'exécution périodique est illustré sur la figure 4.12.

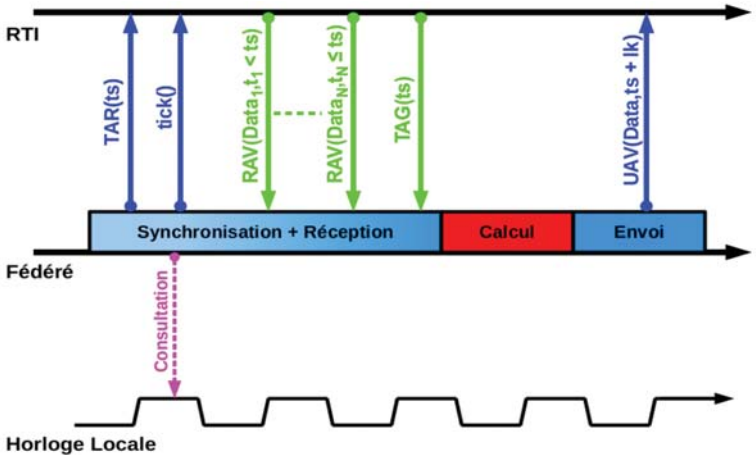


FIG. 4.12 – Modèle d'exécution Time-Stepped (mécanisme de gestion du temps)

L'utilisation des services de gestion du temps nécessite la mise en œuvre d'algorithmes distribués conservatifs complexes issus de la communauté de la simulation à événement discret (*C.f.* la partie 2.2.3.1, page 15). Il faut donc prendre en compte les besoins en ressources nécessaires aux différents calculs et communications de ces mécanismes de gestion du temps. Cela impose aussi que le concepteur des fédérés ayant ce mode d'exécution ait une connaissance suffisante des spécificités des algorithmes conservatifs mis en œuvre dans la RTI utilisée. Par exemple, l'utilisation de l'algorithme de Mattern repose sur des calculs qui peuvent être avortés puis relancés et cela pose un problème pour déterminer les temps d'exécution du fédéré. L'utilisation des algorithmes de première génération comme celui de Chandy, Misra et Bryan (algorithme CMB utilisé dans le CERTI) repose sur l'utilisation de messages réseaux additionnels. Ces messages additionnels sont dit messages *NULL* car ils contiennent uniquement une estampille temporelle du temps simulé et ils peuvent induire une surcharge du trafic réseau et augmenter la latence des messages entre les simulateurs. Cependant, les fédérés *Time-stepped* ne sont pas sensibles au problème du *time creep* car leur avancée dans le temps logique et les échanges additionnels de messages *NULL* ne dépendent pas du choix du paramètre *lookahead*. Ce type de fonctionnement est aussi moins évident à modéliser en utilisant les techniques classiques de la théorie de l'ordonnancement. Pour cela, nous proposons une solution sous différentes hypothèses dans le chapitre suivant (*C.f.* partie 6.2, page 117). Nous proposons, dans cette même partie, un ensemble de métrique pour quantifier l'échange additionnel de messages *NULL* entre les fédérés ayant ce mode de synchronisation.

Ce mode de fonctionnement a été mis en œuvre et testé lors des premières expérimentations de l'application CNES [NdS08] ainsi que lors des expérimentations plus récentes sur la plateforme PRISE [CSAN11]. Ce mode de fonctionnement a aussi fourni de très bon résultats de performance. Il semble que le surcoût induit par les calculs supplémentaires et l'échange additionnel de messages NULL soit compensé par une meilleure synchronisation et un séquencement adéquat des calculs des fédérés. Nous présenterons les résultats de performance plus en détail dans la suite du document (*C.f.* chapitre 5, page 89).

4.3.5 Le modèle *Event-Driven*

Ce mode d'exécution est aussi basé sur les principes des mécanismes de gestion du temps qui permettent de conserver une cohérence temporelle sur une échelle de temps logique (ou temps simulé). Ici aussi, le fédéré reçoit et publie de l'information et il est régulateur et contraint selon les principes de la norme HLA. Pour ce mode de fonctionnement, les fédérés demandent de recevoir le prochain message de simulation disponible entre son temps logique et le temps ts (futur) et sûr du point de vue de la causalité. Pour cela, il appelle le service `nextEventRequest(`ts`)`. En retour, la RTI lui délivrera le premier message disponible et lui accordera une avancée dans le temps à l'estampille t_1 de ce message par la callback `timeAdvanceGrant(`t_1`)` (avec $t_1 \leq ts$). Dans ce cas, la phase de synchronisation et la phase de réception sont réalisées dans le même temps. Le fédéré peut éventuellement vérifier la correspondance entre ce temps logique et son horloge locale. Ce mode d'exécution périodique est illustré sur la figure 4.13. De la même

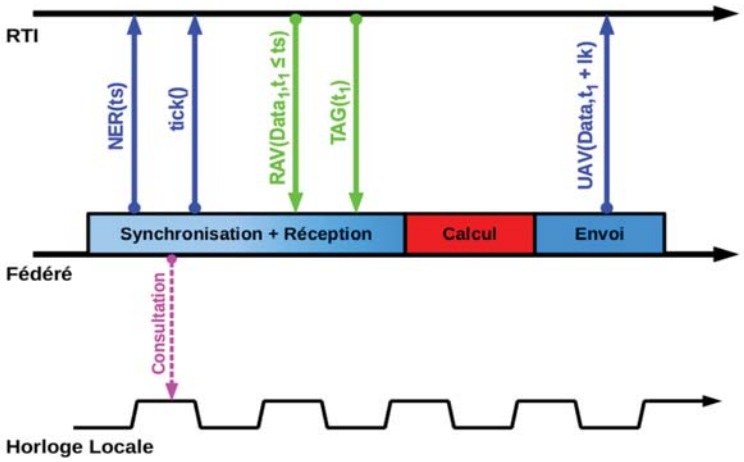


FIG. 4.13 – Modèle de synchronisation Event Driven (gestion du temps)

façon que précédemment, l'utilisation des mécanismes de gestion du temps complexifie l'étude du comportement des fédérés. De plus, si l'on utilise l'algorithme CMB (utilisé dans le CERTI), les fédérés *Event-Driven* sont sensibles à un phénomène appelé le *time creep* qui peut limiter les performances de la fédération. En effet, dans ce cas, la performance de l'algorithme CMB dépend du choix du paramètre *lookahead* pour chaque fédéré. Avec des *lookaheads* mal choisis, la fédération globale est contrainte par un échange excessif de messages *NULL* avant l'autorisation d'une avancée dans le temps. La figure 4.14 illustre ce problème en considérant deux fédérés FED1 et FED2. Au début de la simulation, leurs temps locaux (noté lt) sont égaux à 0 et leurs *lookaheads* (notés lk) sont égaux à 1. Ils n'échangent aucun message de simulation et ils souhaitent, tous les deux, avancer jusqu'au temps de simulation 5. Pour cela, ils appellent le service `nextEventRequest(5)`. Cette demande d'avancée dans le temps simulé nécessite l'échange de 12 messages *NULL* avant d'être acquittée par la RTI (callback `timeAdvanceGrant(5)`).

Ce mode d'exécution n'a pas été mis en œuvre dans nos travaux. Généralement, il est plus communément utilisé dans les simulations de théâtre de guerre ou de réalité virtuelle. Le modèle *Time-stepped* permet une concordance plus évidente entre la périodicité dans le temps réel et son expression dans le temps simulé. Toutefois, nous avons proposé un nouvelle version optimisée de l'algorithme CMB qui résout le problème du *time creep*, ce nouvel algorithme est basé sur l'architecture particulière du CERTI (*C.f.* partie 6.4, page 138). Notamment, nous verrons dans le chapitre 5 que les performances de ce nouvel algorithme sont intéressantes pour le simulateur de

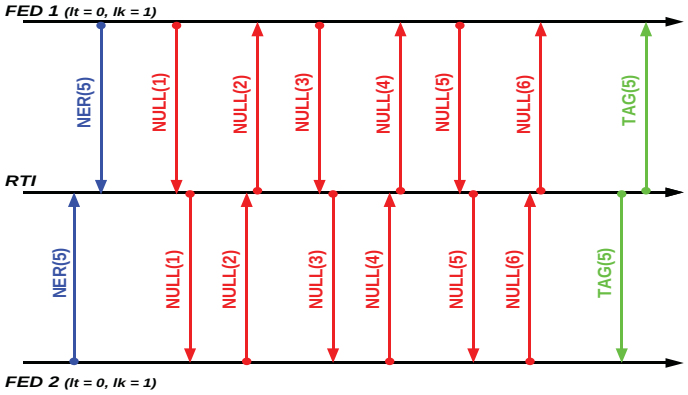


FIG. 4.14 – Problème du time creep avec deux fédérés

vol fonctionnant à partir du mode d'exécution mixte qui est détaillé dans la suite (partie 4.3.6). Ce mode mixte combine l'utilisation des services `nextEventRequest()` et `timeAdvanceRequest()`.

4.3.6 Le modèle mixte

Le comportement fonctionnel de ce type de mode d'exécution dans notre simulateur est plus compliqué. La phase d'exécution du fédéré est composée cette fois ci de deux phases de réception : la première est faite en appelant le service `nextEventRequest(`ts`)` utilisé par les fédérés *Event-Driven* et la deuxième phase est réalisée par l'appel au service `timeAdvanceRequest(`ts`)` utilisé par les fédérés *Time-Stepped*. L'utilité de ces deux phases de réception est due à la particularité de la résolution explicite d'équations différentielles d'état et de sortie des systèmes que nous simulons dans le simulateur de vol. Cette particularité est illustrée dans la figure 4.15.

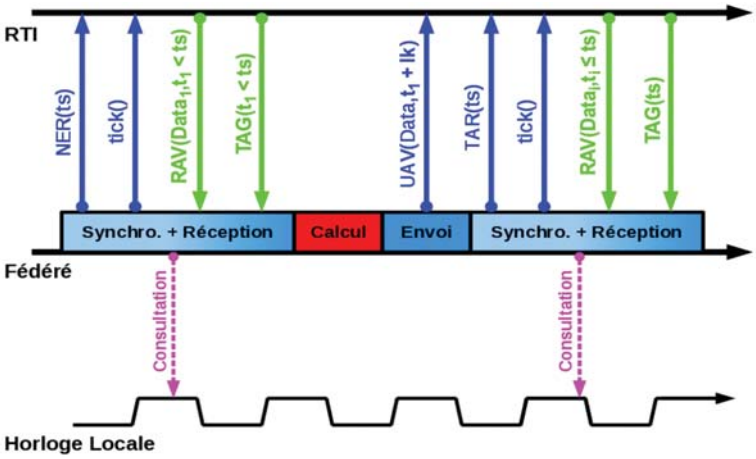


FIG. 4.15 – Modèle de synchronisation Mixte (gestion du temps)

De façon générale, durant ses cycles de calcul, le fédéré doit calculer deux choses :

1. son état suivant x_{n+1} à partir de son état courant x_n et de son entrée u_n ;
2. la sortie correspondante y_{n+1} qui dépend du nouvel état x_{n+1} et qui peut aussi dépendre de la nouvelles entrée u_{n+1}.

Certains fédérés ont donc besoin de la nouvelle entrée u_{n+1} pour calculer la sortie y_{n+1} lors du même pas de calcul. Ainsi si l'on souhaite conserver une concordance entre la période réelle du fédéré (en millisecondes) et son expression dans le temps simulé, chaque cycle de calcul du fédéré doit être composé de deux phases de réception. Cela permet au fédéré de recevoir les données relatives à ses entrées et publier ses sorties pour le même pas de temps. Ce mode de fonctionnement a été mis en avant par l'implémentation de méthodes de résolution des équations différentielles. Nous l'avons implémenté sur notre simulateur de vol et testé toujours sur notre plate-forme [CSSA11, CSSA12]. Les résultats expérimentaux sont décrits dans le chapitre suivant (*C.f.* chapitre 5, page 89). Une description détaillée des spécificités de la résolution de des équations différentielles est aussi disponible dans le rapport technique du simulateur de vol implémenté dans le cadre de ce travail de thèse [SCS11].

4.3.7 Discussion

Nous avons proposé différents modes d'exécution qui se distinguent par les techniques de synchronisation utilisées pour garantir la cohérence temporelle de l'application. Cette liste de modes d'exécution n'est pas exhaustive et certains fédérés peuvent mettre en œuvre des modes d'exécutions plus simples et d'autres des modes plus complexes. Par exemple, certains fédérés, comme le fédéré Joystick de notre simulateur de vol, produisent uniquement des données mais n'en reçoivent pas (dans ce cas ils sont régulateurs si on utilise la gestion du temps) donc, par conséquent, ils n'ont pas de phase de réception. Inversement, d'autres fédérés peuvent uniquement recevoir de l'information comme les deux fédérés de visualisation du simulateur de vol (dans ce cas ils sont contraints si on utilise la gestion du temps) donc, par conséquent, ils n'ont pas de phase d'envoi. Des fédérés plus complexes peuvent aussi être envisagés et peuvent être construits à partir des modes d'exécution présentés ici. Ainsi, pour conclure, ces différents modes d'exécution, utilisés pour implémenter nos cas tests, fournissent la réponse pour le niveau d'action de l'application comme cela est illustré sur la figure 4.16.

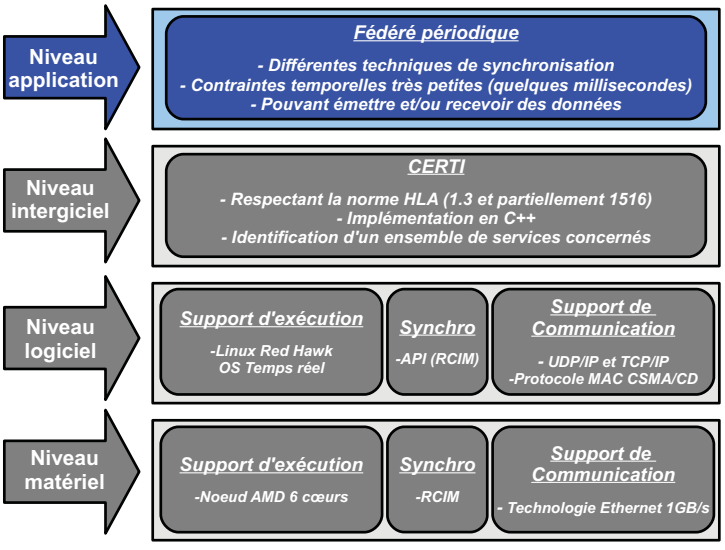


Fig. 4.16 – Réponse du niveau application

Chapitre 5

Mises en œuvre et expérimentations

Préambule

Dans le chapitre précédent, nous avons introduit nos cas tests décrivant des simulations de systèmes embarqués ainsi que différents modes d'exécution pour des fédérés temps réel (les simulateurs). Ici, nous allons présenter les différents résultats relatifs aux expérimentations réalisées lors des exécutions de ces cas tests sur notre plate-forme dédiée. Pour cela, nous exposerons dans un premier temps, les différents travaux que nous avons menés sur des modifications et des nouveaux services intégrés au CERTI. Ces fonctionnalités supplémentaires sont utiles pour réaliser et exécuter des simulations temps réel avec des cycles de calcul très petits. Ensuite, nous présenterons les différentes mesures de bases réalisées sur la plate-forme PRISE afin d'évaluer les temps de calcul d'un programme ainsi que les temps de transfert des messages par le CERTI. Enfin, nous terminerons par les différentes mesures réalisées lors des exécutions des cas tests. La simulation de vol en formation de satellites sera présentée dans un premier temps puis, enfin, nous terminerons par les résultats relatifs au simulateur de vol d'aéronef.

5.1 Vers des fonctionnalités supplémentaires pour le CERTI

5.1.1 Allocation d'un processus sur un processeur particulier

La majorité des systèmes informatiques actuels sont équipés de processeurs ayant plusieurs cœurs, c'est-à-dire plusieurs unités de traitements (pouvant réaliser des opérations logiques). Par exemple, la programmation muti-threads (plusieurs processus légers) divise le programme en plusieurs entités de traitement (les threads[1]) qui peuvent être exécutés parallèlement par les différentes unités de traitement. Dans notre cas, notre architecture est basée sur différents processus (des processus lourds) et notre objectif est donc d'assurer qu'un processus donné de notre application (un fédéré, un RTIA ou le RTIG) s'exécutera sur des cœurs spécifiques (un ou plusieurs). En ce sens, nous avons implémenté dans le CERTI des fonctionnalités qui lui permettent de gérer l'affinité processeur d'un processus. Le terme affinité processeur est une propriété de l'ordonnanceur pour assigner un processus (ou un thread) à un ensemble donné de processeurs sur le système. L'ordonnanceur honorera cette affinité en assurant que ce processus (ou ce thread) ne sera pas exécuté sur n'importe quel autre processeur. La figure 5.1 montre une illustration de ce mécanisme sur un nœud avec deux cœurs `CPU0` et `CPU1`. Ce mécanisme

[1] *La différence entre les threads et les processus a été décrite dans la partie 3.1.2, page 41*

nous permet d'affecter l'ensemble des processus du CERTI sur le processeur `CPU0` et d'assurer qu'ils ne s'exécuteront jamais sur le cœur `CPU1`. Le cœur `CPU1` peut ainsi être dédié pour d'autres tâches ou encore rester disponible pour gérer les activités du système d'exploitation (gestion des interruptions, ...).

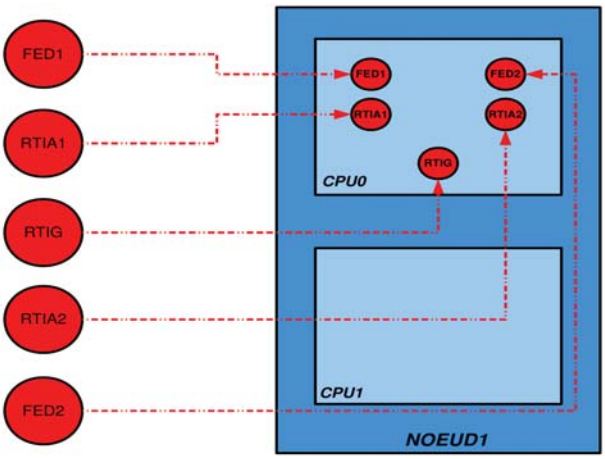


FIG. 5.1 – Affectation d'affinité processeur pour `CPU0`

Ces fonctionnalités sont actuellement disponibles pour le système d'exploitation Linux et nous travaillons sur une version pour le système Windows. Elles se présentent comme une extension non standard de l'API de HLA sous forme d'accesseurs en lecture et en écriture afin d'affecter ou de récupérer l'affinité processeur pour chaque processus du CERTI :

1. `getAffinityForFederate(...)` / `setAffinityForFederate(...)`;
2. `getAffinityForRTIA(...)` / `setAffinityForRTIA(...)`;
3. `getAffinityForRTIG(...)` / `setAffinityForRTIG(...)`.

La gestion de l'affinité du RTIG est assez délicate à mettre en œuvre car elle nécessite l'envoi d'un message réseau et la prise en compte de ce message par le processus central RTIG. Le travail d'implémentation sur ce point est encore en cours. Lors des expérimentations, cette fonctionnalité est implémentée directement dans le code source du processus RTIG. Les autres fonctionnalités pour le RTIA et le fédéré sont intégrées et utilisables dans la version actuelle du CERTI. L'outil Red Hawk NightStar [Con03] permet aussi de gérer l'affinité des processus à l'aide d'une interface graphique qui rend cette manipulation plus aisée pour l'utilisateur novice.

5.1.2 Affectation d'une priorité pour les processus

Le déterminisme temporel dans l'exécution d'un système informatique est, en partie, assuré par l'affectation de priorités pour les tâches selon des politiques définies par la théorie de l'ordonnancement (cette théorie est présentée dans l'annexe C, page 173 et le descriptif des techniques de l'ordonnancement appliquées à nos simulations est disponible dans le chapitre 6, page 113). Nous avons donc ajouté une autre interface dans le CERTI afin d'autoriser la gestion de la priorité et la configuration de l'ordonnanceur pour les processus. Les systèmes d'exploitation ne

sont pas tous égaux de la quantité de niveau de priorité proposée. Le système d'exploitation VxWorks propose 256 niveaux de priorité [Win08], les systèmes Linux proposent, quant à eux, 100 niveaux de priorité [FK03], le système d'exploitation QnX propose 64 niveaux de priorité [Hil92] et enfin Windows ne propose, en général, que 32 niveaux de priorité [83]. Les appels systèmes `sched_get_priority_min()` et `sched_get_priority_max()` permettent de déterminer l'intervalle des priorités valides de manière portable sur les systèmes conformes à la norme POSIX (VxWorks, Linux, Qnx, ...).

De plus, cette modification de la priorité s'accompagne du choix des algorithmes d'ordonnancement temps-réel. Linux, comme la plupart des systèmes d'exploitation temps réel conformes à la norme POSIX, propose deux algorithmes d'ordonnancement temps réel : `SCHED_FIFO` et `SCHED_RR`. Ces algorithmes sont destinés aux applications temps réel critiques qui nécessitent un contrôle précis sur la manière dont les processus exécutables sont exécutés. Ces algorithmes sont préemptifs c'est-à-dire que si un processus (ou un thread) avec une priorité statique plus élevée devient prêt, le processus (ou le thread) en cours est interrompu et retourne dans sa liste d'attente. `SCHED_FIFO` est un ordonnancement utilisé dans la plupart des techniques classiques d'allocation de priorité statiques (Rate Monotonic [LL73], Deadline Monotonic [LW82], ...). Un processus `SCHED_FIFO` qui a été préempté par un autre processus de priorité supérieure restera en tête de sa liste et reprendra son exécution dès que tous les processus de priorités supérieures seront de nouveau bloqués. `SCHED_RR` est une variante de la politique `SCHED_FIFO` où chaque processus ne dispose que d'un intervalle de temps limité pour son exécution. Si un processus sous politique `SCHED_RR` s'est déjà exécuté avec une durée supérieure ou égale à sa tranche temporelle, il sera placé à la fin de la liste d'attente correspondante à sa priorité. Cette politique n'est pas compatible avec les hypothèses des techniques classiques d'allocation de priorité et de validation de la théorie de l'ordonnancement. Cependant, certaines études, telles que la thèse de Nicolas Navet [Nav99], ont montré l'utilité de `SCHED_RR` pour certaines configurations de tâches qui n'étaient pas ordonnançables avec `SCHED_FIFO`. La figure 5.2 montre une illustration de ces mécanismes sur le cœur `CPU0` d'un noeud Linux. Sur cet exemple, nous pouvons configurer la politique d'ordonnancement (`SCHED_FIFO` est la plus utilisée) et nous considérons que le processus du RTIG a la plus forte priorité, ensuite les processus FED1 et son RTIA puis enfin FED2 et son RTIA.

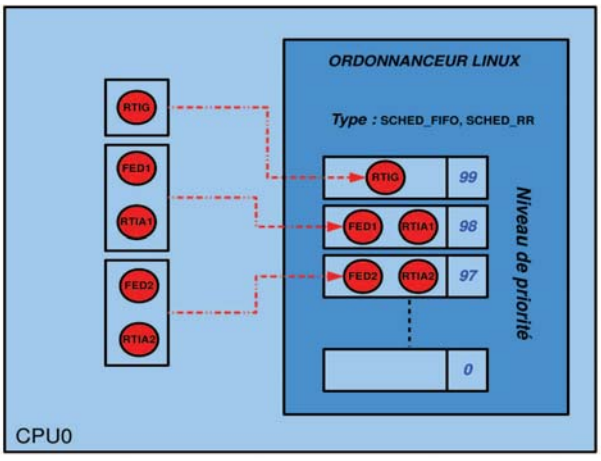


FIG. 5.2 – Affectation de la priorité et de la politique pour l'ordonnanceur Linux du `CPU0`

Comme les fonctionnalités précédentes, celles-ci sont actuellement disponibles pour le système d'exploitation Linux et en cours de développement pour le système Windows. Cela constitue une extension non standard de l'API HLA sous forme d'accesseurs en lecture et en écriture pour affecter et récupérer les paramètres de l'ordonnanceur pour chaque processus du CERTI :

1. `getSchedulingForFederate(...)` / `setSchedulingForFederate(...)` ;
2. `getSchedulingForRTIA(...)` / `setSchedulingForRTIA(...)` ;
3. `getSchedulingForRTIG(...)` / `setSchedulingForRTIG(...)`.

De plus, de la même façon que pour la gestion de l'affinité, la configuration de l'ordonnanceur pour le processus RTIG est aussi délicate à mettre en œuvre (envoi d'un message réseau spécifique, ...). Le travail d'implémentation sur ce point est aussi en cours et cette fonctionnalité est aussi implémentée directement dans le code source du processus RTIG pour les tests. L'outil Red Hawk NightStar [Con03] permet aussi de configurer l'ordonnanceur pour chaque processus à l'aide d'une interface graphique ; néanmoins, notre API propose un avantage non négligeable. En effet, dans le cas d'une configuration par l'outil NightStar, la priorité du processus et l'ordonnancement choisi sont affectés initialement et ne peuvent pas être modifiés pendant l'exécution du système. En revanche, avec nos appels systèmes, nous pouvons programmer le fédéré de telle sorte qu'il puisse effectuer des changements de priorités pendant son exécution. Cela est très utile lorsque l'on souhaite appliquer les techniques de discrétisation qui seront présentées ensuite (*C.f.* partie 6.2.3, page 120).

5.1.3 Les fonctionnalités de gestion de la mémoire

La gestion de la mémoire est un point clé dans la programmation d'un système temps réel. En effet, le déterminisme temporel du programme dépendra de la prise en compte des principes de fonctionnement de la mémoire d'un processus informatique. Cette mémoire est regroupée en quatre parties distinctes (*C.f.* Figure 5.3) :

1. La partie **Code** qui contient l'ensemble des instructions du programme ;
2. La partie **Données** est une mémoire globale et accessible depuis n'importe quel endroit du programme (n'importe quel jeu d'instructions), où sont stockées les données statiques globales ;
3. La partie **Tas** est une mémoire globale de taille variable accessible depuis n'importe quel endroit du programme où sont stockées les données dynamiquement allouées ;
4. La partie **Pile** est une mémoire locale aussi appelée la pile, accessible localement seulement, où sont passés les paramètres des fonctions et stockées les données locales et temporaires.

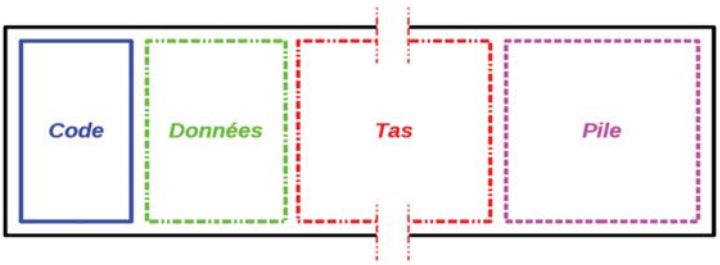


FIG. 5.3 – Les différentes zones de la mémoire d'un processus

Le concepteur du système temps réel devra alors utiliser certains mécanismes pour gérer cette mémoire et s'affranchir des différents problèmes que cela peut poser pour un système temps réel. Dans notre cas, pour la programmation du CERTI et des fédérés, nous nous sommes intéressés à deux points essentiels :

1. la gestion de la pagination des processus est détaillée dans la partie 5.1.3.1 ;
2. les mécanismes d'allocation dynamique de mémoire virtuelle pour le processus sont détaillés dans la partie 5.1.3.2.

5.1.3.1 Gestion de la pagination des processus

La mémoire d'un ordinateur est organisée autour du concept de la mémoire virtuelle dont le rôle est de permettre l'exécution de plusieurs processus qui nécessiteraient normalement un espace mémoire total supérieur à l'espace mémoire primaire (mémoire vive RAM et mémoire cache du processeur). Pour cela, la mémoire virtuelle s'appuie sur une mémoire secondaire (typiquement les disques durs) pour stocker les programmes et les données en entier. La mémoire physique de l'ordinateur est découpée en blocs de taille fixe appelés cases. Par conséquent, la mémoire virtuelle d'un processus informatique est découpée en blocs de même taille fixe appelés des pages. Chaque page peut alors être affectée dans une case quelconque (ou pas du tout tant qu'elle n'est pas utilisée par le programme). Une table de correspondance, appelée table des pages, est gérée par le système d'exploitation pour assurer la correspondance entre les pages mémoires du processus et la mémoire physique de l'ordinateur (*C.f.* Figure 5.4). Le mécanisme de pagination du matériel assure la traduction des adresses de mémoire virtuelle en une adresse de mémoire physique, de façon à retrouver la contiguïté de cette mémoire virtuelle.

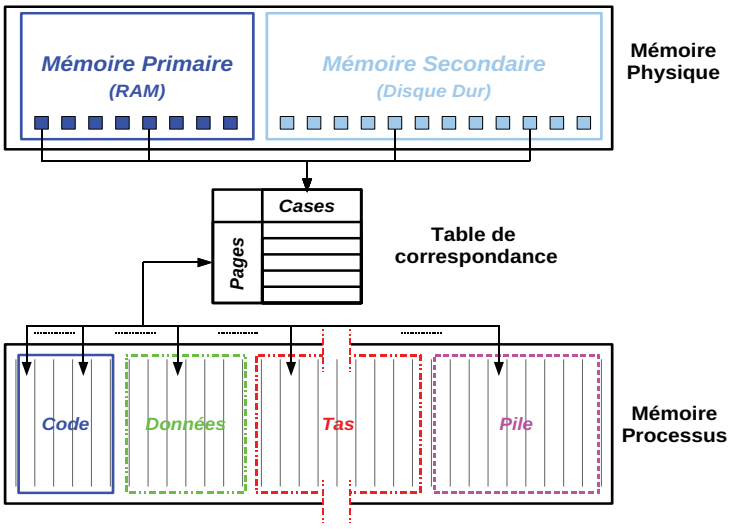


FIG. 5.4 – Correspondance entre la mémoire virtuelle d'un processus et la mémoire physique

La mémoire virtuelle nécessaire à l'exécution de plusieurs processus sur une même machine peut être de taille importante et ainsi dépasser la taille de la mémoire physique primaire disponible (RAM). Pour résoudre cette difficulté, le système d'exploitation ne conserve dans la RAM que les pages de mémoire virtuelle dont les processus ont besoin pour leurs exécutions courantes, les autres ne sont tout simplement pas créées par le système d'exploitation tant qu'elles

ne sont pas utilisées. D'autres fois, elles sont stockées et conservées sur la mémoire secondaire (disque dur). Par exemple, le système Linux Red Hat recommande l'utilisation d'une partie du disque dur explicitement dédiée à ce rôle, appelée l'espace de "*swap*" [57]. Cependant, lorsqu'un processus souhaite accéder à une page qui n'est pas affectée à une case mémoire de la RAM, l'instruction en cours du processus est interrompue : on dit qu'il y a défaut de page. Le système doit alors gérer une allocation d'une case mémoire de la RAM pour cette page, créer la page ou l'exporter depuis la mémoire secondaire et relancer l'exécution de l'instruction précédemment interrompue qui pourra alors se dérouler jusqu'à son terme. Lorsque un problème de défaut de page se produit, s'il y a une case libre, le système peut allouer directement cette case. Cependant, si aucune case n'est libre, il est nécessaire de réquisitionner une case occupée par une autre page : on dit qu'il y a remplacement de page. Plusieurs algorithmes de remplacement de pages ont été proposés [Bel66, SG83] et diffèrent par le choix de la page remplacée (LFU[2], LRU[3], ...). Cependant, ces techniques de remplacement de pages ajoutent de la latence dans l'exécution du programme et peuvent être une source d'indéterminisme temporel (on ne peut pas borner le temps d'exécution du programme). Un temps précieux des ressources de calcul peut donc être utilisé pour récupérer les pages nécessaires à l'exécution d'un ou plusieurs processus. Ainsi, la latence et l'indéterminisme dus à la pagination sont souvent inacceptables pour des processus temps réel critiques.

Sous réserve d'avoir une quantité de mémoire RAM suffisante pour satisfaire les besoins de tous les processus temps réel présents sur le système, il est possible d'empêcher ces mécanismes de pagination. Pour cela, un processus temps réel critique peut se verrouiller en mémoire RAM afin de garantir une exécution sans latence de pagination. Dans notre cas, sous Linux, nous utilisons aussi le mécanisme de `mlockall()` qui permet de désactiver la pagination pour toutes les pages présentes dans l'espace d'adressage du processus appelant. L'option `MCL_CURRENT` permet de verrouiller toutes les pages présentes au moment de l'appel dans l'espace d'adressage du processus et l'option `MCL_FUTURE` permet de verrouiller toutes les pages qui seront dans l'espace d'adressage du processus dans le futur (par exemple une extension du tas suite à une allocation mémoire). Ainsi, lorsque l'appel à cet opérateur réussit, on a l'assurance que toutes les pages concernées du processus résideront en mémoire vive (RAM) et y resteront jusqu'à un déverrouillage par l'opérateur inverse `munlockall()` ou encore jusqu'à ce que le processus en question termine son exécution. Dans notre cas, nous voulons pouvoir appliquer ce mécanisme pour tous les processus mis en œuvre dans le CERTI (fédérés, RTIAs et RTIG) et cela constitue une extension non standard de l'API de HLA :

1. `lockPageTableForFederate(...)` / `unlockPageTableForFederate(...)` ;
2. `lockPageTableForRTIA(...)` / `unlockPageTableForRTIA(...)` ;
3. `lockPageTableForRTIG(...)` / `unlockPageTableForRTIG(...)` .

Ces fonctionnalités sont donc disponibles pour le système d'exploitation Linux et nous travaillons sur une exportation compatible avec le système Windows à l'aide des opérateur `VirtualLock()` et `VirtualUnlock()` [84]. La configuration mémoire du processus RTIG est aussi implémentée directement dans le code source du processus RTIG.

5.1.3.2 Gestion de l'allocation dynamique

De façon générale, les variables utilisées (et donc à créer) ainsi que leurs tailles sont connues au moment de la compilation pour dimensionner la mémoire. Cependant, dans certains cas, on

[2] *Least Frequently Used*
[3] *Least Recently Used*

ne peut pas connaître à l'avance la taille de certaines données et le compilateur ne peut donc pas faire la réservation de l'espace mémoire nécessaire. Cette réservation de mémoire (appelée encore allocation dynamique) doit être faite pendant l'exécution du programme et prévue par le programmeur. Le concepteur déclare simplement un pointeur désignant un espace mémoire du tas qui sera réservé ensuite pendant l'exécution du processus et cet espace n'a donc pas de taille fixe (*C.f.* figure 5.5).

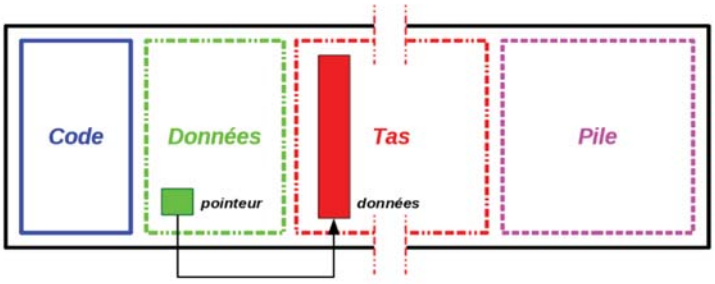


FIG. 5.5 – Principe de l'allocation dynamique

L'exigence de déterminisme doit s'appliquer également à toutes les opérations de gestion dynamique de la mémoire (allocation et libération) qui n'étaient pas adaptées dans leur version originale. Ainsi, la gestion dynamique de la mémoire a été, jusqu'à une date récente, soigneusement évitée dans la conception des logiciels temps réel. Cependant, l'accroissement constant de la complexité des fonctionnalités que l'on souhaite intégrer dans les applications temps réel a poussé les concepteurs à gérer leurs ressources mémoire de manière dynamique. Dans le cas du CERTI, en faisant une étude préliminaire précise sur le dimensionnement mémoire de notre application, nous aurions pu envisager de remplacer l'allocation dynamique de mémoire par des mécanismes statiques. Cependant, l'utilisation de la mémoire dynamique offre une souplesse que nous souhaitons conserver pour le CERTI qui est un outil multi-plateforme ; nous nous sommes donc intéressés aux méthodes déterministes pour l'allocation dynamique de mémoire.

D'un point de vue fonctionnel, ce sont les opérateurs d'allocation qui effectuent explicitement cette réservation (`malloc()` en langage `C` ou `new()` en langage `C++`). Si une zone de mémoire du tas de la taille nécessaire est disponible, l'opérateur d'allocation va la réserver et renvoyer l'adresse de cette zone (qui est stockée dans l'espace mémoire des données). L'opérateur garantit aussi que cette zone du tas ne sera pas utilisée à d'autres fins tant qu'il n'aura pas été explicitement libéré par un désallocateur (`free()` ou `delete()`). Le programmeur doit donc s'assurer d'une gestion correcte de l'espace mémoire du tas lors de la conception de son programme. Cependant, la recherche d'un espace mémoire "valide" est laissée à la charge de l'algorithme mis en œuvre dans l'opérateur d'allocation. Les algorithmes d'allocation dynamique de mémoire classiques utilisent des approches *First-Fit* ou *Best-Fit* fondées sur le maintien d'une liste des blocs libres (de taille fixée). Dans l'approche *First-Fit*, l'allocateur sélectionne le premier bloc libre de taille suffisante pour satisfaire la requête. Si l'espace restant est d'une taille suffisante, un nouveau bloc libre est inséré dans la liste des blocs libres. Dans l'approche *Best-Fit*, l'allocateur sélectionne le bloc libre dont la taille est la plus proche (et suffisante) de celle du bloc requis. L'inconvénient de cette approche est une fragmentation due à l'existence de nombreux blocs de petite taille [HC05]. Ces approches classiques ne permettent pas d'assurer des temps d'exécution déterministes, nous avons donc travaillé sur l'intégration au CERTI de mécanismes d'allocation déterministes avec les allocateurs de la librairie TLSF[4] [74] (disponible sous licence Open-Source). Les allocateurs TLSF sont dédiés aux systèmes embarqués. Pour le moment, nous avons surchargé l'opérateur

[4] *Two Level Segregate Fit*

`new()` du `C++` pour pouvoir l'utiliser simplement et l'intégrer ensuite dans le code source du CERTI. Les premières expérimentations pour la compilation et l'exécution d'une simulation en intégrant ces nouveaux allocateurs dans le code des processus des fédérés ont été concluantes. Actuellement, nous poursuivons nos travaux pour intégrer ces mécanismes à tous les processus du CERTI et ainsi pouvoir exécuter une simulation entièrement sûre du point de vue de la gestion de la mémoire. Ce nouvel allocateur `new()` n'a pas encore été intégré dans la librairie du CERTI car il nécessite de plus amples expérimentations. L'utilisation de cette allocateur doit être rigoureuse pour prévoir un dimensionnement de la mémoire en accord avec les besoins de l'application. Le lecteur intéressé pourra se référer à la partie annexe A.3, page 157, dans laquelle les différentes techniques d'allocation mémoire sont décrites et analysées.

5.1.4 Les services `tick()`

Le service `tick()` est un service particulier de la norme HLA. En utilisant ce service, le fédéré peut rendre la main à la RTI afin de lui permettre d'invoquer la ou les callback(s) attendue(s). A l'origine, ce service n'était pas présent dans les spécifications d'interface de la norme DMSO 1.3 mais uniquement dans les spécifications de l'interface du RTI-NG [DoD00]. Par la suite, ce service fut intégré dans la norme IEEE 1516 sous le nom `EvokeCallback()`. Dans les deux cas, il peut être utilisé selon deux sémantiques différentes. La première version est le `tick()` sans argument (ou `EvokeCallback()` dans la norme IEEE 1516) qui est utilisé lorsque le fédéré attend une callback particulière. Le fédéré utilise ce service périodiquement avec un intervalle de 1 milliseconde jusqu'à ce que la callback attendue lui soit délivrée. Ce mode d'exécution est illustré dans la figure 5.6 dans lequel un fédéré qui se trouve dans une boucle d'avance dans le temps doit "ticker" pour recevoir la callback `reflectAttributeValues()` (*i.e.* `RAV()`) de la donnée 1, celle de la donnée 2 puis encore une fois pour recevoir l'acquittement de son avancée dans le temps simulé par la callback `timeAdvanceGrant()` (*i.e.* `TAG()`).

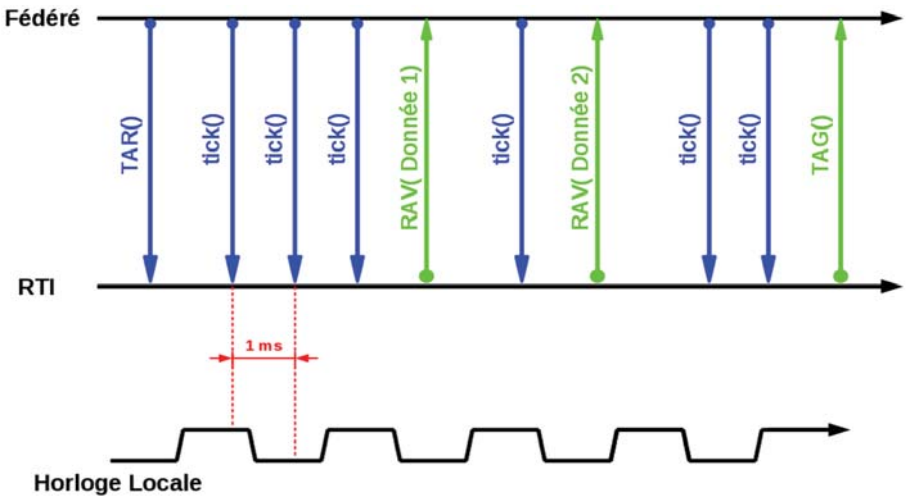


FIG. 5.6 – Fonctionnement du service `tick()` sans argument

La seconde version est le `tick(Min,Max)` (ou `EvokeMultipleCallbacks(Min,Max)` dans la norme IEEE 1516) qui possède deux arguments temporels qui indiquent les intervalles (minimal et maximal) de temps durant lequel la RTI pourra garder la main afin de fournir aux fédérés toutes les callbacks disponibles. Ce mode d'exécution est illustré dans la figure 5.7 dans lequel un fédéré qui se trouve dans une boucle d'avance dans le temps n'appelle qu'une seule fois le service

dans sa boucle d'avance dans le temps pour recevoir la callback de la donnée 1, de la donnée 2 puis l'acquittement de son avancée dans le temps simulé par la callback `timeAdvanceGrant()`. Il faut cependant que ces callbacks soient disponibles dans l'intervalle de temps imparti.

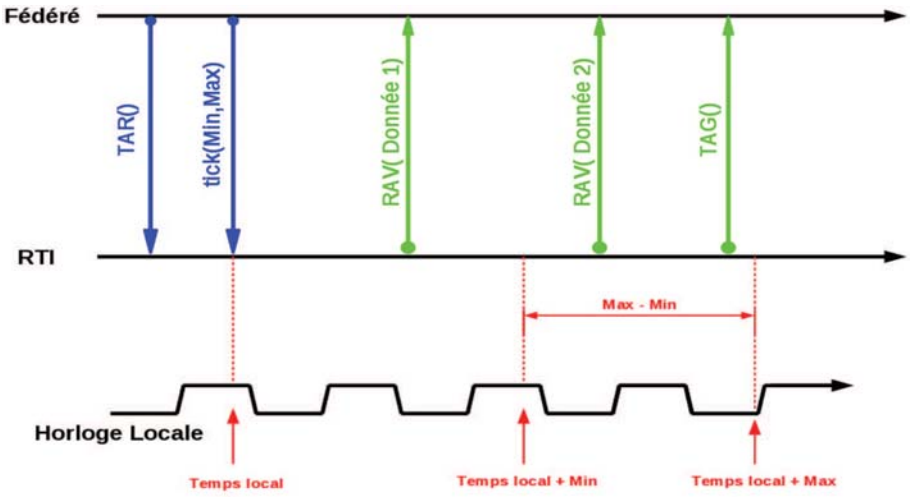


FIG. 5.7 – Fonctionnement du service `tick(Min,Max)`

Ces deux versions peuvent poser des problèmes pour les simulations temps réel. En effet, dans le premier cas du `tick()` sans argument, le fédéré est sûr que la RTI lui rendra la main uniquement lorsqu'elle aura la callback attendue. En revanche, il n'a aucun moyen de savoir à quel moment cette callback arrivera et cela peut amener à des fautes temporelles, c'est-à-dire que le fédéré peut recevoir des callbacks en dehors de son intervalle temporel d'exécution. De plus, les appels successifs (selon une période de 1 ms) à ce service surcharge le système car cela engendre des commutations de contexte entre les processus du fédéré et du RTIA ainsi que des communications sur les sockets Unix. Ces problèmes sont illustrés sur la figure 5.8.

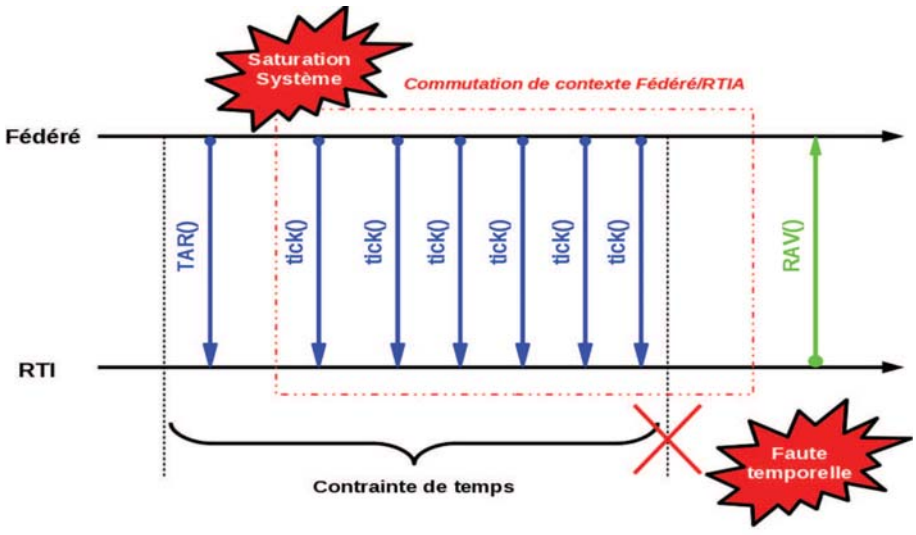


FIG. 5.8 – Illustration des problèmes du service `tick()` sans argument

Dans le second cas du `tick(Min,Max)`, nous avons le déterminisme temporel car nous savons que la RTI rendra la main à son fédéré (et donc à l'application) au pire cas à l'instant `t=Max`. En revanche, on ne peut pas être sûr que le fédéré recevra toutes les callbacks attendues avant l'expiration du délai `Tmax`. Cela amène à des situations de fautes fonctionnelles de l'application et pose des problèmes de fiabilité et de tolérance aux fautes car un fédéré pourrait reprendre son exécution sans avoir reçu ses nouvelles données d'entrée (*C.f.* figure 5.9).

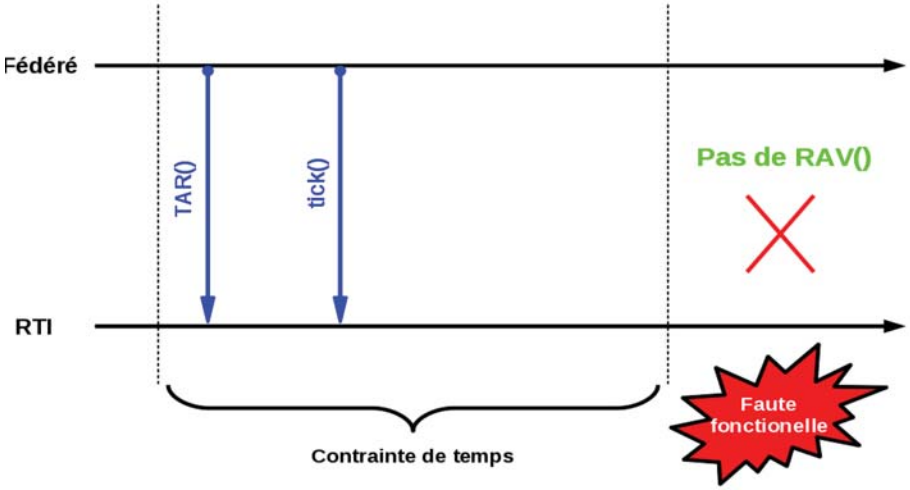


FIG. 5.9 – Illustration du problème du service `tick(Min,Max)`

Nous souhaitons assurer le déterminisme fonctionnel de notre application car nous voulons que chaque fédéré de nos applications soit sûr de recevoir toutes les données attendues avant de réaliser ses calculs locaux. Nous avons donc conservé le principe de fonctionnement du `tick()` sans argument pour implémenter un nouveau service nommé `tick2()`. Ce service ne permet pas de s'affranchir de l'indéterminisme temporel mais en revanche il permet de résoudre le problème de saturation du système. En effet, cette version `tick2()` est dite "bloquante" et ne rendra la main à l'application que lorsque la callback attendue sera disponible. Du coup, on évite les commutations de contexte excessives entre le fédéré et son RTIA. Une illustration de ce mode de fonctionnement est donnée sur la figure 5.10.

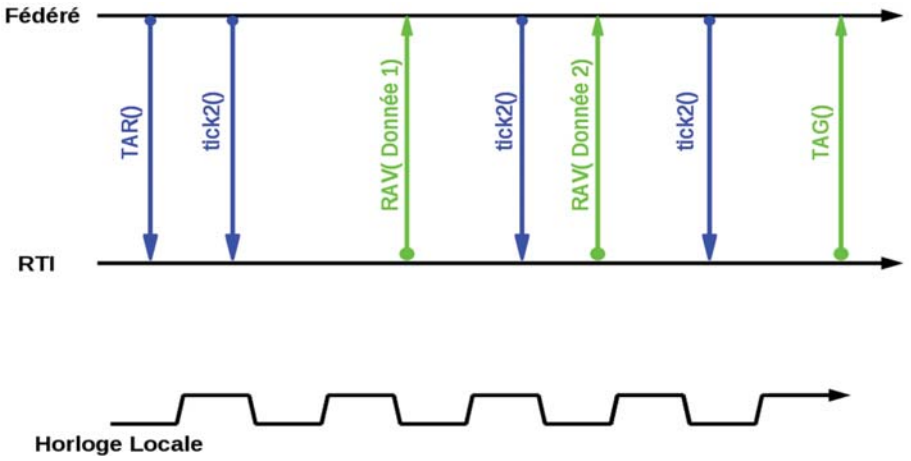


FIG. 5.10 – Fonctionnement du service `tick2()` ONERA

5.1.5 Vers une communication par mémoire partagée

Nous avons repris les travaux menés dans le cadre de l'étude sur HP-CERTI [ABR+04] afin de remplacer les communications par sockets par des communications par mémoire partagée. Le CERTI est concerné par deux types de communications : les communications Fédéré-RTIA et les communications RTIA-RTIG. Les communications entre le processus du fédéré et son RTIA se font généralement par l'utilisation d'une socket Unix sous les systèmes Linux. Nous pouvons remplacer cette communication par une socket TCP selon une option de compilation dédiée et, sous le système Windows, on utilisera dans tous les cas cette communication TCP. Les communications entre les différents RTIAs et le processus central du RTIG sont assurées respectivement par des sockets TCP et UDP selon les modes de transport choisis pour les attributs HLA (mode `RELIABLE` ou mode `BEST-EFFORT`). Le but de la communication par mémoire partagée est d'optimiser les performances des communications entre les processus situés sur un même nœud du système distribué. Dans les travaux sur HP-CERTI, ce sont les sockets de communication TCP et UDP entre le RTIG et les différents RTIAs qui avaient été remplacées par des segments de mémoire partagée composés de trois parties : *Flag*, *Header* et *Body*. La synchronisation pour la lecture et l'écriture sur ce segment entre les deux utilisateurs (RTIA et RTIG) est basée sur une gestion adéquate de l'indicateur d'état *Flag*. Ainsi lorsque un producteur écrit un message sur le segment, toutes éventuelles opérations d'écriture sont bloquées jusqu'à le message ait été lu par le consommateur. La partie *Header* contient les informations internes aux CERTI pour la transmission du message et la partie *Body* contient les données du message (mises à jour d'un attribut par exemple). Ce fonctionnement est illustré dans la figure 5.11.

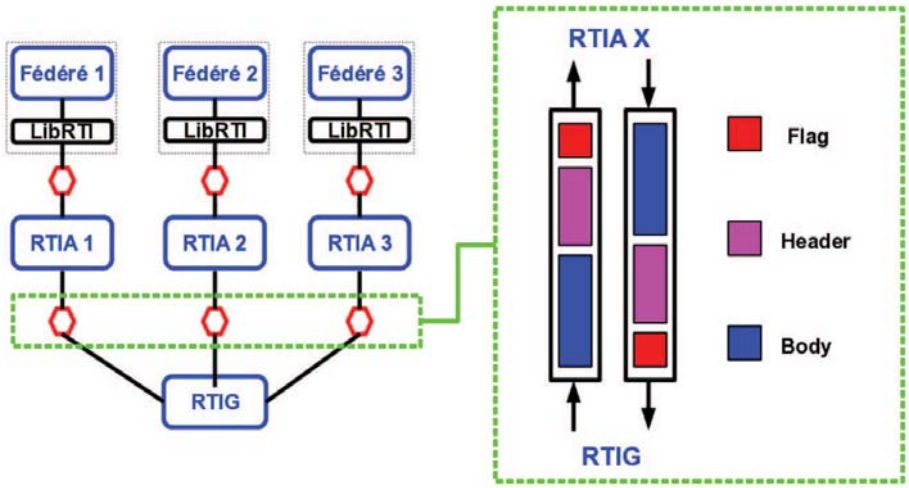


Fig. 5.11 – Communication mémoire partagée dans HP CERTI

Nous avons donc poursuivi les travaux concluants menés sur le projet HP-CERTI avec deux objectifs : (1) améliorer les techniques et les implémentations existantes et (2) pérenniser le travail en le rendant multi-plate-forme. Dans un premier temps, nous avons modifié le protocole de synchronisation entre le producteur et le consommateur du message en remplaçant l'utilisation du *flag* par des sémaphores (principe de Edsger W. Dijkstra [Dij68]) qui est la méthode la plus couramment utilisée pour restreindre l'accès à des ressources partagées. Ensuite, nous nous sommes intéressés à l'intégration des communications par mémoire partagée entre les fédérés et leurs RTIAs respectifs. Ces communications sont néanmoins, plus délicates à mettre en œuvre en utilisant la mémoire partagée que les communications entre le RTIG et les RTIAs. En effet, le mécanisme particulier de livraison des messages entre le fédéré et son RTIA impose l'utilisation

du service `tick()` (*C.f.* partie 5.1.4 précédente) et ainsi le RTIA peut stocker plusieurs messages avant de les délivrer explicitement au fédéré. Pour résoudre ce problème, nous avons utilisé le principe de programmation du *Circular Buffer* permettant de stocker plusieurs données avant leur consommation en dissociant les pointeurs de lecture et d'écriture (une présentation détaillée est disponible au chapitre 28 du livre de Steven W. Smith [Smi97]).

Ces nouveaux services sont regroupés dans un certain nombre de classes `C++` offrant des services de communication par des segments mémoire partagée entre 2 processus. Les appels classiques d'envoi `Send()` et de réception `Receive()` de messages des sockets ont été conservés. Les codes `C++` de ces classes sont disponibles et intégrés dans la librairie du CERTI (libCERTI) et un cas test pour l'utilisation des différentes classes est disponible dans les exécutables livrés avec le CERTI. Ces différentes classes sont disponibles sous les systèmes d'exploitation Linux selon les normes *System V* et *Posix* ainsi que sur le système d'exploitation Windows. Une fois intégré directement dans le CERTI, ces sockets pourraient remplacer n'importe quelle autre socket de communication classique comme cela est montré sur la figure 5.12. L'inclusion de ces nouvelles classes dans le code source du CERTI fait partie des prochaines évolutions envisagées.

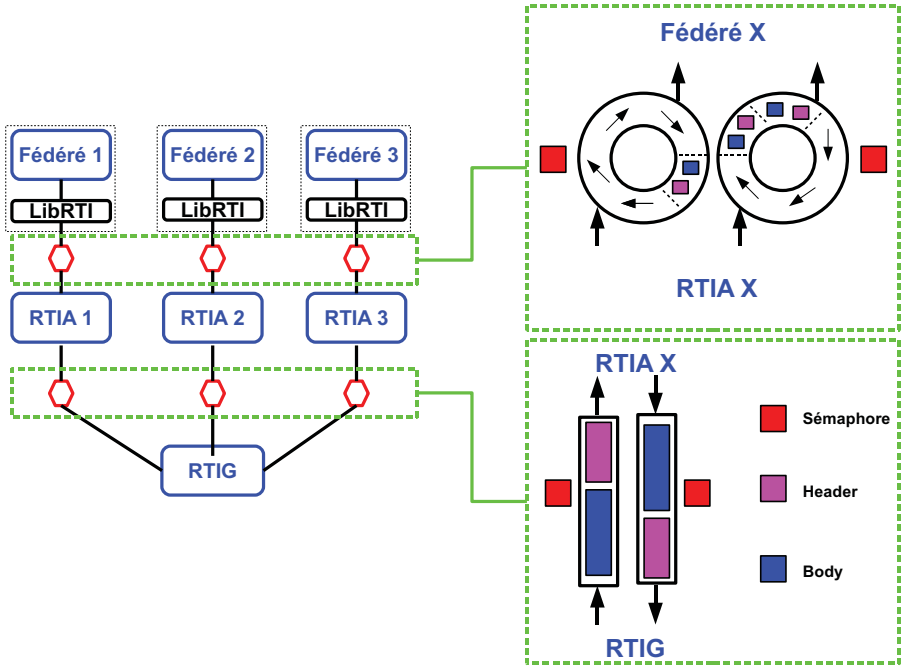


FIG. 5.12 – Communication mémoire partagée envisagée dans les prochaine évolution du CERTI

5.1.6 Discussion

Cette partie permet de présenter l'ensemble des points qui ont été étudiés pour inclure des nouveaux services et de nouvelles fonctionnalités à l'intergiciel CERTI ainsi que d'en améliorer les performances. Certaines parties de ces travaux sont encore en cours au moment de la rédaction de ce document. Cependant, le lecteur peut trouver ici un état des lieux des différents travaux envisagés et mis en œuvre dans le contexte de ce travail de thèse. Les fonctionnalités de gestion de l'affinité processeur et de configuration de l'ordonnanceur (et des priorités) sont indispensables pour assurer l'exécution d'une simulation distribuée en accord avec les techniques formelles présentées dans la suite du document (*C.f.* partie 6, page 113). Les autres fonctionnalités permettent un gain de performance et une meilleure utilisation des ressources pour assurer le déterminisme de la simulation ainsi que le respect de cycles très courts pour chaque fédéré.

5.2 Mesures de référence pour l'exécution et la communication

5.2.1 Mesures sur les temps de communication avec CERTI

Nous avons développé un benchmark *PING-PONG* qui peut être utilisé pour mesurer les latences de communication pour différentes tailles de messages transitant par le CERTI. Dans ce cas, deux fédérés PING et PONG s'échangent des messages, avec une taille spécifiée par l'utilisateur, aussi vite que possible. Le tableau 5.1 regroupe les différents temps nécessaires (en millisecondes) au transfert de messages avec des tailles différentes selon trois configurations :

- ***Configuration 1*** : Le fédéré PING, le fédéré PONG, leurs RTIAs respectifs ainsi que le RTIG sont situés sur un seul nœud Red Hawk de la plate-forme PRISE ;
- ***Configuration 2*** : Le fédéré PING, son RTIA et le RTIG sont situés sur un nœud Red Hawk de PRISE, le fédéré PONG et son RTIA fonctionnent sur un autre nœud ;
- ***Configuration 3*** : Le fédéré PING et son RTIA sont situés sur un nœud Red Hawk de PRISE, le fédéré PONG et son RTIA fonctionnent sur un autre nœud et finalement le RTIG fonctionne seul sur un troisième nœud.

Les mesures pour chaque configuration sont faites sans (*i.e.* N) ou avec les services en temps réel (*i.e.* RT) décrits dans le chapitre précédent pour assigner les priorités, les affinités ainsi que les verrouillages des pages mémoires pour chaque processus (fédérés, RTIAs et RTIG). Lorsque les mécanismes temps réel du CERTI sont utilisés, nous séparons physiquement les processus du CERTI (les RTIAs et le RTIG) et les processus des fédérés de la même manière que pour les configurations des deux cas test (*C.f.* partie 5.3.1, page 103 et partie 5.4.1, page 105). Par exemple, le fédéré PING s'exécutera sur un cœur (processeur) différent de son RTIA. Cette approche nous permet de distinguer explicitement les traitements dédiés aux communications (CERTI) et les traitements dédiés aux exécutions (algorithmes des fédérés[5]). Le mode d'exécution utilisant les services temps réels donnent des résultats légèrement meilleurs ; cela est dû à une meilleure allocation des ressources de calculs et aux priorités importantes des processus. Cela permet, par exemple, à chaque processus de traiter plus rapidement les interruptions matérielles (sockets de communication, ...).

Taille des messages	***Conf. 1*** *(ms)*		***Conf. 2*** *(ms)*		***Conf. 3*** *(ms)*	
	N	***RT***	***N***	***RT***	***N***	***RT***
100 bits	**0.293**	**0.224**	**0.252**	**0.214**	**0.236**	**0.203**
500 bits	**0.315**	**0.233**	**0.263**	**0.230**	**0.256**	**0.225**
1000 bits	**0.353**	**0.262**	**0.281**	**0.244**	**0.286**	**0.247**
5000 bits	**0.406**	**0.316**	**0.411**	**0.336**	**0.422**	**0.367**
10000 bits	**0.422**	**0.373**	**0.478**	**0.426**	**0.522**	**0.487**
50000 bits	**1.066**	**0.952**	**1.372**	**1.224**	**1.607**	**1.536**

TAB. 5.1 – Mesures des temps de communications du CERTI selon des messages d'une taille donnée

[5] *Notons que, dans ce cas test, les fédérés PING et PONG n'exécutent aucun algorithme et doivent simplement s'échanger des chaînes de bits.*

5.2.2 Mesures sur les temps d'exécution

Dans le contexte de systèmes temps-réel, il est nécessaire de pouvoir estimer le temps d'exécution pire-cas pour dimensionner correctement le système et trouver la meilleure répartition des tâches entre les processeurs. Dans le cas des simulations du CNES (*C.f.* partie 4.2.1, page 73), nous ne possédions pas les spécifications techniques des algorithmes mis en œuvre dans les différents fédérés mais nous connaissions les différents temps de calculs. Ainsi, dans notre cas, nous avons fait quelques mesures des temps de calculs sur un nœud de PRISE lors de l'exécution d'un algorithme avec une complexité temporelle donnée. Ici, nous supposons que la complexité spatiale (relative à la mémoire) est correctement dimensionnée en accord avec les principes et les contraintes des systèmes embarqués. Les mesures obtenues ont été regroupées dans le tableau 5.2. Dans ce cas, les services supplémentaires temps réel ne permettent pas de réduire les temps de calculs pour l'exécution linéaire d'un algorithme unique (un processus donc un seul fil d'exécution).

$O(n^m)$	$m=1$ *(ms)*	$m=2$ *(ms)*	$m=3$ *(ms)*	$m=4$ *(ms)*
$n=10$	**0.001**	**0.003**	**0.007**	**0.065**
$n=20$	**0.001**	**0.005**	**0.052**	**1.182**
$n=30$	**0.001**	**0.008**	**0.181**	**5.838**
$n=40$	**0.001**	**0.010**	**0.386**	**18.240**
$n=50$	**0.001**	**0.015**	**0.803**	**44.077**
$n=60$	**0.001**	**0.026**	**1.271**	**90.887**
$n=70$	**0.001**	**0.033**	**2.174**	**167.686**
$n=80$	**0.001**	**0.047**	**2.971**	**286.123**
$n=90$	**0.001**	**0.038**	**4.601**	**456.116**
$n=100$	**0.001**	**0.058**	**5.753**	**693.845**

TAB. 5.2 – Mesures de temps d'exécution pour un algorithme avec une complexité de $O(n^m)$

5.2.3 Discussion

Ces mesures de références sont très utiles pour estimer les temps d'exécution possibles pour un fédéré qui exécute un algorithme donné. Cela permet aussi d'évaluer les temps nécessaires pour transmettre des messages d'une taille donnée par le CERTI. Il faut préciser que ces estimations des temps d'exécution et de communications ne constituent pas une évaluation précise des WCETs des fédérés et des WCTTs des messages. Le calcul exact des WCETs et des WCTTs sont deux domaines de recherche indispensables au temps réel. Le calcul du WCET dépend de la structure du programme mais aussi de la structure matérielle d'exécution et en particulier du type de processeur(s) ou encore de la hiérarchie des caches mémoires [PK89, CPRS02, WEE+08]. Le calcul du WCTT dépend du protocole MAC et du support de communication utilisé [MZ95, VdC96]. Dans notre cas, nous nous basons sur ces différentes mesures pour évaluer les temps d'exécution des programmes et les temps de transit des messages pour nos simulations. Nous verrons dans le chapitre suivant (*C.f.* chapitre 6 , page 113) que ces mesures nous permettent d'affecter des valeurs pour les pires temps d'exécution des tâches et les pires temps de transfert des messages dans les modèles formels (le paramètre C_i des tâches et des messages). Bien entendu, ces estimations expérimentales pourraient être facilement remplacées par des évaluations plus précises des WCETs et des WCTTs.

5.3 Expérimentations pour le vol en formation

5.3.1 Choix de la configuration

Nous présentons dans cette partie les mesures pour la fédération de vol en formation de satellites (application CNES). Les mesures initiales [NdS08] avaient été réalisées sur du matériel standard en utilisant deux modes d'exécution : le mode d'exécution synchronisé par interaction (*C.f.* partie 4.3.3, page 82) et le mode d'exécution *Time-stepped* (*C.f.* partie 4.3.4, page 83) utilisant les mécanismes HLA de gestion du temps. Les différentes mesures présentées ici complètent les résultats précédents et ont été réalisées sur la plate-forme PRISE. Nous avons repris la fédération du cas simple (*C.f.* figure 4.3, page 73) en la faisant fonctionner sur un seul nœud de la plate-forme PRISE (*C.f.* figure 5.13). L'utilisation des nouvelles fonctionnalités du CERTI pour la gestion de l'affinité processeur nous a permis d'affecter les fédérés 1 et 2 sur le processeur `CPU0` (un des cœurs de la plateforme), leurs RTIAs respectifs sur le processeur `CPU3`, les fédérés 3 et 4 sur le processeur `CPU1`, leurs RTIAs respectifs sur le processeur `CPU3` et enfin le RTIG est affecté sur le processeur `CPU5`. Le processeur `CPU2` reste disponible, éventuellement pour exécuter les traitements propres au système d'exploitation. Nous utilisons les fonctionnalités de verrouillage

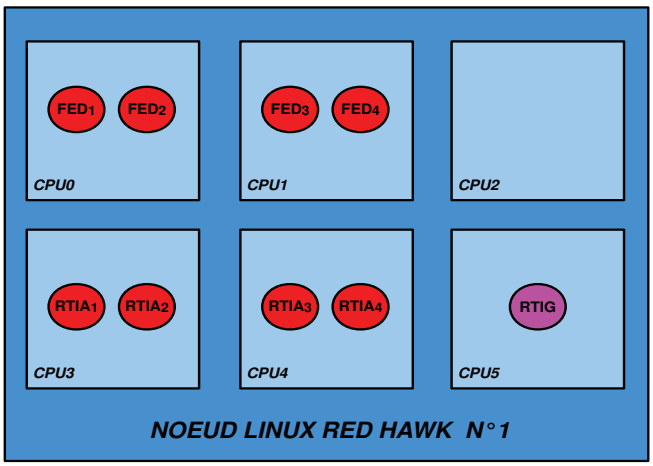


Fig. 5.13 – Affinité processeur pour la fédération cas simple sur un nœud Red Hawk

des pages mémoires dans la RAM ainsi que le nouveau service `tick2()` pour assurer la réception des callbacks attendues sans saturer le système d'exploitation. De plus, une étude préliminaire formelle, selon les différents principes détaillés dans le prochain chapitre (*C.f.* chapitre 6 , page 113), a aussi été réalisée pour assurer une affectation cohérente et déterministe des priorités pour les processus. Les résultats précédents avaient montré que le modèle par envoi d'interaction se comportait moins bien que le mode *Time-stepped*. Nous avons choisi de le remplacer par le mode d'exécution *Data Flow* (*C.f.* partie 4.3.2, page 81) ; ces résultats ont fait l'objet d'une communication dans une conférence internationale [CSAN11].

5.3.2 Mesures de la fédération *Data Flow*

La fédération cas simple est composée de 4 fédérés. Les fédérés 1 et 4 exécutent une boucle de 50 ms (soit 20 Hz) et les fédérés 2 et 3 exécutent une boucle de 10 ms (soit 100 Hz). Nous avons choisi que les temps d'exécution correspondent à environ 10% de leurs périodes de calcul. Ainsi,

en correspondance avec le tableau 5.1, les fédérés 1 et 4 réalisent un algorithme de $O(30^4)$ (ce qui représente un peu plus que 10% de 50 ms dans le pire des cas) et les fédérés 2 et 3 réalisent un algorithme de $O(10^4)$ (ce qui représente un peu moins que 10% de 10 ms dans le pire des cas). Sous ces hypothèses, le tableau 5.3 montre que les cycles de chaque fédérés sont respectés (10 et 50 ms) et que le comportement global est stable et cohérent.

	Minimum (ms)	**Moyenne (ms)**	**Maximum (ms)**	**Déviation Std.**
Fédéré 1	**49.394**	**49.449**	**49.585**	**0.059**
Fédéré 2	**9.056**	**9.119**	**9.458**	**0.127**
Fédéré 3	**9.058**	**9.131**	**9.501**	**0.146**
Fédéré 4	**49.01**	**49.077**	**49.150**	**0.058**

TAB. 5.3 – Mesure de la durée des cycles pour chaque fédéré (Data Flow)

Cette configuration, limitant les durées des exécutions pour les fédérés à environ 10% de leurs périodes de calcul, n'est pas très gourmande en consommation de la ressource processeur. Nous avons donc accéléré le rythme de l'application originelle pour l'autoriser à s'exécuter aussi rapidement que possible pour cette complexité des algorithmes de calcul. Pour cela, nous avons conservé les rapports entre les fréquences des différents fédérés, les fédérés 1 et 4 étant cinq fois plus lents que les fédérés 2 et 3. Ce principe est aussi valable pour les communications correspondantes. Le tableau 5.4 montrent que les fédérés les plus rapides (2 et 3) peuvent raisonnablement respecter une période de calcul d'environ 2 ms et les fédérés les plus lents (1 et 4) peuvent assurer le respect de périodes inférieures à 10 ms. Ces résultats montrent que le CERTI équipé de ses nouvelles fonctionnalités peut supporter des fédérés (en mode *Data Flow*) réalisant des calculs et des communications avec des fréquences très élevées.

	Minimum (ms)	**Moyenne (ms)**	**Maximum (ms)**	**Déviation Std.**
Fédéré 1	**6.108**	**6.136**	**6.292**	**0.0558**
Fédéré 2	**1.041**	**1.176**	**2.119**	**0.267**
Fédéré 3	**1.045**	**1.213**	**2.082**	**0.354**
Fédéré 4	**6.048**	**6.158**	**6.355**	**0.092**

TAB. 5.4 – Mesure de la durée des cycles pour chaque fédéré (Data Flow aussi rapide que possible)

5.3.3 Mesures de la fédération *Time Stepped*

Nous avons procédé de la même façon pour les expérimentations sur la fédération suivant le mode d'exécution *Time Stepped*. L'algorithme classique CMB (messages NULLs) mis en œuvre dans les mécanismes de gestion du temps du CERTI propose un comportement compatible avec les contraintes des simulations temps réel. Le tableau 5.5 montre que tous les cycles de calcul sont respectés (10 ms et 50 ms) et que le comportement global est également assez régulier même si certaines irrégularités apparaissent par rapport au mode d'exécution *Data Flow*.

	Minimum (ms)	Moyenne (ms)	Maximum (ms)	Déviation Std.
Fédéré 1	**48.64**	**49.765**	**50.807**	**0.532**
Fédéré 2	**9.514**	**9.592**	**10.618**	**0.172**
Fédéré 3	**9.372**	**9.624**	**10.959**	**0.248**
Fédéré 4	**48.029**	**49.474**	**50.787**	**0.841**

TAB. 5.5 – Mesure de la durée des cycles pour chaque fédéré (Time Stepped)

De la même façon que précédemment, nous avons accéléré le rythme de l'application en conservant la complexité des algorithmes de calcul choisis pour chaque fédéré. Le tableau 5.6 montre que les fédérés les plus rapides (2 et 3) peuvent raisonnablement respecter des périodes de calculs légèrement inférieurs à 7 ms et les fédérés les plus lents (1 et 4) peuvent assurer des cycles inférieurs à 15 ms. Ces résultats montrent que les mécanismes de gestion du temps mis en œuvre dans CERTI génèrent de la latence mais ils permettent de synchroniser efficacement des fédérés *Time Stepped* avec des fréquences élevées pour les calculs et les communications.

	Minimum (ms)	Moyenne (ms)	Maximum (ms)	Déviation Std.
Fédéré 1	**13.266**	**13.376**	**13.607**	**0.100**
Fédéré 2	**1.582**	**2.676**	**6.487**	**1.883**
Fédéré 3	**1.544**	**2.678**	**6.587**	**1.875**
Fédéré 4	**13.293**	**13.427**	**13.766**	**0.139**

TAB. 5.6 – Mesure de la durée des cycles pour chaque fédéré (Time Stepped aussi rapide que possible)

5.4 Expérimentations sur le simulateur de vol

5.4.1 Choix de configuration

Nous allons maintenant présenter les différents résultats et les mesures réalisées lors des tests du simulateur de vol (*C.f.* partie 4.2.2, page 74). Ce simulateur est une agrégation de différents fédérés HLA représentant les éléments essentiels pour le vol d'un avion. Au final, les 9 fédérés communiquant par le CERTI sont placés topologiquement sur deux supports matériels de la plateforme PRISE. Le fédéré Joystick ainsi que les deux fédérés de visualisation (3D et PFD) sont placés sur une plateforme graphique HP. Les six autres fédérés (Contrôleur, Dynamique du vol, Actuateur Moteur, Actuateur Mécavol, Capteurs et Environnement) sont placés sur un des nœuds Red Hawk. Les processus RTIAs correspondants sont situés sur les mêmes nœuds que leurs fédérés respectifs et le processus central RTIG est situé sur le nœud Red Hawk. Cette configuration est illustrée dans la figure 5.14.

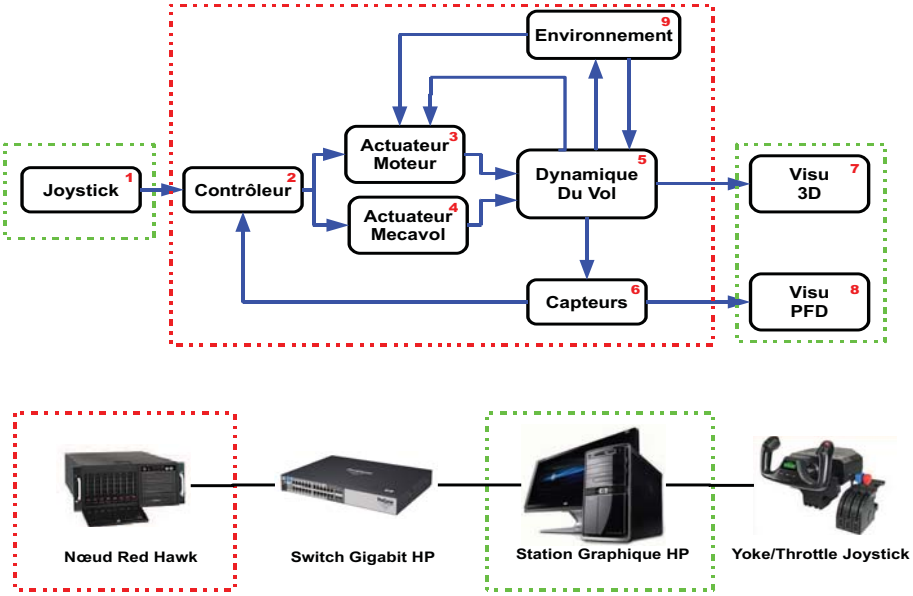


FIG. 5.14 – Configuration et placement des fédérés sur la plateforme PRISE

Sur le noeud Red Hawk, la disposition des fédérés correspond à l'illustration de la figure 5.15. Nous respectons la dissociation entre les processus d'exécution (les fédérés) et les processus de communication du CERTI (les RTIAs et le RTIG). Ici, la particularité est de laisser au processus RTIG la possibilité de s'exécuter sur trois processeurs (`CPU3`, `CPU4` et `CPU5`). Comme il est le processus avec la plus haute priorité, nous sommes sûrs qu'il sera activé dès qu'il sera sollicité pour une communication tout en diminuant certaines perturbations avec des RTIAs déjà en cours d'exécution.

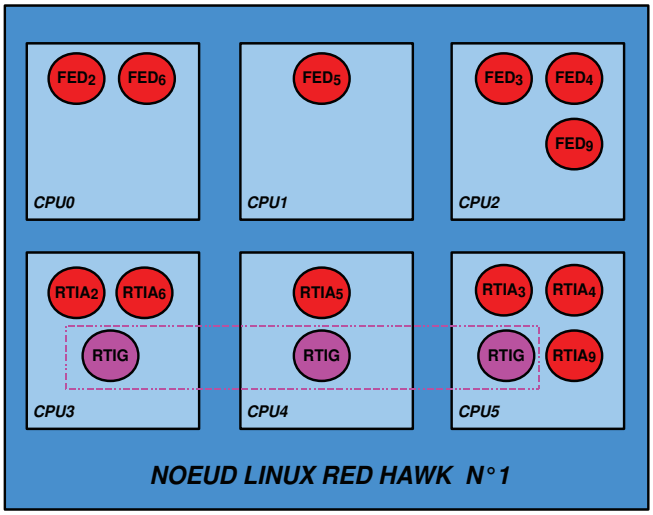


FIG. 5.15 – Affinité processeur pour la fédération du simulateur de vol sur un nœud Red Hawk

Pour l'exécution, nous avons choisi d'utiliser le mode d'exécution *Data Flow* et le mode d'exécution mixte (*C.f.* partie 4.3.6, page 86) due à la particularité de la résolution des équations différentielles pendant la simulation. Les résultats présentés dans ce chapitre ont aussi fait l'objet d'une communication dans une conférence internationale [CSSA11] ainsi que d'un article soumis dans le journal spécialisé SIMULATION [CSSA12].

5.4.2 Mesures sur les temps d'exécution des fédérés

Dans l'application du vol en formation de satellites, nous ne possédions pas les spécificités techniques des implémentations de chaque fédéré. Nous connaissions simplement les durées des temps de calculs et nous nous sommes basés sur les mesures de référence pour l'exécution de codes avec une complexité donnée (*C.f.* tableau 5.2, page 102). Pour le simulateur de vol, nous avons implémenté directement les algorithmes pour les calculs spécifiques à chaque fédéré. En conséquence, nous avons refait des mesures de temps d'exécution afin de dimensionner correctement notre application. Le fédéré Dynamique du Vol (*C.f.* tableau 5.7), le fédéré des capteurs ainsi que les fédérés actuateurs mécavol (*C.f.* tableau 5.9) et moteurs (*C.f.* tableau 5.10) implémentent des équations différentielles résolues à l'aide de la méthode numérique d'Euler[6] et leurs temps d'exécution dépendent du pas de temps choisi pour la discrétisation. Selon ce pas de temps, ils peuvent réaliser plusieurs itérations de résolution pour le même cycle de calcul.

Dynamique du vol	Minimum (ms)	Moyenne (ms)	Maximum (ms)	Déviation std.
100 itérations	0.96	0.963	1.004	0.005
500 itérations	4.743	4.753	4.78	0.0047
1000 itérations	9.58	9.703	9.751	0.048

TAB. 5.7 – Temps d'exécution pour le fédéré de dynamique du vol

Capteurs	Minimum (ms)	Moyenne (ms)	Maximum (ms)	Déviation Std.
100 itérations	0.077	0.0775	0.088	0.0016
500 itérations	0.377	0.37857	0.389	0.0020
1000 itérations	0.753	0.75472	0.764	0.0016

TAB. 5.8 – Temps d'exécution pour le fédéré des capteurs

Act. MecaVol	Minimum (ms)	Moyenne (ms)	Maximum (ms)	Déviation Std.
100 itérations	0.022	0.0224	0.024	0.0005
500 itérations	0.11	0.112	0.123	0.002
1000 itérations	0.217	0.221	0.244	0.0041

TAB. 5.9 – Temps d'exécution pour le fédéré des actuateurs (surfaces de contrôle)

Act. Moteur	Minimum (ms)	Moyenne (ms)	Maximum (ms)	Déviation Std.
100 itérations	0.024	0.02492	0.047	0.0028
500 itérations	0.123	0.1234	0.145	0.0028
1000 itérations	0.245	0.2461	0.27	0.0029

TAB. 5.10 – Temps d'exécution pour le fédéré des moteurs

[6] *Notons que d'autres méthodes de résolution numérique sont implémentées dans le code des différents fédérés (méthode des trapèzes, Adams Bashforth). Ces méthodes n'ont pas été utilisées pour les expérimentations qui sont présentées dans ce manuscrit.*

Le fédéré contrôleur, quant à lui, ne résout aucune équation différentielle et son temps d'exécution ne dépend donc pas d'un nombre d'itérations pour ses algorithmes locaux (*C.f.* tableau 5.11). Cette première version du contrôleur de vol est simplifiée avec des contrôles du vol latéral et longitudinal et un pilote automatique, les temps de calcul sont, par conséquent, relativement petits. Les autres fédérés réalisent des calculs qui peuvent être négligeables par rapport aux temps de transfert nécessaires pour des communications haute fréquence avec le CERTI.

Contrôleur	Minimum (ms)	Moyenne (ms)	Maximum (ms)	Déviation Std.
Calcul	**0.01**	**0.07**	**0.3**	**0.005**

TAB. 5.11 – Temps d'exécution pour le fédéré Contrôleur

5.4.3 Mesures des cycles de la fédération

Le simulateur de vol met en œuvre des fédérés avec des durées de cycles différents. Ainsi, le fédéré contrôleur fonctionne sur une boucle de calcul de 20 ms (*i.e.* une fréquence de 50 Hz) et cependant il va recevoir des données à partir du fédéré simulant les capteurs qui fonctionne avec une boucle de calcul de 10 ms (*i.e.* une fréquence de 100 Hz). Ainsi, lorsque le contrôleur reçoit des données provenant des capteurs, les fédérés cadencés à 100 hz ont accompli un cycle complet (calculs et communications). Lors de son propre cycle de calcul, le fédéré contrôleur peut mesurer les temps nécessaires pour exécuter deux cycles complets de la simulation. Cette particularité sur les cycles (notés cycles 1 et 2) est illustrée dans la figure 5.16. Chacun de ces cycles doit ainsi être accompli avant un délai de 10 millisecondes pour garantir les propriétés temps réel de la simulation.

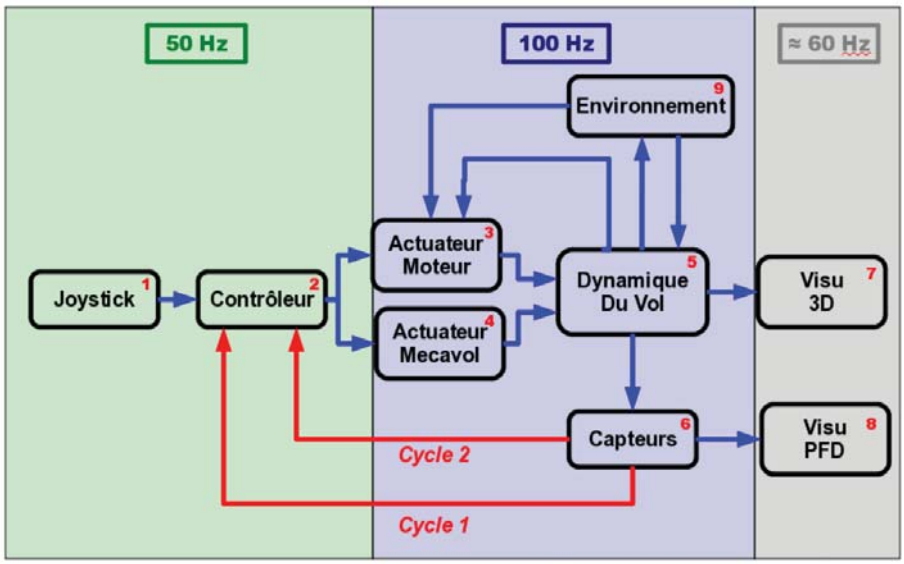


FIG. 5.16 – Les deux cycles mesurés sur le simulateur de vol

La fédération exécutée selon le mode *Data Flow* a un comportement assurant le respect de ses contraintes temps réel. La figure 5.17 et le tableau 5.12 montrent que les deux cycles respectent le délai de 10 ms. De plus, le comportement global de la fédération est stable.

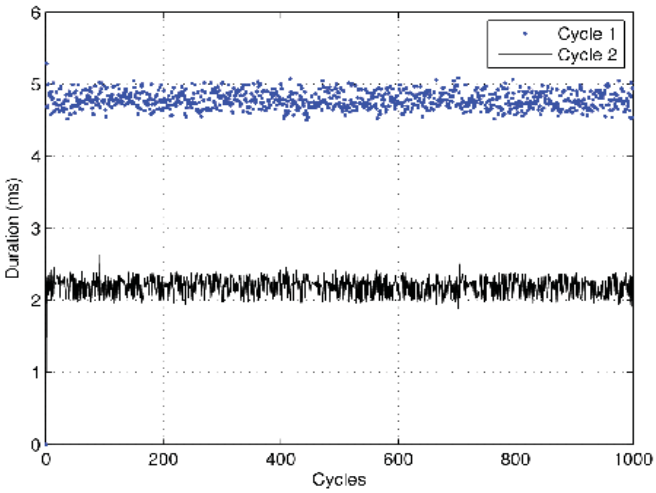


FIG. 5.17 – Comportement du simulateur selon le mode *Data Flow*

Mode *Data Flow*	Minimum (ms)	Moyenne (ms)	Maximum (ms)	Déviation Std.
Cycle 1	**4.50**	**4.78**	**5.30**	**0.11**
Cycle 2	**1.88**	**2.19**	**2.65**	**0.12**

TAB. 5.12 – Comportement temporel du mode d'exécution *Data Flow*

Pour le mode d'exécution mixte utilisant les mécanismes HLA de la gestion du temps simulé, l'algorithme CMB classique (algorithme des messages NULL), mis en œuvre dans CERTI, semble avoir un bon comportement pour assurer les propriétés temps réel nécessaires à notre simulateur. En effet, la figure 5.18 et le tableau 5.13 montrent que tous les cycles de calcul respectent le délai imposé de 10 ms. Le comportement global de l'ensemble de la simulation paraît régulier. La combinaison particulière des appels aux services HLA (`NER()` et `TAR()`) pour avancer dans le temps simulé, assurer la synchronisation et la livraison des messages semble produire certaines latences. Cependant, dans ce cas, le nombre de messages additionnels générés par l'algorithme CMB est acceptable et ne limite pas les propriétés temps réel du simulateur de vol.

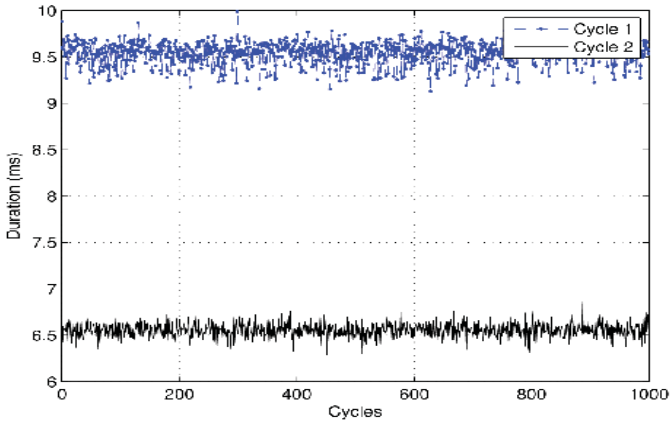


FIG. 5.18 – Comportement du simulateur selon le mode mixte (CMB classique)

Mode Mixte (CMB classique)	Minimum (ms)	Moyenne (ms)	Maximum (ms)	Déviation Std.
Cycle 1	**9.13**	**9.54**	**9.99**	**0.11**
Cycle 2	**6.28**	**6.55**	**6.85**	**0.07**

TAB. 5.13 – Comportement temporel du mode d'exécution mixte (CMB classique)

Nous avons implémenté et mis en œuvre un nouvel algorithme noté CMB ONERA et appelé aussi l'algorithme NULL Message PRIME [CSN11]. Ce nouvel algorithme permet d'optimiser l'échange des messages additionnels sur le réseau entre les fédérés utilisant le service `nextEventRequest()` lors de l'exécution de simulation. Les spécificités de ce nouvel algorithme sont présentées dans la suite du document (*C.f.* partie 6.4.2, page 138). Pour le simulateur de vol, nous l'avons utilisé pour exécuter et synchroniser notre fédération suivant le mode d'exécution mixte. La figure 5.19 et le tableau 5.14 montrent que les calculs supplémentaires, dus aux calculs du RTIG pour établir l'état global, génèrent une surcharge pour le premier cycle qui dépasse parfois légèrement son échéance de 10 ms. Inversement, l'utilisation de l'algorithme permet de réduire la latence globale pour le deuxième cycle. Cette nouvelle méthode de synchronisation permet d'assurer des propriétés temps réel, cependant nous continuons nos travaux afin de comprendre et de quantifier les sources de latences et ainsi pouvoir les borner. Toutefois, nous pensons que ce nouvel algorithme fournirait de meilleures résultats que l'algorithme original sur des simulations de plus grande échelle (exemple : une simulation collaborative de type WAN).

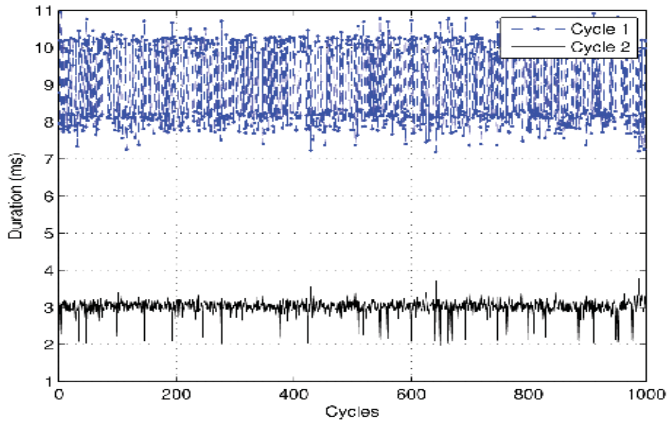


FIG. 5.19 – Comportement du simulateur selon le mode mixte (CMB ONERA)

Mode mixte (CMB ONERA)	Minimum (ms)	Moyenne (ms)	Maximum (ms)	Déviation Std.
Cycle 1	**7.19**	**8.99**	**10.94**	**1.07**
Cycle 2	**1.96**	**3.01**	**3.78**	**0.20**

TAB. 5.14 – Comportement temporel du mode d'exécution mixte (CMB ONERA)

5.4.4 Mesures de simulation

Nous terminons cette partie en présentant et comparant certaines réponses du simulateur de vol à une demande d'altitude selon les deux modes d'exécution. Le pilote automatique mis en œuvre dans le fédéré contrôleur est activé et réglé sur le maintien d'une altitude donnée. L'avion se trouvant initialement à une altitude de 10200 mètres avec une vitesse de 250 mètres par seconde, doit atteindre une altitude de croisière de 10000 mètres. Dans les deux cas, l'avion atteint cette altitude de croisière en douceur (*C.f.* figure 5.20) en utilisant principalement le braquage de la gouverne de profondeur (*C.f.* figure 5.21) . Les temps des réponses sont similaires entre les deux modes d'exécution et représentatifs du comportement attendu, qui avait été précédemment validé sous Matlab/Simulink.

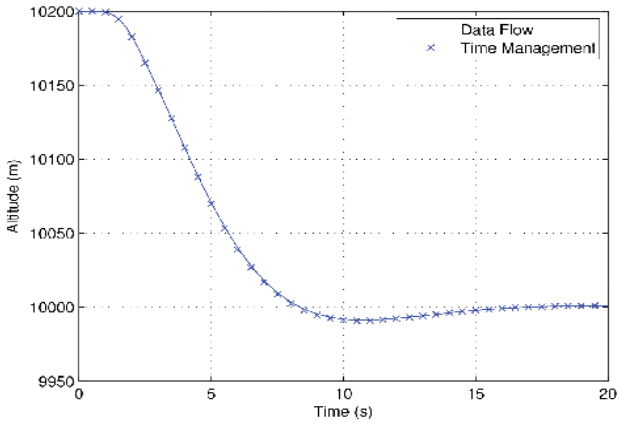


FIG. 5.20 – Temps de réponse au changement d'altitude

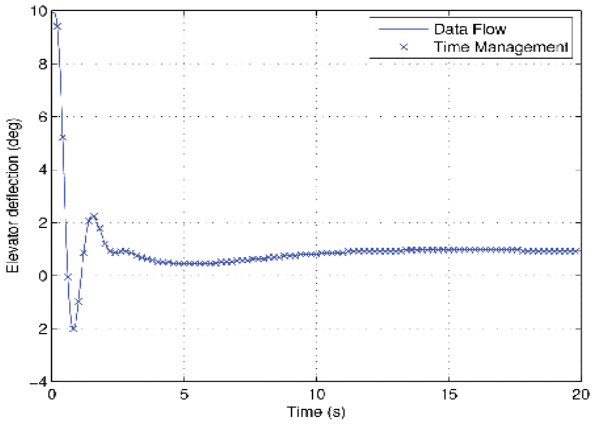


FIG. 5.21 – Mesure de l'angle de braquage de la gouverne de profondeur

Chapitre 6

Validation formelle d'une simulation distribuée HLA

Préambule

Au même titre que tous les systèmes temps réel, les simulations HLA temps réel sont soumises à des contraintes temporelles et doivent donc assurer un comportement temporel prédictible. Ce comportement temporel peut être prouvé formellement en utilisant des techniques de validation issues de la théorie de l'ordonnancement. Ainsi dans un premier temps, nous expliquerons les hypothèses de départ retenues pour valider formellement nos simulations. Cette validation formelle étant dépendante du mode d'exécution choisi, nous présenterons ensuite les différentes techniques et méthodes utilisées pour la validation des simulations utilisant le mode d'exécution Data Flow. Puis, nous expliquerons les différents travaux menés sur les fédérations composées de fédérés ayant le mode d'exécution Time Stepped basé sur les mécanismes de gestion du temps. Nous présenterons notamment différentes métriques que nous avons établies et qui permettent de quantifier l'échange de messages durant la simulation. Enfin, nous terminerons par les différentes études menées sur les mécanismes de gestion du temps suivant le modèle Event Driven. ainsi que le nouvel algorithme distribué développé et mis en œuvre dans le CERTI pour la gestion du temps.

6.1 Hypothèses de base

6.1.1 Nécessité d'une validation formelle

Comme nous l'avons précisé précédemment, les systèmes informatiques temps réel se différencient des autres systèmes informatiques par la prise en compte de contraintes temporelles dont le respect est aussi important que l'exactitude du résultat fourni par le système. La théorie de l'ordonnancement (expliquée en détail dans l'annexe C, page 173) fournit un ensemble de méthodes algébriques et algorithmiques permettant de vérifier le comportement temporel d'une application temps réel. Les recherches menées dans le cadre du développement de la norme RT CORBA (*C.f.* partie 3.2.2, page 50), tels que l'article très complet [DWE+01], ont permis d'aboutir à des modèles formels permettant de valider formellement une application distribuée CORBA. L'ensemble des travaux à été pérennisé par l'inclusion d'un certain nombre de recommandations, concernant la validation formelle ainsi que les techniques de mises en œuvre, dans les documents de la norme.

Par exemple, ce type d'étude formelle nous permet de définir une allocation adéquate des différents processus sur les processeurs du système ainsi qu'une affectation de priorité. À notre connaissance, aucun travail de la communauté de la simulation HLA n'a relié des modèles formels et des méthodes de la théorie de l'ordonnancement avec les concepts et les principes de la simulation distribuée. Certains travaux évoquent l'utilisation d'un service d'ordonnancement global pour la RTI comme ceux de Azzedine Boukerche [BL05, BSZ07] ou encore de Hui Zhao [ZG01b, Zha01] mais la mise en œuvre de modèles ou de méthodes formelles n'est jamais présentée en détail. Généralement, les simulations HLA temps réel sont validées par des expérimentations pour vérifier que les fédérés respectent bien leurs échéances temporelles. En ce sens, le chapitre précédent nous a permis de voir que les exécutions de nos simulations selon des modes d'exécution spécifique permettaient d'assurer des simulations composées de fédérés avec des cycles très courts. Nous donnerons ici les pistes pour réussir à valider formellement une simulation distribuée. L'exemple de la norme RT-CORBA montre la nécessité de modèles formels, des méthodes et des tests de validation de la théorie de l'ordonnancement pour valider formellement le comportement temps réel d'une simulation respectant le standard HLA. Une telle approche est essentielle pour envisager l'intégration dans le futur de recommandations adaptées aux spécificités des simulations aux différents documents de la norme HLA. Cela permettrait d'établir un canevas formel permettant la validation du comportement temporel de tout type de simulation HLA. Dans ce travail, nous nous concentrerons sur les méthodes statiques qui furent, historiquement, les premières à être intégrées dans la norme RT-CORBA. De plus, les méthodes statiques sont les plus utilisées dans le domaine des systèmes embarqués car elles ont un comportement temporel prédictible et reproductible. Nous pouvons ainsi compléter le dernier niveau de notre approche méthodologique (*C.f.* partie 4.1.1, page 69) comme cela est montré sur la figure 6.1.

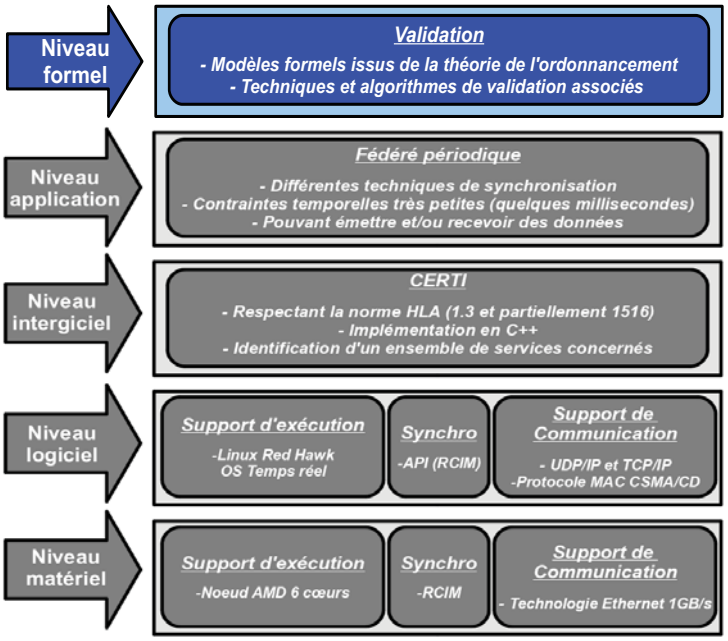


Fig. 6.1 – Réponse du niveau formel

6.1.2 Granularité du modèle

Afin de mettre en place un modèle ainsi que des méthodes associées qui soient adaptées au système temps réel, il faut définir les entités que l'on souhaite ordonnancer. Dans la norme RT-CORBA, les entités ordonnançables sont les threads qui sont des processus légers (partageant un espace mémoire commun). Dans notre cas, chaque fédéré est un processus (*i.e.* un processus lourd) qui possède son propre espace mémoire et d'adressage. Idéalement, nos fédérés peuvent être vus comme des processus communicants comme cela est illustré dans la figure 6.2.

FIG. 6.2 – Vision idéale de deux processus fédérés communicants

Cependant, lorsque l'on utilise le CERTI pour gérer les communications entre ces processus périodiques, son architecture a un impact sur le nombre de processus à prendre en compte dans le problème d'ordonnancement posé. En effet, comme cela est représenté sur la figure 6.3, différents processus supplémentaires sont créés par le CERTI afin d'assurer les communications. Chaque communication est alors composée de trois phases :

1. l'écriture sur la socket Unix par le fédéré et la lecture sur cette socket et le calcul du RTIA correspondant ;
2. l'écriture sur la socket UDP ou TCP par le RTIA, la lecture et le calcul du processus global RTIG et enfin l'écriture sur la socket TCP ou UDP de l'autre RTIA ;
3. la lecture sur la socket TCP ou UDP par le RTIA , le calcul et l'écriture sur la socket Unix de son fédéré ainsi la lecture de ce fédéré.

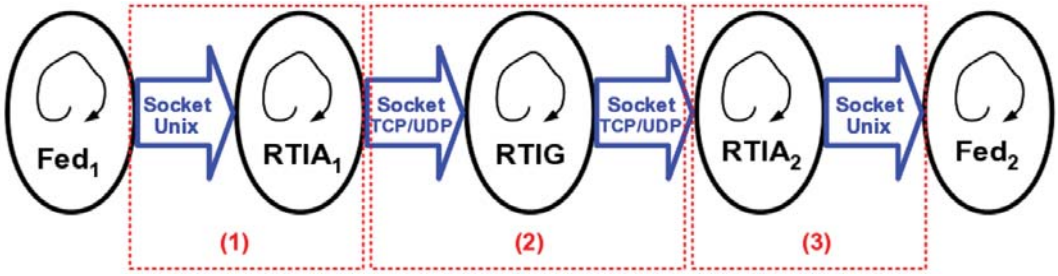


FIG. 6.3 – Vision concrète de deux processus fédérés communicants avec le CERTI

Ainsi, pour chaque processus fédéré, il existera un processus RTIA directement lié au fédéré ainsi que le processus central RTIG. Donc, pour N processus périodiques (les fédérés HLA dans notre cas) qui communiquent par le CERTI, cela donne en réalité $2N+1$ processus à ordonnancer sur l'ensemble du système distribué. Pour ce travail, nous avons décidé de séparer physiquement les processus d'exécution (les fédérés) des processus du CERTI gérant la communication (les RTIAs et le RTIG). Dans le modèle formel, cela nous permet de modéliser le système avec une granularité élevée en considérant les calculs relatifs aux fédérés comme des exécutions et les

transports de messages (calculs des processus RTIAs et RTIG et utilisation des sockets) comme des communications (*C.f.* figure 6.4).

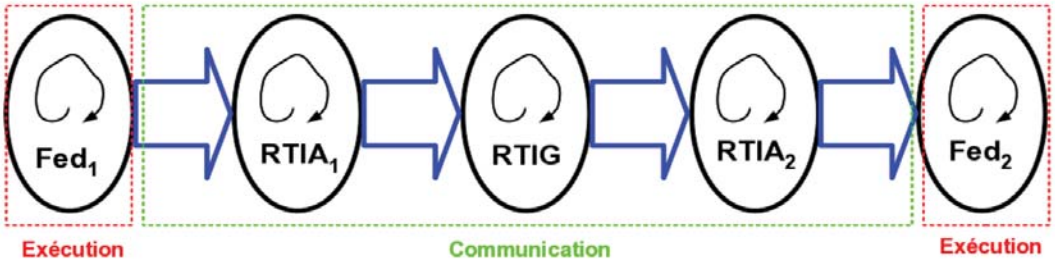


FIG. 6.4 – Niveau de granularité choisi

Cette distinction repose en partie sur le fait que nous considérons que le CERTI et les protocoles réseau qu'il met en œuvre sont assez performants pour assurer des communications déterministes. C'est à dire que nous supposons que dans tous les cas les temps de transfert des messages au travers du CERTI seront toujours connus et bornés tout en respectant les échéances attendues. Si cette hypothèse n'est plus imposée, il faudra travailler sur l'intégration de nouveaux protocoles de communications car les protocoles UDP et TCP ne sont pas des protocoles temps réel déterministes comme peuvent l'être CAN[1] [ISO99, THW94] ou TDMA[2] [IEC00]. De plus, il faudra prendre en compte l'exécution de chaque processus comme une exécution et donc il pourra s'avérer nécessaire de descendre le niveau de granularité en considérant chaque processus comme une seule tâche et chaque communication par socket comme un seul message (*C.f.* figure 6.5).

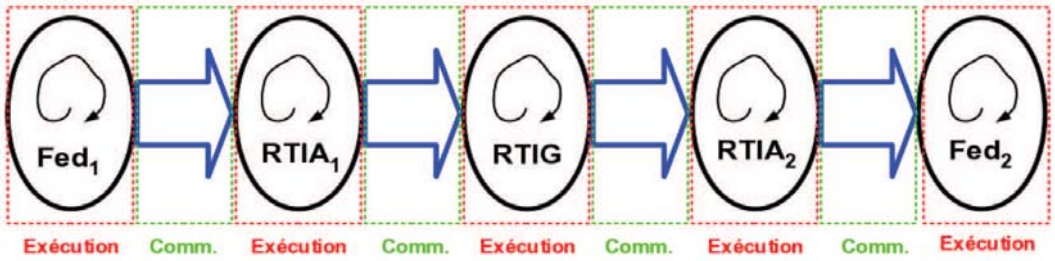


FIG. 6.5 – Vers un niveau de granularité plus fin

Dans tous les cas, l'approche que nous détaillons dans la suite de ce chapitre restera valable et adaptable à un niveau de granularité plus faible. Les techniques d'analyse et de validation pour nos simulations sont adaptées à l'architecture de CERTI mais cette approche pourra aussi être utilisée pour d'autres concepteurs de simulation HLA. En effet, la plupart des RTIs sont basées sur le principe d'un composant local LRC[3] associé à chaque fédéré et un composant central CRC[4] qui sert de passerelle de transmission. Ainsi, les principes des techniques de validation proposées pour le CERTI devront être adaptables de façon générale à tout type de simulation HLA (indépendamment de l'intergiciel RTI utilisé).

[1] *Carrier Sense Multiple Access with Collision Avoidance*
[2] *Time Division Multiple Access*
[3] *Local Run-Time Component*
[4] *Central Run-Time Component*

6.1.3 Ordonnancement fonctionnel et temporel

L'ordonnancement d'une application est composé de deux facettes complémentaires qu'il faut distinguer et que nous appelons : l'ordonnancement fonctionnel et l'ordonnancement temporel.

L'ordonnancement fonctionnel de l'application permet d'assurer que le cadencement des communications et des exécutions de la simulation globale fournira des résultats qui seront toujours justes et pertinents. Par exemple, dans le cas du simulateur de vol, chaque fédéré implémentait un algorithme spécifique. Ainsi, les résolutions complémentaires des différents algorithmes pendant une exécution globale de simulation impose de garantir un séquencement des exécutions et des communications de manière à assurer une simulation correcte pour le vol d'un avion.

L'ordonnancement temporel est propre à la théorie de l'ordonnancement. Ici, l'objectif est de vérifier formellement que la simulation distribuée s'exécutera tout en assurant le respect de des contraintes temporelles auxquelles elle est soumise. L'objectif est donc de garantir que la simulation fournira des résultats dans les bons temps. L'assurance d'un bon ordonnancement fonctionnel et temporel assurera à nos simulations de toujours donner des résultats justes et dans les bons temps. Pour nos cas tests, la simulation du vol en formation ne mettait en œuvre que la partie de l'ordonnancement temporel tandis que le simulateur de vol met en œuvre les deux types d'ordonnancement car nous possédons la sémantique des codes de calculs associés aux différents fédérés.

6.2 Étude sur les fédérations *Data Flow*

6.2.1 Modèle utilisé

Le mode d'exécution *Data Flow* (*C.f.* partie 4.3.2, page 81) montre que chaque fédéré réalise périodiquement des calculs et aussi qu'il reçoit et publie périodiquement des données. Nous pouvons donc assimiler un fédéré ayant le mode d'exécution *Data Flow* comme une tâche périodique et les différentes tâches de la fédération devront se partager les unités d'exécution du système informatique (les processeurs). Nous reprenons donc le modèle classique d'une tâche périodique et nous considérons qu'un fédéré noté Fed_i est une tâche périodique définie par le quadruplet $< r_i, C_i, D_i, P_i >$ tel que :

1. r_i moment de l'activation initiale de la tâche ;
2. C_i la durée d'exécution pire cas de la tâche (WCET[5] du fédéré) ;
3. D_i la deadline, la durée limite à ne pas dépasser ;
4. P_i la période d'activation de la tâche.

Ces différentes tâches communiquent en utilisant les services HLA fournis par le CERTI. Ces communications peuvent être représentées par des messages périodiques qui représentent l'échange de données entre deux fédérés. Un message périodique est décrit de la même manière qu'une tâche par un quadruplet $< r_{m_{i,j}}, C_{m_{i,j}}, D_{m_{i,j}}, P_{m_{i,j}} >$ tel que :

1. $r_{m_{i,j}}$ moment du premier envoi du message par un fédéré ;
2. $C_{m_{i,j}}$ la durée de transfert pire cas (WCTT[6] de la transmission du message CERTI) ;
3. $D_{m_{i,j}}$ la deadline, l'instant limite à ne pas dépasser pour la livraison du message ;
4. $P_{m_{i,j}}$ la periode de production du message.

[5] *Worst Case Execution Time*
[6] *Worst Case Transit Time*

La combinaison de ces deux modèles va nous permettre de décrire nos simulations distribuées. En respectant la séparation physique des processus d'exécution (les fédérés) des processus de communication du CERTI (les RTIAs et le RTIG), nous évitons les interférences entre les communications et les exécutions dans le modèle. Les messages sont vus comme des tâches qui se partagent les ressources du CERTI. Nous considérons que le CERTI et les protocoles réseau qu'il supporte sont assez performants pour être considérés comme des communications déterministes. Et en ce sens, nous basons les pires temps de transferts des messages sur les mesures réalisées sur PRISE (*C.f.* tableau 5.1, partie 5.2.1, page 101). Chaque communication entre les fédérés périodiques peut être représentée par une flèche. Ces communications sont vues comme des dépendances de données qui imposent des contraintes de précédences entre les fédérés. En effet, lors de l'exécution de la simulation, un fédéré aura besoin de recevoir les données correspondantes à son cycle courant avant de réaliser ses calculs locaux (déroulement interne de son algorithme). Ces messages sont donc vus comme des relations de précédence entre les tâches [CSA10].

6.2.2 Analyse Holistique

L'analyse holistique a été introduite par Ken Tindell et John Clark [TC94, Tin94] pour intégrer la dépendance entre l'ordonnancement des tâches et des messages dans un système distribué temps réel possédant une horloge globale. Pour un ensemble donné de tâches, les dépendances de données sont représentées par un ensemble de messages. Toutes les tâches peuvent être retardées par cette dépendance de données. Pour cela, l'analyse holistique introduit deux paramètres supplémentaires à la description des tâches et des messages :

1. J_i pour les tâches[7] correspond au pire temps de gigue sur l'activation, c'est la différence entre la date d'activation r_i et la date de réveil effective de la tâche.
2. R_i pour les tâches[8] correspond au pire temps de réponse de la tâche, c'est la différence entre la date d'activation r_i et la fin de son exécution.

Considérons un exemple simple avec deux fédérés Fed_1 et Fed_2 ayant la même période de calcul (10 ms). De plus, Fed_1 envoie un message $m_{1,2}$ au fédéré Fed_2 sur cette même période. Ce fonctionnement simple est illustré sur la figure 6.6. L'illustration correspondante pour les gigues sur l'activation et les temps de réponses des tâches et du message est sur la figure 6.7.

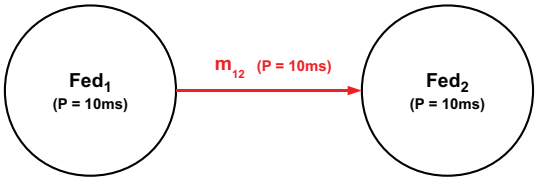


FIG. 6.6 – Deux fédérés Fed_1 et Fed_2

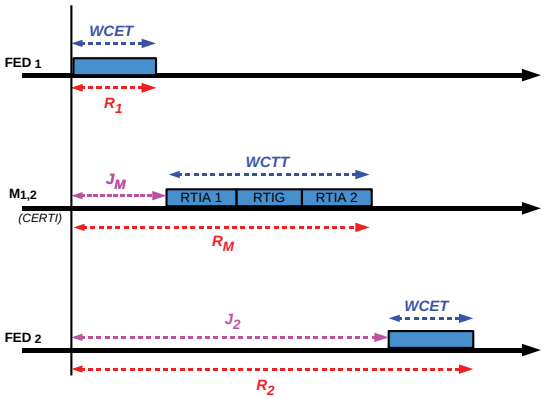


FIG. 6.7 – Gigue et temps de réponse

[7] *respectivement* $J_{m_{i,j}}$ *pour les messages*
[8] *respectivement* $R_{m_{i,j}}$ *pour les messages*

Les fédérés Fed_1 et Fed_2 sont chacun situés sur leur propre processeur et les processus du CERTI sur d'autres processeurs (isolation des exécutions et des communications). Considérons que, au pire des cas, le fédéré Fed_1 met 4 millisecondes pour s'exécuter, le message $m_{1,2}$ met 2 millisecondes pour transiter à travers le CERTI et les couches réseau et, enfin, le fédéré Fed_2 s'exécute avec un temps de 3 millisecondes. L'objectif est de déterminer les paramètres J_i et R_i pour les tâches et les messages. La date d'émission du message (sa gigue) est calculée en fonction du temps de réponse de la tâche Fed_1 et donc $J_{m_{1,2}} = 4$. De même, la date de réveil de la tâche réceptrice dépendra du temps de réponse de la tâche à recevoir donc $J_2 = R_{m_{1,2}} = C_1 + C_{m_{1,2}} = 6$. Au final, le système peut être décrit de la façon suivante :

- Une tâche émettrice $Fed_1 :< r_1 = 0, C_1 = 4, J_1 = 0, R_1 = 4, D_1 = 10, P_1 = 10 >$;
- Un message $m_{1,2} :< r_{m_{1,2}} = 0, C_{m_{1,2}} = 2, J_{m_{1,2}} = 4, R_{m_{1,2}} = 6, D_{m_{1,2}} = 10, P_{m_{1,2}} = 10 >$;
- Une tâche receptrice $Fed_2 :< r_2 = 0, C_2 = 3, J_2 = 6, R_2 = 9, D_2 = 10, P_2 = 10 >$.

L'objectif de cette méthode est d'assurer que chaque tâche et chaque message respectera ses échéances temporelles et donc que : $\forall\, i,\, R_i < D_i$[9]. Nous voyons ici que les temps de réponse pour chaque tâche et chaque message sont inférieurs aux délais correspondants et, par conséquent, le comportement temporel de ce système simple est validé formellement. En ce sens, l'ajout de ces nouveaux paramètres permet de représenter formellement les dépendances explicites de données entre l'ordonnancement des tâches et des messages puis de les intégrer dans une démarche de validation globale d'un système temps réel distribué.

Pour les systèmes plus complexes, l'analyse holistique permet de calculer les paramètres R_i et J_i pour chaque tâche et chaque message par la résolution de systèmes d'équations récurrentes. Ainsi toute modification de la gigue d'une tâche va modifier les temps de réponse des tâches dépendantes qu'elle précède ainsi que ceux des tâches ordonnancées sur le même processeur. Dans un système composé de N tâches et messages, on note $ResponseTimes()$ la fonction de calcul des temps de réponse et $prec(i)$ la fonction qui retourne l'ensemble des indices relatifs aux tâches précédentes à la tâche i. Ainsi, le système d'équations récurrentes de l'analyse holistique peut être écrit comme ceci :

$$\begin{cases} R_i^{(0)} &= C_i \\ J_i^{(0)} &= 0 \\ R_i^{(k)} &= ResponseTimes\big(J_i^{(k-1)}\big) \\ J_i^{(k)} &= \max\limits_{j \in prec(i)} \big(R_j^{(k)}\big) \end{cases} \tag{6.1}$$

Ces équations se résolvent récursivement jusqu'à ce qu'un point fixe soit atteint (chaque itération est notée par l'entier k). Ce point fixe apparaît lorsque pour un entier k on obtient pour chaque tâche i : $\forall\, i,\, R_i^{(k)} = R_i^{(k-1)}$[10]. L'ensemble des caractéristiques de ces résolutions, notamment pour l'implémentation de la fonction $ResponseTimes()$ selon les algorithmes d'ordonnancement de chaque site, peut être trouvé dans le document très détaillé suivant [RRC03b].

[9] *Pour les messages :* $\forall\, i, j,\, R_{i,j} < D_{i,j}$

[10] *Pour les messages :* $\forall\, i, j,\, R_{i,j}^{(k)} = R_{i,j}^{(k-1)}$

Dans notre cas, nous déroulons ces algorithmes à la main ou encore nous utilisons le logiciel open-source CHEDDAR [SLNM04, Sin07] [10] développé et maintenu au sein de l'équipe LISyC[11] de l'université de Brest.

6.2.3 Discrétisation

Les précédences entre les tâches (dues aux communications) ne sont pas toutes aussi simples que celles décrites dans la figure 6.6 où la tâche émettrice, la tâche réceptrice et le message ont la même période. Ce type de précédence, appelé une précédence simple, impose que la période (et donc la fréquence) du fédéré producteur, du fédéré récepteur ainsi que du message soient égales. Cela est assez restrictif car dans un système réel les tâches et les messages peuvent mettre en œuvre des mécanismes de communication plus complexes. Un travail récent réalisé en partie à l'ONERA [FBG+10] a étudié les différentes techniques de prise en compte du problème des précédences complexes. Cela permet une modélisation plus souple du système en n'imposant aucune restriction sur les différentes périodes. En revanche, les techniques pour la prise en compte de ces contraintes sont beaucoup plus délicates à mettre en œuvre et particulièrement dans le cas d'un système distribué. L'analyse holistique de Tindell et Clark est, quant à elle, basée sur le modèle des précédences simples ; nous allons donc dans cette partie, expliquer les différentes techniques de discrétisation pour ramener un problème avec des précédences complexes à un problème de précédences simples. L'idée de base revient à discrétiser l'ensemble de tâche sur un intervalle d'étude $[0, PPCM]$. Le PPCM[12] étant l'hyper-période relative aux périodes de toutes les tâches et de tous les messages. Chaque tâche i de base est ainsi découpée en nombre N_i de sous-tâches avec $N_i = \frac{PPCM}{P_i}$. Cette notion de sous-tâche est relativement proche de la notion d'instance de tâche mais offre la possibilité de pouvoir affecter une priorité différente à chacune des sous-tâches. Ces sous-tâches représentent les différents cycle d'exécution du fédéré sur l'intervalle d'étude. Afin de respecter la périodicité des tâches initiales, nous avons modifié les dates de réveil des sous-tâches et donc les sous-tâches produites ne respectent plus l'hypothèse de l'instant critique selon lequel on doit respecter : $\forall i$ et $\forall j$, $r_i = r_{m_{i,j}} = 0$. Ainsi, les techniques classiques telles que les algorithmes classiques tels que le Rate Monotonic [LL73] ou le Deadline Monotonic [LW82] ne fonctionnent plus pour affecter une priorité à chacune des sous-tâches. De plus, nous avons dû modifier la méthode originale de Tindell et Clark. Tous les détails de ces ajouts ainsi que des adaptations algorithmiques sont données dans la partie 6.2.4. Ici, nous détaillons tous les cas possibles pour la discrétisation en nous basant toujours sur un exemple simple mettant en œuvre deux fédérés qui communiquent.

Cas 1 : La tâche émettrice a une période plus petite que le message et la tâche réceptrice et le message (*C.f.* figure 6.8) :

- Une tâche émettrice $Fed_1 :< r_1 = 0, C_1 = wcet(Fed_1), D_1 = 10, P_1 = 10 >$;
- Un message $m_{1,2} :< r_{m_{1,2}} = 0, C_{m_{1,2}} = wctt(m_{1,2}), D_{m_{1,2}} = 20, P_{m_{1,2}} = 20 >$;
- Une tâche réceptrice $Fed_2 :< r_2 = 0, C_2 = wcet(Fed_1), D_2 = 20, P_2 = 20 >$.

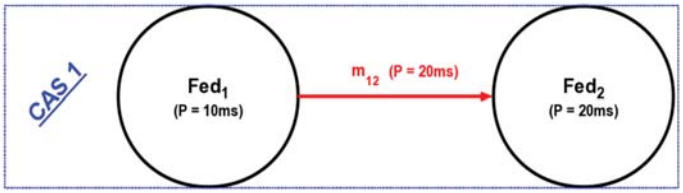


FIG. 6.8 – Illustration du cas 1

[11] *Laboratoire d'Informatique des Systèmes Complexes*
[12] *Plus Petit Commun Multiple,* $\forall\ i,\ PPCM = PPCM(P_i)$

Dans ce cas, on discrétise la tâche émettrice Fed_1 en deux sous-tâches Fed_1^1 et Fed_1^2. On obtient le système suivant :

- Une tâche $Fed_1^1 :< r_1^1 = 0, C_1^1 = wcet(Fed_1), D_1^1 = 10, P_1^1 = 20 >$;
- Une tâche $Fed_1^2 :< r_1^2 = 10, C_1^2 = wcet(Fed_1), D_1^2 = 20, P_1^2 = 20 >$;
- Un message $m_{1,2} :< r_{m_{1,2}} = 0, C_{m_{1,2}} = wctt(m_{1,2}), D_{m_{1,2}} = 20, P_{m_{1,2}} = 20 >$;
- Une tâche $Fed_2^1 :< r_2^1 = 0, C_2^1 = wcet(Fed_2), D_2^1 = 20, P_2^1 = 20 >$.

Deux choix sont alors possibles pour effectuer la communication (*C.f.* figure 6.9) et ce choix dépend des principes d'ordonnancement fonctionnel mis en œuvre dans la simulation. Ces choix doivent être étudiés et évalués par le concepteur de la fédération. Toutefois, ici, le choix 1 paraît le plus judicieux pour assurer un ordonnancement temporel correct pour l'application.

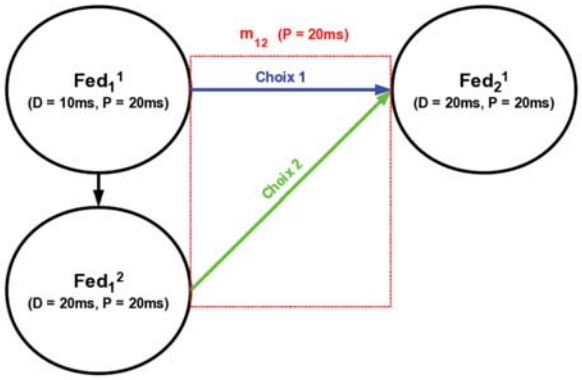


FIG. 6.9 – Solutions de discrétisation pour le cas 1

Cas 2 : La tâche réceptrice a une période plus petite que le message ainsi que la tâche émettrice (*C.f.* figure 6.10) :

- Une tâche émettrice $Fed_1 :< r_1 = 0, C_1 = wcet(Fed_1), D_1 = 20, P_1 = 20 >$;
- Un message $m_{1,2} :< r_{m_{1,2}} = 0, C_{m_{1,2}} = wctt(m_{1,2}), D_{m_{1,2}} = 20, P_{m_{1,2}} = 20 >$;
- Une tâche réceptrice $Fed_2 :< r_2 = 0, C_2 = wcet(Fed_2), D_2 = 10, P_2 = 10 >$.

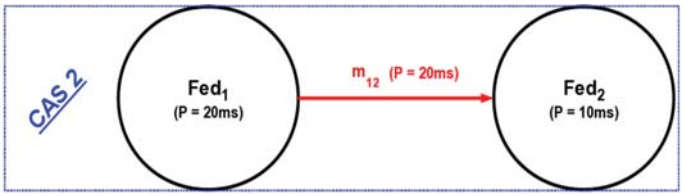


FIG. 6.10 – Illustration du cas 2

Dans ce cas, on discrétise la tâche réceptrice Fed_2 en deux sous-tâches Fed_2^1 et Fed_2^2 et on obtient le système suivant :

- Une tâche $Fed_1^1 :< r_1^1 = 0, C_1^1 = wcet(Fed_1), D_1^1 = 10, P_1^1 = 20 >$;
- Un message $m_{1,2} :< r_{m_{1,2}} = 0, C_{m_{1,2}} = wctt(m_{1,2}), D_{m_{1,2}} = 20, P_{m_{1,2}} = 20 >$;
- Une tâche $Fed_2^1 :< r_2^1 = 0, C_2^1 = wcet(Fed_2), D_2^1 = 10, P_2^1 = 20 >$;
- Une tâche $Fed_2^2 :< r_1^2 = 10, C_1^2 = wcet(Fed_2), D_1^2 = 20, P_1^2 = 20 >$.

Deux choix sont aussi possibles (*C.f.* figure 6.11) pour effectuer la communication en accord avec les principes d'ordonnancement fonctionnel mis en œuvre dans la simulation.

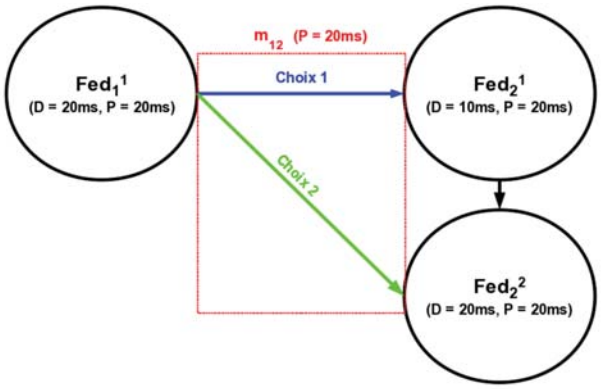


FIG. 6.11 – Solutions de discrétisation pour le cas 2

Cas 3 : La tâche émettrice **et** la tâche réceptrice ont une période plus petite que le message
- Une tâche émettrice $Fed_1 :< r_1 = 0, C_1 = wcet(Fed_1), D_1 = 10, P_1 = 10 >$;
- Un message $m_{1,2} :< r_{m_{1,2}} = 0, C_{m_{1,2}} = wctt(m_{1,2}), D_{m_{1,2}} = 20, P_{m_{1,2}} = 20 >$;
- Une tâche réceptrice $Fed_2 :< r_2 = 0, C_2 = wcet(Fed_2), D_2 = 10, P_2 = 10 >$.

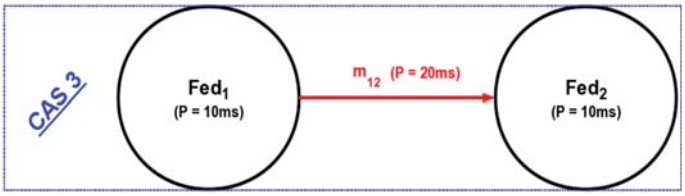


FIG. 6.12 – Illustration du cas 3

Dans ce cas, on discrétise la tâche émettrice Fed_1 ainsi que la tâche réceptrice Fed_2 et on obtient le système suivant :
- Une tâche $Fed_1^1 :< r_1^1 = 0, C_1^1 = wcet(Fed_1), D_1^1 = 10, P_1^1 = 20 >$;
- Une tâche $Fed_1^2 :< r_1^2 = 10, C_1^2 = wcet(Fed_1), D_1^2 = 20, P_1^2 = 20 >$;
- Un message $m_{1,2} :< r_{m_{1,2}} = 0, C_{m_{1,2}} = wctt(m_{1,2}), D_{m_{1,2}} = 20, P_{m_{1,2}} = 20 >$;
- Une tâche $Fed_2^1 :< r_2^1 = 0, C_2^1 = wcet(Fed_2), D_2^1 = 10, P_2^1 = 20 >$;
- Une tâche $Fed_2^2 :< r_2^2 = 10, C_2^2 = wcet(Fed_2), D_2^2 = 20, P_2^2 = 20 >$.

Quatre choix sont possibles pour effectuer la communication. Trois choix sont illustrés sur la figure 6.13 et le quatrième choix identique au choix 2 de la figure 6.9 est aussi possible.

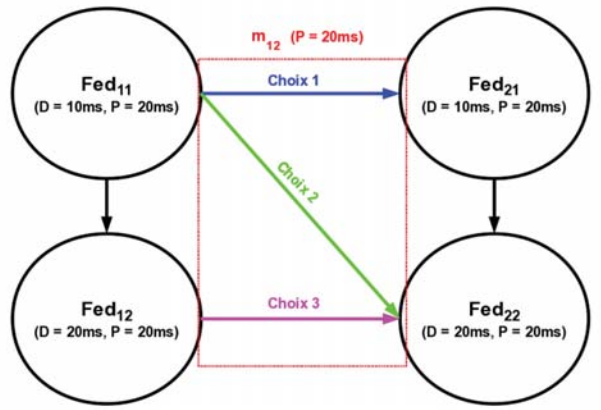


FIG. 6.13 – Solutions de discrétisation pour le cas 3

Cas 4 : La tâche émettrice **et** le message ont une période plus petite que tâche réceptrice (*C.f.* figure 6.14) :

- Une tâche émettrice $Fed_1 :< r_1 = 0, C_1 = wcet(Fed_1), D_1 = 10, P_1 = 10 >$;
- Un message $m_{1,2} :< r_{m_{1,2}} = 0, C_{m_{1,2}} = wctt(m_{1,2}), D_{m_{1,2}} = 20, P_{m_{1,2}} = 20 >$;
- Une tâche receptrice $Fed_2 :< r_2 = 0, C_2 = wcet(Fed_2), D_2 = 10, P_2 = 10 >$.

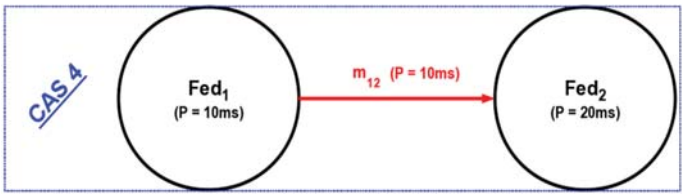


FIG. 6.14 – Illustration du cas 4

Ici la deux découpages sont possibles pour la discrétisation. Dans le premier cas (*C.f.* figure 6.15), on discrétise la tâche émettrice Fed_1 ainsi que la tâche réceptrice Fed_2 et chaque sous-tâche émettrice produira un message pour la sous-tâche réceptrice correspondante et on obtient le système :

- Une tâche $Fed_1^1 :< r_1^1 = 0, C_1^1 = wcet(Fed_1), D_1^1 = 10, P_1^1 = 20 >$;
- Une tâche $Fed_1^2 :< r_1^2 = 10, C_1^2 = wcet(Fed_1), D_1^2 = 20, P_1^2 = 20 >$;
- Un message $m_{1,2}^1 :< r_{m_{1,2}}^1 = 0, C_{m_{1,2}}^1 = wctt(m_{1,2}), D_{m_{1,2}}^1 = 10, P_{m_{1,2}}^1 = 20 >$;
- Un message $m_{1,2}^2 :< r_{m_{1,2}}^2 = 0, C_{m_{1,2}}^2 = wctt(m_{1,2}), D_{m_{1,2}}^2 = 20, P_{m_{1,2}}^2 = 20 >$;
- Une tâche $Fed_2^1 :< r_2^1 = 0, C_2^1 = wcet(Fed_2), D_2^1 = 10, P_2^1 = 20 >$;
- Une tâche $Fed_2^2 :< r_2^2 = 10, C_2^2 = wcet(Fed_2), D_2^2 = 20, P_2^2 = 20 >$.

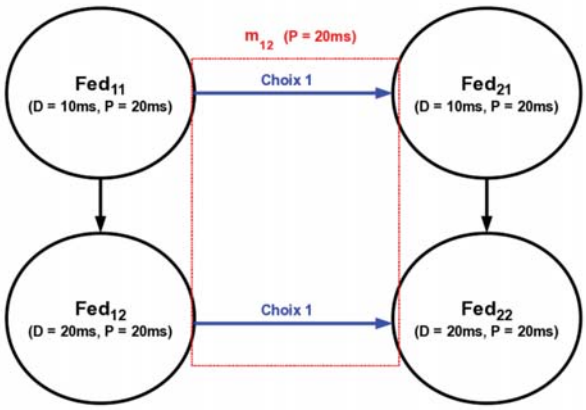


FIG. 6.15 – Première solution de discrétisation pour le cas 4

Dans le second cas (*C.f.* figure 6.16), on discrétise uniquement la tâche émettrice Fed_1 chaque sous-tâche émettrice produira un message pour l'unique sous-tâche réceptrice correspondante et on obtient le système :

- Une tâche $Fed_1^1 :< r_1^1 = 0, C_1^1 = wcet(Fed_1), D_1^1 = 10, P_1^1 = 20 >$;
- Une tâche $Fed_1^2 :< r_1^2 = 10, C_1^2 = wcet(Fed_1), D_1^2 = 20, P_1^2 = 20 >$;
- Un message $m_{1,2}^1 :< r_{m_{1,2}}^1 = 0, C_{m_{1,2}}^1 = wctt(m_{1,2}), D_{m_{1,2}}^1 = 10, P_{m_{1,2}}^1 = 20 >$;
- Un message $m_{1,2}^2 :< r_{m_{1,2}}^2 = 10, C_{m_{1,2}}^2 = wctt(m_{1,2}), D_{m_{1,2}}^2 = 20, P_{m_{1,2}}^2 = 20 >$;
- Une tâche $Fed_2^1 :< r_2 = 0, C_2 = wcet(Fed_2), D_2 = 20, P_2 = 20 >$.

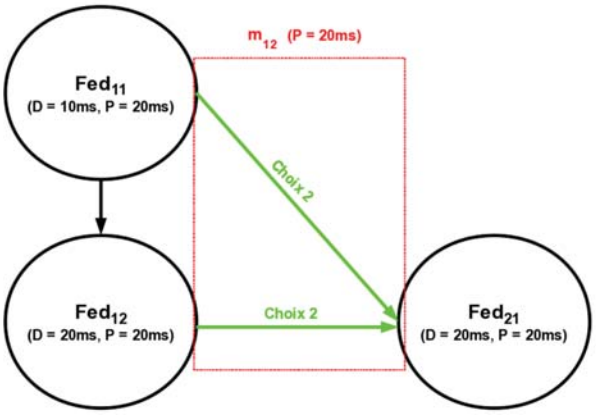


FIG. 6.16 – Seconde solution de discrétisation pour le cas 4

Cas 5 : Il y a une inter-dépendance : les tâches s'exécutent et communiquent dans les deux sens à la même période (*C.f.* figure 6.17) :

- Une tâche émettrice $Fed_1 :< r_1 = 0, C_1 = wcet(Fed_1), D_1 = 20, P_1 = 20 >$;
- Un message $m_{1,2} :< r_{m_{1,2}} = 0, C_{m_{1,2}} = wctt(m_{1,2}), D_{m_{1,2}} = 20, P_{m_{1,2}} = 20 >$;
- Une tâche réceptrice $Fed_2 :< r_2 = 0, C_2 = wcet(Fed_2), D_2 = 20, P_2 = 20 >$;
- Un message $m_{2,1} :< r_{m_{2,1}} = 0, C_{m_{2,1}} = wctt(m_{2,1}), D_{m_{2,1}} = 20, P_{m_{2,1}} = 20 >$.

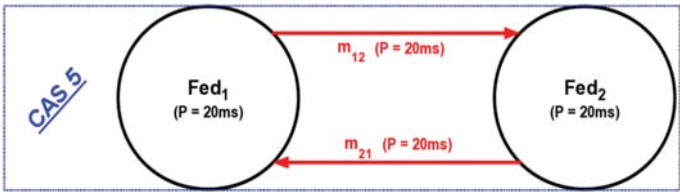


FIG. 6.17 – Illustration du cas 5

Ici, deux découpages sont possibles pour la discrétisation et reposent, principalement, sur les possibilités pour fractionner l'exécution du code des fédérés en plusieurs parties. Cela dépend des spécificités sémantiques des codes de calculs implémentés dans chaque fédéré ainsi que des principes mis en œuvre pour l'ordonnancement fonctionnel de l'application. Pour cet exemple, nous considérerons, que le code source des deux fédérés est divisible en deux parties avec des temps de calculs égaux à la moitié du temps de calcul de la tâche de base. Dans le premier cas (*C.f.* figure 6.15), on discrétise uniquement la tâche Fed_1. La première sous-tâche de Fed_1 produira un message pour l'unique sous-tâche de Fed_2 correspondante. La deuxième sous-tâche de Fed_1 recevra le message qui lui sera transmis par l'unique sous-tâche de Fed_2 correspondante. La sous-tâche de Fed_2 sera donc émettrice et réceptrice sur le même cycle. Au final, on obtient le système :

- Une tâche $Fed_1^1 :< r_1 = 0, C_1 = \frac{wcet(Fed_1)}{2}, D_1 = 10, P_1 = 20 >$;
- Une tâche $Fed_1^1 :< r_1 = 10, C_1 = \frac{wcet(Fed_1)}{2}, D_1 = 20, P_1 = 20 >$;
- Un message $m_{1,2} :< r_{m_{1,2}} = 0, C_{m_{1,2}} = wctt(m_{1,2}), D_{m_{1,2}} = 10, P_{m_{1,2}} = 20 >$;
- Une tâche $Fed_2^1 :< r_2 = 0, C_2 = wcet(Fed_2), D_2 = 20, P_2 = 20 >$;
- Un message $m_{2,1} :< r_{m_{2,1}} = 10, C_{m_{2,1}} = wctt(m_{2,1}), D_{m_{2,1}} = 20, P_{m_{2,1}} = 20 >$.

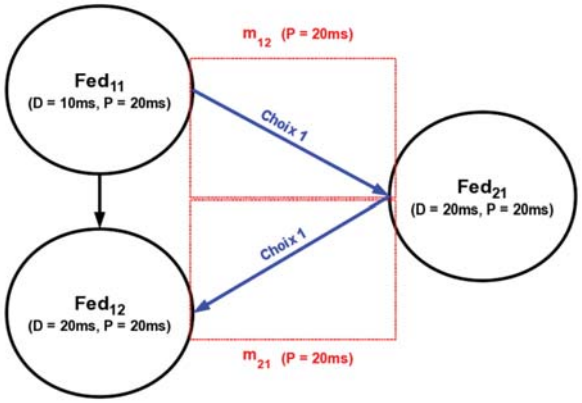


FIG. 6.18 – Première solution de discrétisation pour le cas 5

Dans le second cas (*C.f.* figure 6.16), on discrétise la tâche Fed_1 ainsi que la tâche Fed_2. La première sous tâche de Fed_1 envoie un message pour la première sous-tâche de Fed_2 et inversement pour les deuxièmes sous-tâches. Au final, on obtient le système :

- Une tâche $Fed_1^1 :< r_1^1 = 0, C_1^1 = \frac{wcet(Fed_1)}{2}, D_1^1 = 10, P_1^1 = 20 >$;
- Une tâche $Fed_1^2 :< r_1^2 = 10, C_1^2 = \frac{wcet(Fed_1)}{2}, D_1^2 = 20, P_1^2 = 20 >$;
- Un message $m_{1,2} :< r_{m_{1,2}} = 0, C_{m_{1,2}} = wctt(m_{1,2}), D_{m_{1,2}} = 10, P_{m_{1,2}}^1 = 20 >$;
- Une tâche $Fed_2^1 :< r_2 = 0, C_2 = \frac{wcet(Fed_2)}{2}, D_2 = 10, P_2 = 20 >$;
- Une tâche $Fed_2^2 :< r_2^2 = 10, C_2^2 = \frac{wcet(Fed_2)}{2}, D_2^2 = 20, P_2 = 20 >$;
- Un message $m_{2,1} :< r_{m_{2,1}} = 10, C_{m_{2,1}} = wctt(m_{2,1}), D_{m_{2,1}} = 20, P_{m_{2,1}} = 20 >$.

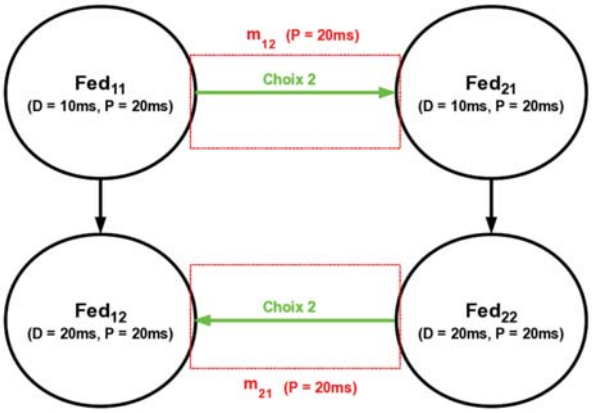


FIG. 6.19 – Seconde solution de discrétisation pour le cas 5

6.2.4 Modification de Tindell et Clark

La démarche initiale proposée par Tindell et Clark ne permet de prendre en compte que des tâches et des messages qui respectent l'hypothèse de l'instant critique $\forall i$ et $\forall j$, $r_i = r_{m_{i,j}} = 0$. Cela n'est pas adapté à notre méthode de discrétisation car nous modifions le date de réveil des tâches et des messages afin de conserver la propriété de périodicité des tâches initiales. Nous nous sommes inspirés de l'approche des mises à jours des dates de réveil développée dans la thèse de Samia Saad-Bouzefrane [SB98] en l'intégrant directement dans l'holistique de Tindell et Clark. Ainsi, les dates de réveil différentes de 0 (non respect de l'instant critique) peuvent être prises en compte dans les équations de Tindell et Clark. Nous modifions le système originale des équations récurrentes (*C.f.* équation 6.1, page 119) de la façon suivante :

$$\begin{cases} R_i^{(0)} &= C_i \\ J_i^{(0)} &= r_i \\ R_i^{(k)} &= ResponseTimes\left(J_i^{(k-1)}\right) \\ J_i^{(k)} &= \max\left(r_i, \max_{j \in prec(i)}\left(R_j^{(k)}\right)\right) \end{cases} \tag{6.2}$$

Cependant, dans [SB98], une fois que les dates de réveil sont mis à jour, chaque site est toujours ordonnancé par des algorithmes classiques tels que Rate Monotonic ou Deadline Monotonic. Cette approche n'est pas correcte car ces algorithmes classiques ne sont plus optimaux lorsque l'hypothèse de l'instant critique n'est pas respectée. Nous avons donc combiné notre approche en utilisant l'affectation de priorité selon l'algorithme de Audsley [Aud91] qui est le seul algorithme qui peut prendre en compte des tâches avec des dates de réveil différentes de 0.

Normalement, lorsque l'on considère un ensemble de tâches qui ne respectent pas l'instant critique : $\forall i$ et $\forall j$, $r_i = r_{m_{i,j}} \geq 0$, il faut aussi modifier l'intervalle d'étude considéré. Généralement, on considère l'intervalle d'étude proposé par Leung et Merrill [LM80] : $[0, \max_{\forall i}(r_i) + 2 * PPCM]$. Audsley propose, dans son article [Aud91], un autre intervalle d'étude possible pour ce type de tâches. Cet autre intervalle d'étude, bien que plus compliqué à calculer, est plus petit que le précédent d'un facteur 2. Toutefois, dans notre cas, les tâches initiales avant la discrétisation respectent l'instant critique, nous conserverons par conséquent l'intervalle d'étude $[0, PPCM]$ pour valider notre ensemble de sous-tâches après la discrétisation.

6.2.5 Application à un cas test

Nous allons maintenant appliquer ces différentes techniques de validation à la fédération CNES cas simple (*C.f.* partie 4.2.1, page 73) comprenant 4 fédérés. Cette fois-ci, nous considérerons que les fédérés 1 et 3 respectent un temps de calcul égal à 10% de leurs périodes respectives. Les fédérés 2 et 4 ont quant à eux des temps de calculs égaux à 20% de leurs périodes respectives. Le message le plus fréquent (entre les fédérés 3 et 2) met une milliseconde pour transiter par le CERTI (au pire des cas) et le message entre les fédérés 1 et 4 met quant à lui trois millisecondes pour être acheminé via le CERTI. Les autres messages mettent deux millisecondes pour transiter

entre les fédérés. Les différentes mesures (*C.f.* partie 5.2.1, page 101), nous montrent que cela est pessimiste par rapport aux expérimentations. En effet, le CERTI permet de tenir des latences de transfert dans des temps beaucoup plus petits. Au final, nous obtenons ainsi un système composé de 4 tâches et de 7 messages :

- $Fed_1 :< r_{Fed_1} = 0, C_{Fed_1} = 5, D_{Fed_1} = 50, P_{Fed_1} = 50 >$;
- $Fed_2 :< r_{Fed_2} = 0, C_{Fed_2} = 1, D_{Fed_2} = 10, P_{Fed_2} = 10 >$;
- $Fed_3 :< r_{Fed_3} = 0, C_{Fed_3} = 2, D_{Fed_3} = 10, P_{Fed_3} = 10 >$;
- $Fed_4 :< r_{Fed_4} = 0, C_{Fed_4} = 10, D_{Fed_4} = 50, P_{Fed_4} = 50 >$.
- $m_{1,2} : < r_{m_{1,2}} = 0, C_{m_{1,2}} = 2, D_{m_{1,2}} = 50, P_{m_{1,2}} = 50 >$;
- $m_{1,4} : < r_{m_{1,4}} = 0, C_{m_{1,4}} = 3, D_{m_{1,4}} = 50, P_{m_{1,4}} = 50 >$;
- $m_{2,1} : < r_{m_{2,1}} = 0, C_{m_{2,1}} = 2, D_{m_{2,1}} = 50, P_{m_{2,1}} = 50 >$;
- $m_{3,2} : < r_{m_{3,2}} = 0, C_{m_{3,2}} = 1, D_{m_{3,2}} = 10, P_{m_{3,2}} = 10 >$.
- $m_{3,4} : < r_{m_{3,4}} = 0, C_{m_{3,4}} = 2, D_{m_{3,4}} = 50, P_{m_{3,4}} = 50 >$;
- $m_{4,1} : < r_{m_{4,1}} = 0, C_{m_{4,1}} = 3, D_{m_{4,1}} = 50, P_{m_{4,1}} = 50 >$;
- $m_{4,3} : < r_{m_{4,3}} = 0, C_{m_{4,3}} = 2, D_{m_{4,3}} = 50, P_{m_{4,3}} = 50 >$;

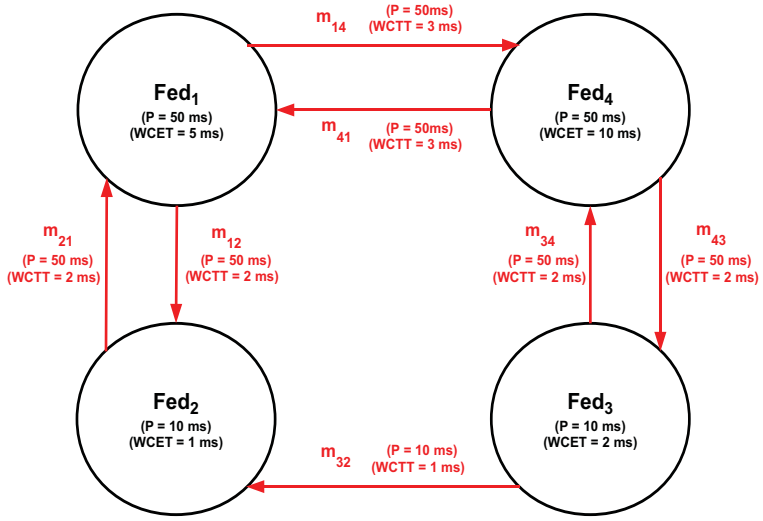


FIG. 6.20 – Graphe de tâche basique pour la fédération CNES

Nous pouvons discrétiser ce système original en utilisant les différentes techniques présentées dans la partie 6.2.3. La fédération CNES ne mettant pas en œuvre des algorithmes avec une sémantique spécifique, nous avons découpé arbitrairement l'application comme cela est décrit sur la figure 6.21. Le système est composé de 14 sous-tâches et de 10 sous-messages :

- $Fed_1^1 :< r_{Fed_1^1} = 0, C_{Fed_1^1} = 2, D_{Fed_1^1} = 25, P_{Fed_1^1} = 50 >$;
- $Fed_1^2 :< r_{Fed_1^2} = 25, C_{Fed_1^2} = 3, D_{Fed_1^2} = 50, P_{Fed_1^2} = 50 >$;
- $Fed_2^1 :< r_{Fed_2^1} = 0, C_{Fed_2^1} = 1, D_{Fed_2^1} = 10, P_{Fed_2^1} = 50 >$;
- $Fed_2^2 :< r_{Fed_2^2} = 10, C_{Fed_2^2} = 1, D_{Fed_2^2} = 20, P_{Fed_2^2} = 50 >$;
- $Fed_2^3 :< r_{Fed_2^3} = 20, C_{Fed_2^3} = 1, D_{Fed_2^3} = 30, P_{Fed_2^3} = 50 >$;
- $Fed_2^4 :< r_{Fed_2^4} = 30, C_{Fed_2^4} = 1, D_{Fed_2^4} = 40, P_{Fed_2^4} = 50 >$;
- $Fed_2^5 :< r_{Fed_2^5} = 40, C_{Fed_2^5} = 1, D_{Fed_2^5} = 50, P_{Fed_2^5} = 50 >$;
- $Fed_3^1 :< r_{Fed_3^1} = 0, C_{Fed_3^1} = 2, D_{Fed_3^1} = 10, P_{Fed_3^1} = 50 >$;

- $Fed_3^2 :< r_{Fed_3^2} = 10, C_{Fed_3^2} = 2, D_{Fed_3^2} = 20, P_{Fed_3^2} = 50 >$;
- $Fed_3^3 :< r_{Fed_3^3} = 20, C_{Fed_3^3} = 2, D_{Fed_3^3} = 30, P_{Fed_3^3} = 50 >$;
- $Fed_3^4 :< r_{Fed_3^4} = 30, C_{Fed_3^4} = 2, D_{Fed_3^4} = 40, P_{Fed_3^4} = 50 >$;
- $Fed_3^5 :< r_{Fed_3^5} = 40, C_{Fed_3^5} = 2, D_{Fed_3^5} = 50, P_{Fed_3^5} = 50 >$;
- $Fed_4^1 :< r_{Fed_4^1} = 0, C_{Fed_4^1} = 5, D_{Fed_4^1} = 25, P_{Fed_4^1} = 50 >$;
- $Fed_4^2 :< r_{Fed_4^1} = 25, C_{Fed_4^2} = 5, D_{Fed_4^2} = 50, P_{Fed_4^2} = 50 >$.
- $m_{1,2}^1 : < r_{m_{1,2}^1} = 0, C_{m_{1,2}^1} = 2, D_{m_{1,2}^1} = 50, P_{m_{1,2}^1} = 50 >$;
- $m_{1,4}^1 : < r_{m_{1,4}^1} = 0, C_{m_{1,4}^1} = 3, D_{m_{1,4}^1} = 25, P_{m_{1,4}^1} = 50 >$;
- $m_{2,1}^1 : < r_{m_{2,1}^1} = 0, C_{m_{2,1}^1} = 2, D_{m_{2,1}^1} = 50, P_{m_{2,1}^1} = 50 >$;
- $m_{3,2}^1 : < r_{m_{3,2}^1} = 0, C_{m_{3,2}^1} = 1, D_{m_{3,2}^1} = 10, P_{m_{3,2}^1} = 50 >$;
- $m_{3,2}^2 : < r_{m_{3,2}^2} = 10, C_{m_{3,2}^2} = 1, D_{m_{3,2}^2} = 20, P_{m_{3,2}^2} = 50 >$;
- $m_{3,2}^3 : < r_{m_{3,2}^3} = 20, C_{m_{3,2}^3} = 1, D_{m_{3,2}^3} = 30, P_{m_{3,2}^3} = 50 >$;
- $m_{3,2}^4 : < r_{m_{3,2}^4} = 30, C_{m_{3,2}^4} = 1, D_{m_{3,2}^4} = 40, P_{m_{3,2}^4} = 50 >$;
- $m_{3,2}^5 : < r_{m_{3,2}^5} = 40, C_{m_{3,2}^5} = 1, D_{m_{3,2}^5} = 50, P_{m_{3,2}^4} = 50 >$;
- $m_{3,4}^1 : < r_{m_{3,4}^1} = 0, C_{m_{3,4}^1} = 2, D_{m_{3,4}^1} = 50, P_{m_{3,4}^1} = 50 >$;
- $m_{4,1}^1 : < r_{m_{1,4}^1} = 25, C_{m_{1,4}^1} = 3, D_{m_{1,4}^1} = 50, P_{m_{1,4}^1} = 50 >$.

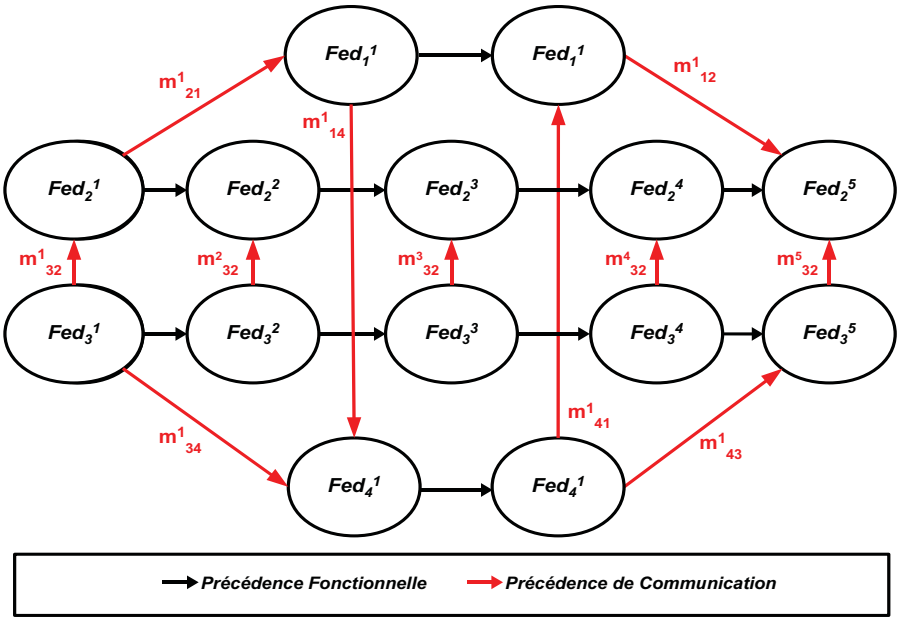


FIG. 6.21 – Graphe de tâches de l'application CNES après la discrétisation

A ce stade, nous avons un modèle exploitable pour valider formellement notre application. Nous respectons l'affinité décrite dans le chapitre précédent (*C.f.* figure 5.13). Les fédérés 1 et 2 sont situés sur le processeur `CPU0` et les fédérés 3 et 4 sont sur le processeur `CPU1`. Nous affectons les priorités pour les fédérés sur chacun de ces deux processeurs en utilisant l'algorithme de Audsley [Aud91]. De plus, nous assurons que les précédences simples sont prises en compte selon la méthode de Blazewich [Bla76] selon laquelle une tâche donnée (les sous-tâches dans notre cas) a une priorité supérieure à une tâche qui lui succède sur le même processeur. Ainsi nous obtenons la distribution de priorité illustrée sur la figure 6.22. Rappelons que, sur les systèmes Linux, il y a 100 niveaux de priorité dédiés aux applications temps réel. Nous pourrions donc envisager des applications avec un ensemble plus important de tâches sur les processeurs.

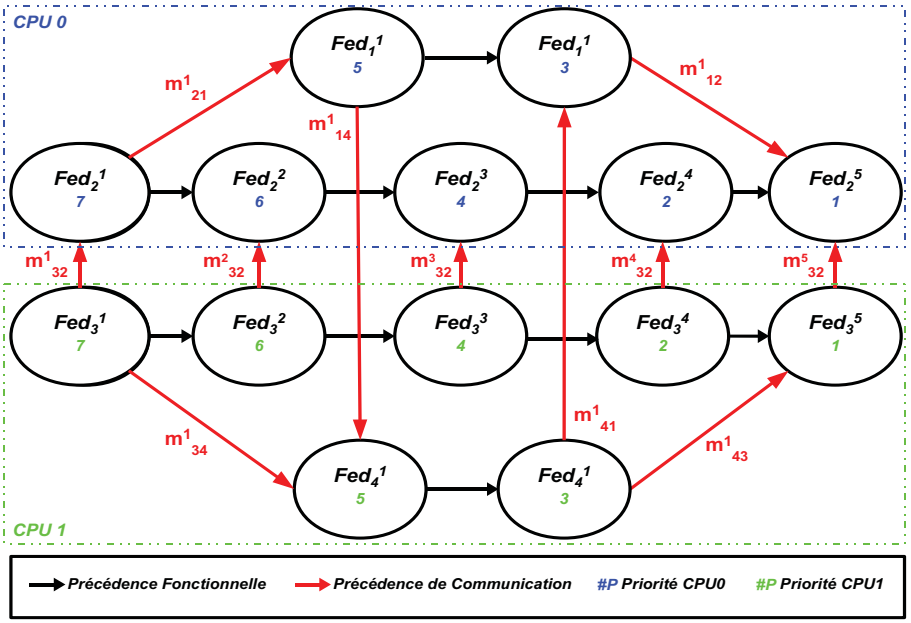


FIG. 6.22 – Graphe de tâches de l'application CNES avec les priorités

En se basant sur les exécutions de cette ensemble de tâches selon cette configuration de l'affinité et de la priorité pour chaque processus, nous pouvons calculer les gigues et les pires temps de réponses pour chacune des tâches du système en appliquant les équations modifiées de Tindell et Clark (*C.f.* équation 6.2, page 126).

- $Fed_1^1 :< r_{Fed_1^1} = 0, C_{Fed_1^1} = 5, J_{Fed_1^1} = 6, R_{Fed_1^1} = 11, D_{Fed_1^1} = 25, P_{Fed_1^1} = 50 >$;
- $Fed_1^2 :< r_{Fed_1^2} = 25, C_{Fed_1^2} = 5, J_{Fed_1^2} = 31, R_{Fed_1^2} = 36, D_{Fed_1^2} = 50, P_{Fed_1^2} = 50 >$;
- $Fed_2^1 :< r_{Fed_2^1} = 0, C_{Fed_2^1} = 1, J_{Fed_2^1} = 3, R_{Fed_2^1} = 4, D_{Fed_2^1} = 10, P_{Fed_2^1} = 50 >$;
- $Fed_2^2 :< r_{Fed_2^2} = 10, C_{Fed_2^2} = 1, J_{Fed_2^2} = 13, R_{Fed_2^2} = 14, D_{Fed_2^2} = 20, P_{Fed_2^2} = 50 >$;
- $Fed_2^3 :< r_{Fed_2^3} = 20, C_{Fed_2^3} = 1, J_{Fed_2^3} = 23, R_{Fed_2^3} = 24, D_{Fed_2^3} = 30, P_{Fed_2^3} = 50 >$;
- $Fed_2^4 :< r_{Fed_2^4} = 30, C_{Fed_2^4} = 1, J_{Fed_2^4} = 33, R_{Fed_2^4} = 34, D_{Fed_2^4} = 40, P_{Fed_2^4} = 50 >$;
- $Fed_2^5 :< r_{Fed_2^5} = 40, C_{Fed_2^5} = 1, J_{Fed_2^5} = 43, R_{Fed_2^5} = 44, D_{Fed_2^5} = 50, P_{Fed_2^5} = 50 >$;
- $Fed_3^1 :< r_{Fed_3^1} = 0, C_{Fed_3^1} = 2, J_{Fed_3^1} = 0, R_{Fed_3^1} = 2, D_{Fed_3^1} = 10, P_{Fed_3^1} = 50 >$;
- $Fed_3^2 :< r_{Fed_3^2} = 10, C_{Fed_3^2} = 2, J_{Fed_3^2} = 10, R_{Fed_3^2} = 12, D_{Fed_3^2} = 20, P_{Fed_3^2} = 50 >$;
- $Fed_3^3 :< r_{Fed_3^3} = 20, C_{Fed_3^3} = 2, J_{Fed_3^3} = 20, R_{Fed_3^3} = 22, D_{Fed_3^3} = 30, P_{Fed_3^3} = 50 >$;
- $Fed_3^4 :< r_{Fed_3^4} = 30, C_{Fed_3^4} = 2, J_{Fed_3^4} = 30, R_{Fed_3^4} = 32, D_{Fed_3^4} = 40, P_{Fed_3^4} = 50 >$;
- $Fed_3^5 :< r_{Fed_3^5} = 40, C_{Fed_3^5} = 2, J_{Fed_3^5} = 40, R_{Fed_3^5} = 42, D_{Fed_3^5} = 50, P_{Fed_3^5} = 50 >$;
- $Fed_4^1 :< r_{Fed_4^1} = 0, C_{Fed_4^1} = 2, J_{Fed_4^1} = 14, R_{Fed_4^1} = 16, D_{Fed_4^1} = 25, P_{Fed_4^1} = 50 >$;
- $Fed_4^2 :< r_{Fed_4^2} = 25, C_{Fed_4^2} = 3, J_{Fed_4^2} = 25, R_{Fed_4^2} = 28, D_{Fed_4^2} = 50, P_{Fed_4^2} = 50 >$.
- $m_{1,2}^1 : < r_{m_{1,2}^1} = 0, C_{m_{1,2}^1} = 2, J_{m_{1,2}^1} = 36, R_{m_{1,2}^1} = 38, D_{m_{1,2}^1} = 50, P_{m_{1,2}^1} = 50 >$;
- $m_{1,4}^1 : < r_{m_{1,4}^1} = 0, C_{m_{1,4}^1} = 3, J_{m_{1,4}^1} = 11, R_{m_{1,4}^1} = 14, D_{m_{1,4}^1} = 50, P_{m_{1,4}^1} = 50 >$;
- $m_{2,1}^1 : < r_{m_{2,1}^1} = 0, C_{m_{2,1}^1} = 2, J_{m_{2,1}^1} = 4, R_{m_{2,1}^1} = 6, D_{m_{2,1}^1} = 50, P_{m_{2,1}^1} = 50 >$;
- $m_{3,4}^1 : < r_{m_{3,4}^1} = 0, C_{m_{3,4}^1} = 2, J_{m_{3,4}^1} = 2, R_{m_{3,4}^1} = 4, D_{m_{3,4}^1} = 50, P_{m_{3,4}^1} = 50 >$;
- $m_{3,2}^1 : < r_{m_{3,2}^1} = 0, C_{m_{3,2}^1} = 1, J_{m_{3,2}^1} = 2, R_{m_{3,2}^1} = 3, D_{m_{3,2}^1} = 10, P_{m_{3,2}^1} = 50 >$;
- $m_{3,2}^2 : < r_{m_{3,2}^2} = 10, C_{m_{3,2}^2} = 1, J_{m_{3,2}^2} = 12, R_{m_{3,2}^2} = 13, D_{m_{3,2}^2} = 20, P_{m_{3,2}^2} = 50 >$;
- $m_{3,2}^3 : < r_{m_{3,2}^3} = 20, C_{m_{3,2}^3} = 1, J_{m_{3,2}^3} = 22, R_{m_{3,2}^3} = 23, D_{m_{3,2}^3} = 30, P_{m_{3,2}^3} = 50 >$;
- $m_{3,2}^4 : < r_{m_{3,2}^4} = 30, C_{m_{3,2}^4} = 1, J_{m_{3,2}^4} = 32, R_{m_{3,2}^4} = 33, D_{m_{3,2}^4} = 40, P_{m_{3,2}^4} = 50 >$;

- $m^5_{3,2} : < r_{m^5_{3,2}} = 40, C_{m^5_{3,2}} = 1,\ J_{m^5_{3,2}} = 42,\ R_{m^5_{3,2}} = 43, D_{m^5_{3,2}} = 50, P_{m^4_{3,2}} = 50 >$;
- $m^1_{4,1} : < r_{m^1_{1,4}} = 25, C_{m^1_{1,4}} = 3,\ J_{m^1_{1,4}} = 28,\ R_{m^1_{1,4}} = 31, D_{m^1_{1,4}} = 50, P_{m^1_{1,4}} = 50 >$.

Nous avons prouvé que les temps de réponse pour chaque tâche et chaque message sont inférieurs aux délais correspondants : pour les tâches $\forall\ i,\ R_i < D_i$ et pour les messages $\forall\ i, j$, $R_{i,j} < D_{i,j}$. Par conséquent, le comportement temporel de la fédération CNES cas simple utilisant le mode d'exécution *Data Flow* est validé formellement.

6.3 Étude sur les fédérations *Time Stepped*

6.3.1 Hypothèse et modèle

Le mode d'exécution *Time Stepped* (*C.f.* partie 4.3.4, page 83) ne possède pas d'horloge globale commune à tous les fédérés. Le fédéré attend l'acceptation de l'avancée dans le temps acquittée par la RTI (callback `timeAdvanceGrant()`). Lorsque le fédéré recevra cette callback, il est assuré de pouvoir réaliser ses algorithmes locaux car il aura reçu toutes les données correspondantes à son cycle en cours. De plus, cet acquittement assurera au fédéré d'être synchronisé avec les autres fédérés dans la progression commune dans le temps simulé. Chaque fédéré peut vérifier localement la concordance entre son temps local et l'expression de sa cyclicité dans le temps simulé. Ainsi, en utilisant des timers locaux, nous pouvons programmer la date à laquelle le fédéré réclame cette avance dans le temps par un appel au service `timeAdvanceRequest(`ts`)`. De la même façon que pour le mode *Data Flow*, nous utilisons le modèle des tâches pour modéliser le comportement des fédérés *Time Stepped* et nous considérons qu'un fédéré noté Fed_i est une tâche périodique défini par le quadruplet $< r_i, C_i, D_i, P_i >$ tel que :

1. r_i moment de l'activation initiale de la tâche ;
2. C_i la durée d'exécution pire cas de la tâche ;
3. D_i la deadline, la durée limite à ne pas dépasser ;
4. P_i la période d'activation de la tâche.

L'utilisation des mécanismes de gestion du temps rend la prise en compte des précédences entre les tâches difficile à modéliser en utilisant ou en dérivant les techniques de discrétisation présentées précédemment pour le mode d'exécution *Data Flow*. En revanche, l'utilisation des mécanismes de gestion du temps (et donc de l'algorithme du Null Message dans le CERTI) permettent d'assurer que le fédéré aura reçu toutes les données disponibles (par les callbacks `reflectAttributeValues()`) lorsqu'il recevra son acquittement d'avancée dans le temps par la RTI. Par conséquent, en respectant notre hypothèse de séparation physique des processus d'exécution et de communication, on peut prendre en compte les contraintes de précédence entre les fédérés communicants directement dans le modèle de la tâche sans utiliser un modèle additionnel de message. La particularité ici est donc de prendre en compte cette précédence dans le coût d'exécution de la tâche (*C.f.* figure 6.23).

Ainsi, la principale difficulté revient à déterminer ce temps C_{Fed_i} pour le fédéré qui est composé de deux parties : $C_{Fed_i} = C_{Fed_i}^{reception} + C_{Fed_i}^{execution}$. Plus particulièrement, il faut réussir à estimer le temps nécessaire pour réaliser sa phase de réception et de synchronisation utilisant les mécanismes de gestion du temps. Cette durée dépend principalement de l'algorithme de synchronisation distribué (*C.f.* partie 2.2.3, page 15) implémenté dans la RTI.

6.3.2 Proposition de métriques pour quantifier les messages

6.3.2.1 Hypothèses et notations

Toujours dans un souci de déterminisme, nous voulons assurer la prédictibilité des mécanismes de gestion du temps mis en œuvre dans le CERTI. Par opposition à l'algorithme de Mattern qui fournit des méthodes non déterministes sur des plateformes asynchrones, nous avons choisi de poursuivre nos travaux sur l'utilisation de l'algorithme original CMB. Nous avons donc proposé un certain nombre de métriques afin d'aider à la détermination du temps $C_{Fed_i}^{reception}$ nécessaire à la réception et à la synchronisation des fédérés *Time Stepped*. Ce travail a fait l'objet d'une com-

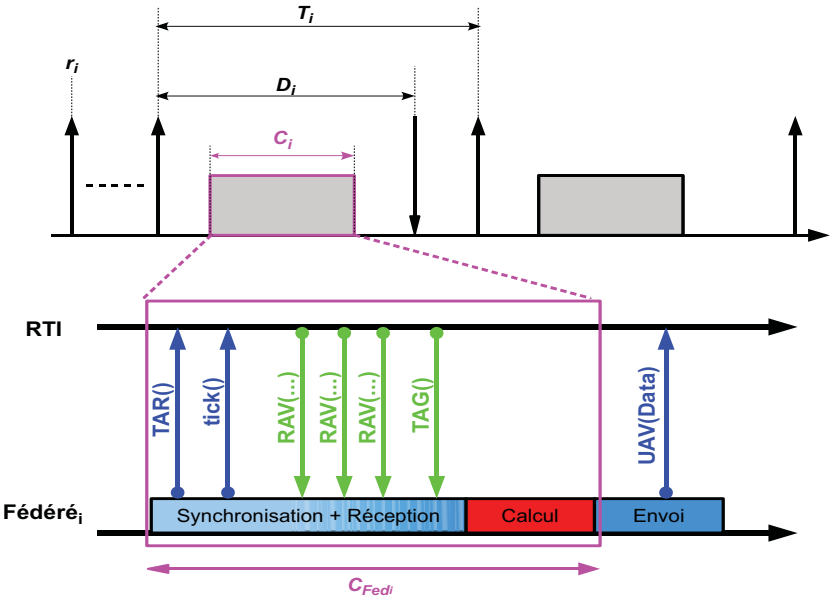


FIG. 6.23 – Correspondance entre modèle de tâche et mode d'exécution

munication dans une conférence internationale [CSN11]. Pour décrire les différentes métriques, nous avons besoin de faire quelques hypothèses de base :

1. La simulation globale (la fédération HLA) est composée de N fédérés *Time Stepped* périodiques indexés par un entier $i \in [1, ..., N]$;
2. Pour chacun de ces fédérés Fed_i :
   - son temps logique est noté t_i ;
   - son pas de temps, ou encore l'expression de sa périodicité de calcul dans le temps simulé, est noté ts_i ;
   - son pas de communication, ou encore l'expression de sa périodicité de la communication dans le temps simulé, est noté cs_i ;
   - son lookahead est noté lk_i ;
3. $PS(i, j)$ est une matrice Publicateurs/Souscripteurs (de taille $N*N$) qui permet d'identifier les différentes communications entre les fédérés ;
4. $PPCM_{ts}$ est la valeur du plus petit commun multiple (PPCM) de tous les pas de temps des fédérés ;
5. L'intervalle d'étude, exprimé en temps simulé, est généralement égal à $PPCM_{ts}$.

6.3.2.2 Métriques statiques

Le nombre de messages NULL envoyés par un fédéré Fed_i sur un intervalle d'étude $PPCM_{ts}$ de temps simulé est noté NM_i^e et se calcule de la façon suivante $\forall\ i \in [1, ..., N]$:

$$NM_i^e = \frac{PPCM_{ts}}{ts_i} \tag{6.3}$$

Réciproquement, le nombre de messages NULL reçus par un fédéré Fed_i sur un intervalle d'étude $PPCM_{ts}$ de temps simulé est noté NM_i^r et se calcule de la façon suivante $\forall\ i, j\ \in\ [1, ..., N]$ avec $j \neq i$:

$$NM_i^r \quad = \quad \sum_j \frac{PPCM_{ts}}{ts_j} \tag{6.4}$$

Nous avons aussi proposé des métriques pour quantifier l'échange des messages de simulation (messages contenant de vraies données). Le nombre de messages de simulation envoyés par un fédéré Fed_i sur un intervalle d'étude $PPCM_{ts}$ est noté $SimM_i^e$ et se calcule de la façon suivante $\forall\ i\ \in\ [1, ..., N]$:

$$SimM_i^e \quad = \quad \frac{PPCM_{ts}}{cs_i} \tag{6.5}$$

Réciproquement, le nombre de messages de simulation reçus par un fédéré Fed_i sur un intervalle d'étude $PPCM_{ts}$ est noté $SimM_i^r$ et se calcule de la façon suivante $\forall\ i, j\ \in\ [1, ..., N]$ avec $j \neq i$:

$$SimM_i^r \quad = \quad \sum_j \left(PS(i,j) * \frac{PPCM_{ts}}{cs_j} \right) \tag{6.6}$$

6.3.2.3 Métrique dynamique

Les métriques présentées dans la partie précédente permettent de déterminer et de quantifier l'échange de messages (NULL et de simulation) avant de faire fonctionner la fédération. Cependant, il peut être utile pour un fédéré de savoir combien de messages NULL il lui reste à recevoir avant l'acquittement explicite de son avancée dans le temps par la RTI (*C.f.* figure 6.24).

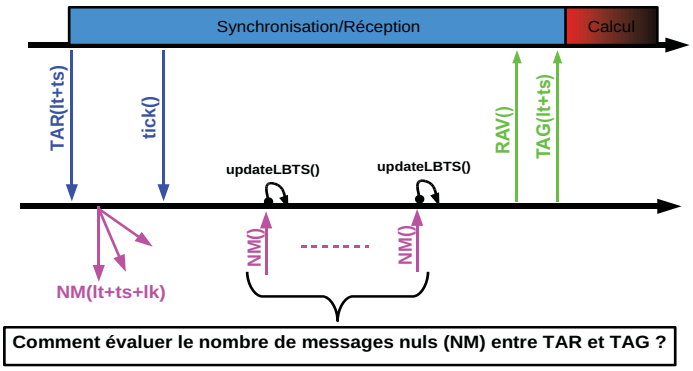


FIG. 6.24 – Utilité d'une métrique dynamique

Afin d'appliquer cette métrique, nous avons besoin d'ajouter quelques hypothèses supplémentaires aux hypothèses précédentes pour tous les fédérés :

1. Chaque fédéré Fed_i doit localement maintenir un vecteur statique des pas de temps de chaque autre fédéré de la simulation. Cela permet de connaitre les cycles de tous les autres fédérés en jeu dans la fédération. Ce vecteur est noté :

$$\mathbf{TS_i} = \begin{pmatrix} ts_1 \\ \vdots \\ ts_N \end{pmatrix}$$

2. Chaque fédéré Fed_i doit localement maintenir un vecteur dynamique GT_i qui sert de vision de l'état global d'avancée dans le temps simulé de l'ensemble de la fédération pour le fédéré i. Chaque gt_j représente l'état d'avancée dans le temps par le fédéré j et il est égal au temps local du fédéré j plus son pas de temps et son lokkahead : $gt_j = lt_j + ts_j + lk_j$. Ce vecteur est mis à jour pour chaque fédéré par l'échange des messages NULL et la valeur du LBTS[13] (*C.f.* partie 2.5.3, 34) est calculée à partir de ce vecteur.

$$\mathbf{GT_i} = \begin{pmatrix} gt_1 \\ \vdots \\ gt_N \end{pmatrix}$$

Avec ces hypothèses additionnelles, chaque fédéré i pourra calculer à n'importe quel moment de la simulation le nombre de messages NULL, noté NM_i^{cycles}, qu'il lui reste à recevoir pour son cycle entre l'appel au service `timeAdvanceRequest()` et son acquittement pour avancer à son cycle suivant par la réception de la callback `timeAdvanceGrant()`. L'algorithme CMB est un algorithme distribué asynchrone et ainsi quand un des fédérés noté Fed_i calcule le nombre de messages NULL qu'il doit recevoir, les autres fédérés pourraient éventuellement avoir déjà demandé leur avance de temps pour le prochain cycle. L'algorithme CMB appliqué à des fédérés *Time Stepped* garantit que tous les fédérés ne peuvent pas avoir plus d'un cycle de différence dans leurs avancées respectives dans le temps simulé. Ainsi, nous pouvons assurer des bornes minimale et maximale pour le calcul du paramètre NM_i^{cycles} par les formules suivantes[14] $\forall\, i, j \in [1, ..., N]$ avec $j \neq i$:

$$W_j = \left\lceil \frac{t_i + ts_i - gt_j}{ts_j} \right\rceil \tag{6.7}$$

$$\sum_j W_j \leq NM_i^{cycles} \leq \sum_j W_j + (N-1) \tag{6.8}$$

[13] *Lower Bound on Time Stamp*
[14] *Rappelons que* $\lceil x \rceil$ *désigner le plus grand entier plus grand ou égal à* x

6.3.2.4 Validation

Les algorithmes distribués asynchrones sont assez complexes à étudier et à mettre en œuvre. Par conséquent, toujours dans une démarche de validation formelle, nous nous sommes intéressés au formalisme des automates finis et temporisé [HU01] et nous avons travaillé avec le logiciel UPPAAL [BDL04] [76]. En effet, une exécution de simulation HLA peut avoir une modélisation discrète caractérisée par un état donné à un instant donné, et leurs comportements au fil du temps peuvent être décrits par une séquence de transition d'état. Nous avons donc créé deux modèles UPPAAL : un pour décrire les fédérés *Time Stepped* et un autre pour décrire le fonctionnement de la RTI fournissant des services de gestion du temps avec l'algorithme CMB. Une illustration du modèle UPPAAL du fédéré *Time Stepped* est donnée sur la figure 6.25, nous voyons donc qu'il peut être décrit comme une suite d'états et de transitions entre ces états selon certaines conditions.

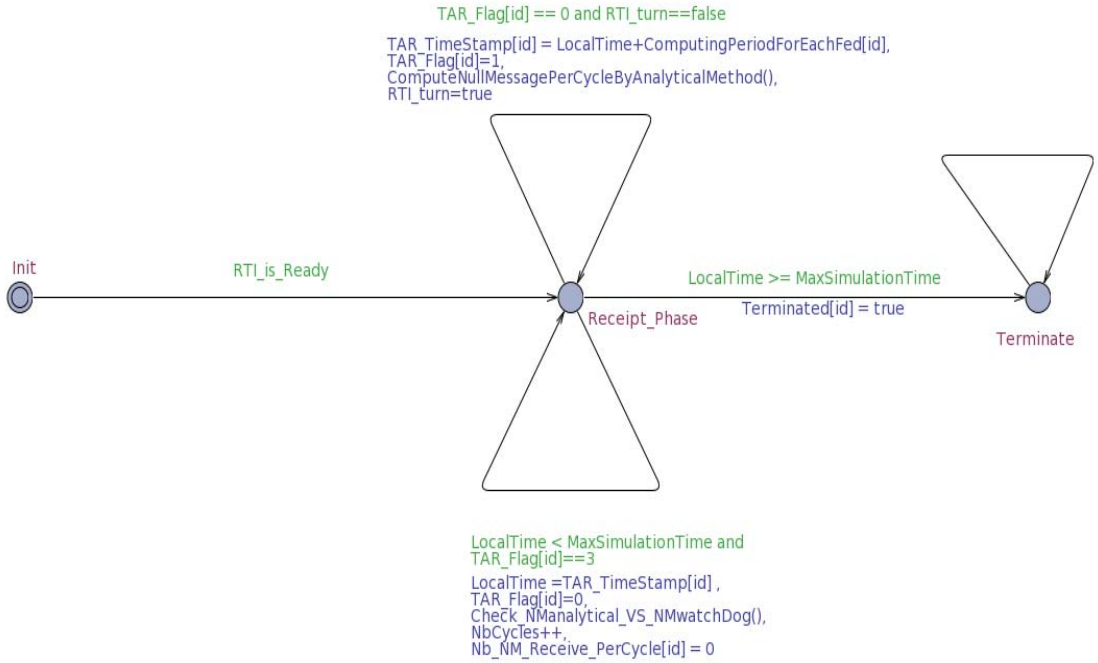


FIG. 6.25 – Modèle UPPAAL d'un fédéré *Time Stepped*

L'outil de validation par model-checking intégré à UPPAAL nous a permis de vérifier formellement les différents métriques. La validité des métriques statiques a été démontrée quelque soit le nombre de fédérés. En revanche, l'espace de recherche pour la validation de la métrique dynamique est beaucoup plus important. Nous avons donc réussi à la valider pour 2, 3 et 4 fédérés périodiques *Time Stepped*. Cependant, dès que le nombre de fédérés mis en œuvre dans la fédération dépasse 4, nous avons une explosion combinatoire de l'espace de recherche pour le model-checking. La validité de l'intervalle de la métrique dynamique n'a pas pu être validé pour plus de 4 fédérés. Actuellement nous investiguons de nouvelles méthodes de validation pour vérifier cette métrique distribuée sur une plus grande échelle.

6.3.3 Validation et application au cas tests

Reprenons le cas test de la fédération CNES cas simple, nous avons donc un ensemble de quatre tâches :

- $Fed_1 :< r_{Fed_1} = 0, C_{Fed_1} = C_{Fed_1}^{reception} + C_{Fed_1}^{execution}, D_{Fed_1} = 50, P_{Fed_1} = 50 >;$
- $Fed_2 :< r_{Fed_2} = 0, C_{Fed_2} = C_{Fed_1}^{reception} + C_{Fed_1}^{execution}, D_{Fed_2} = 10, P_{Fed_2} = 10 >;$
- $Fed_3 :< r_{Fed_3} = 0, C_{Fed_3} = C_{Fed_3}^{reception} + C_{Fed_3}^{execution}, D_{Fed_3} = 10, P_{Fed_3} = 10 >;$
- $Fed_4 :< r_{Fed_4} = 0, C_{Fed_4} = C_{Fed_4}^{reception} + C_{Fed_4}^{execution}, D_{Fed_4} = 50, P_{Fed_4} = 50 >.$

Nous conservons les hypothèses émises pour la fédération suivant le mode d'exécution *Data Flow*. Les fédérés 1 et 3 respectent un temps de calcul égal à 10% de leurs périodes respectives et les fédérés 2 et 4 ont des temps de calculs égaux à 20% de leurs périodes respectives. Le système s'écrit alors :

- $Fed_1 :< r_{Fed_1} = 0, C_{Fed_1} = C_{Fed_1}^{reception} + 5, D_{Fed_1} = 50, P_{Fed_1} = 50 >;$
- $Fed_2 :< r_{Fed_2} = 0, C_{Fed_2} = C_{Fed_1}^{reception} + 1, D_{Fed_2} = 10, P_{Fed_2} = 10 >;$
- $Fed_3 :< r_{Fed_3} = 0, C_{Fed_3} = C_{Fed_3}^{reception} + 2, D_{Fed_3} = 10, P_{Fed_3} = 10 >;$
- $Fed_4 :< r_{Fed_4} = 0, C_{Fed_4} = C_{Fed_4}^{reception} + 10, D_{Fed_4} = 50, P_{Fed_4} = 50 >.$

De la même façon que pour la fédération *Data Flow*, ces quatre tâches sont réparties sur deux processeurs de la plate-forme PRISE. Les fédérés Fed_1 et Fed_3 sont sur `CPU0` et les fédérés Fed_2 et Fed_4 sont sur `CPU1`. Les précédences de communication et de synchronisation étant pris en compte dans le coût de chaque tâche, nous pouvons faire une affectation des priorités selon l'algorithme classique Rate Monotonic [LL73] selon lequel les tâches avec les périodes les plus petites sur un processeur sont les plus prioritaires. Ainsi, sur le `CPU0`, Fed_3 est plus prioritaire que Fed_1 et sur `CPU1` c'est Fed_2 qui est plus prioritaire que Fed_4. La formule classique de validation de l'algorithme Rate Monotonic est basée sur le calcul du taux d'utilisation du processeur donné, pour un ensemble de tâches notées par un entier i, par la formule :

$$U = \sum_{i=0}^{n} \frac{C_i}{P_i} \tag{6.9}$$

Ainsi la condition nécessaire et suffisante pour vérifier qu'un ensemble de tâches est ordonnançable sur un processeur peut être décrite comme ceci :

$$U \leq n \times (2^{1/n} - 1) \tag{6.10}$$

Dans notre cas, nous pouvons appliquer ce test d'ordonnancabilité, on calcule d'abord le taux d'utilisation pour le processeurs `CPU0` et `CPU1` :

$$U_{CPU0} = \frac{C_{Fed_1}^{reception}+5}{50} + \frac{C_{Fed_3}^{reception}+2}{10} = \frac{C_{Fed_1}^{reception}}{50} + \frac{C_{Fed_3}^{reception}}{10} + \frac{3}{10}$$

$$U_{CPU1} = \frac{C_{Fed_4}^{reception}+10}{50} + \frac{C_{Fed_2}^{reception}+1}{10} = \frac{C_{Fed_4}^{reception}}{50} + \frac{C_{Fed_2}^{reception}}{10} + \frac{3}{10}$$

Et nous pouvons ensuite appliquer le test d'ordonnancabilité pour chaque processeur

$$U_{CPU0} \leq 2 \times (2^{1/2} - 1) \Longleftrightarrow \frac{C_{Fed_1}^{reception}}{50} + \frac{C_{Fed_3}^{reception}}{10} + \frac{3}{10} \leq 0.83 \Longleftrightarrow \frac{C_{Fed_1}^{reception}}{50} + \frac{C_{Fed_3}^{reception}}{10} \leq 0.53$$

$$U_{CPU1} \leq 2 \times (2^{1/2} - 1) \Longleftrightarrow \frac{C_{Fed_4}^{reception}}{50} + \frac{C_{Fed_2}^{reception}}{10} + \frac{3}{10} \leq 0.83 \Longleftrightarrow \frac{C_{Fed_4}^{reception}}{50} + \frac{C_{Fed_2}^{reception}}{10} \leq 0.53$$

Pour ce cas test, pour le processeur `CPU0`, les temps $C_{Fed_1}^{reception}$ et $C_{Fed_3}^{reception}$ doivent vérifier que $\frac{C_{Fed_1}^{reception}}{50} + \frac{C_{Fed_3}^{reception}}{10} \leq 0.53$ et pour le processeur `CPU0`, les temps $C_{Fed_4}^{reception}$ et $C_{Fed_2}^{reception}$ doivent vérifier que $\frac{C_{Fed_4}^{reception}}{50} + \frac{C_{Fed_2}^{reception}}{10} \leq 0.53$. Notre objectif est donc de déterminer les paramètres $C_{Fed_i}^{reception}$ de façon à pouvoir assurer l'ordonnancabilité du système. Nous pensons que les différentes métriques présentées précédemment sont indispensables pour le calcul de ces temps de réception et de synchronisation. La méthodologie étant posée, les travaux futurs devront permettre de déterminer exactement le coût temporel de la synchronisation d'un fédéré *Time Stepped* en fonction de ces métriques. La figure 6.26 illustre le fonctionnement de la métrique de calcul des messages NULL reçus par le fédéré Fed_4 durant l'intervalle d'étude (intervalle de temps simulé) :

$$NM_{Fed_4}^{r} = \frac{50}{50} + \frac{50}{10} + \frac{50}{10} = 11$$

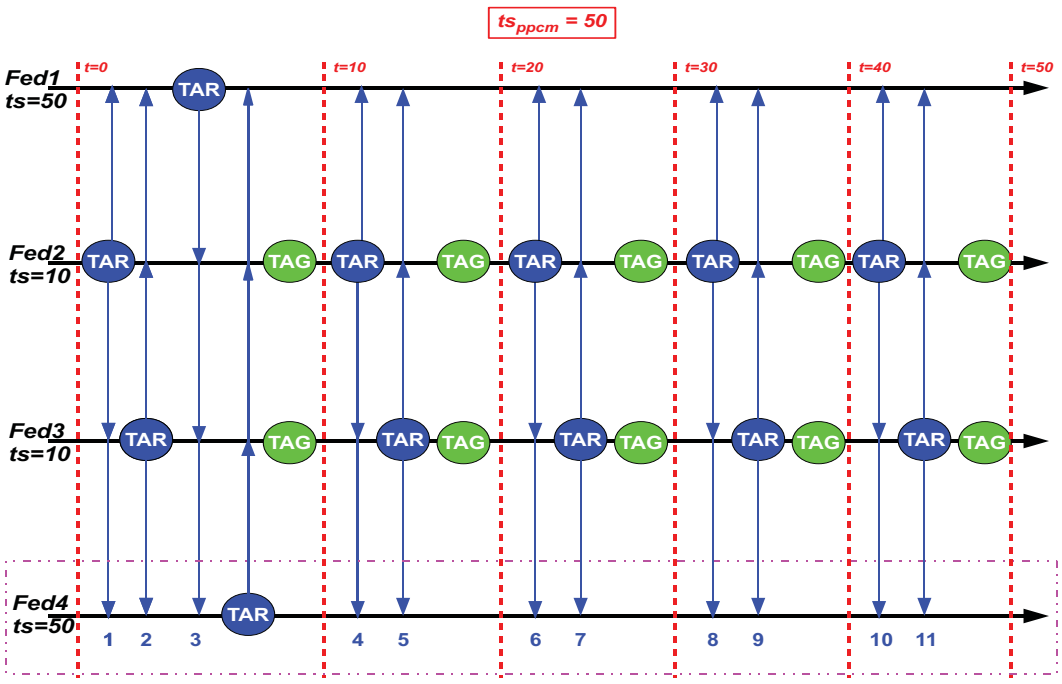


FIG. 6.26 – Exemple de quantification de l'échange de messages NULL

6.4 Etude sur les fédérations *Event Driven*

6.4.1 Comparaison avec le mode d'exécution time-stepped

Ce mode d'exécution est aussi basé sur les principes des mécanismes de gestion du temps qui permettent d'assurer une synchronisation sur l'échelle du temps logique (ou temps simulé). Le fédéré qui suit ce mode d'exécution reçoit et publie de l'information et il est régulateur et contraint selon les principes de la norme HLA. Pour ce mode de fonctionnement, les fédérés demandent de recevoir le prochain message de simulation disponible entre son temps logique et le temps ts (futur) et sûr du point de vue de la causalité. Pour cela, il appelle le service `nextEventRequest(`ts`)`. En retour, la RTI lui délivrera le premier message disponible et lui accordera une avancée dans le temps à l'estampille t_1 de ce message par la callback `timeAdvanceGrant(`t_1`)` (avec $t_1 \leq ts$) (*C.f.* partie 4.3.5, page 85).

La principale différence avec le mode d'exécution *Time Stepped* est qu'il est difficile de garder une cohérence entre le temps simulé et le temps réel. Le fédéré avancera dans le temps simulé à la prochaine date sûre du point de vue de la causalité. Tandis que pour le mode *Time Stepped*, le fédéré demande explicitement une avancée dans le temps simulé à des dates correspondantes à l'expression de sa cyclicité dans le temps simulé. Les fédérés *Event Driven* sont souvent mis en œuvre dans les simulations de théâtre de guerre plutôt que dans des simulations de systèmes embarqués. Cependant pour nos expérimentations, nous utilisons les principes de ce mode fonctionnement lorsque nous travaillons avec le mode d'exécution mixte (*C.f.* partie 4.3.6, page 86) mis en œuvre pour la résolution des équations différentielles dans les expérimentations du simulateur de vol (*C.f.* partie 5.4, page 105).

6.4.2 Le nouvel algorithme NULL Message PRIME

Lorsque l'on utilise l'algorithme original CMB pour synchroniser des fédérés selon le mode d'exécution *Event Driven*, on peut être confronté au problème du *time creep* (*C.f.* partie 4.3.5, page 85). Ce problème peut amener, certaines fois, à une baisse importante des performances de la simulation. Pour solutionner ce problème, nous avons implémenté et mis en œuvre un nouvel algorithme noté CMB ONERA (algorithme des messages NULL PRIME) ; ce travail est aussi intégré dans le papier de communication [CSN11]. L'idée de ce nouvel algorithme est de profiter de l'architecture particulière du CERTI et particulièrement du processus central RTIG. Dans l'algorithme classique CMB, le RTIG agit uniquement en tant qu'une pure passerelle et distribue simplement les messages à chaque fédéré impliqué dans l'avancée du temps simulé (régulateur et contraint). Il ne connait pas l'estampille temporelle du message ni si un fédéré est dans une boucle de demande d'avancée dans le temps simulé. Dans l'algorithme CMB ONERA, le RTIG est impliqué dans cette avancée du temps simulé : il est au courant de l'état d'un fédéré et particulièrement s'il est dans une boucle d'avancée dans le temps (appel au service `nextEventRequest()` ou encore `timeAdvanceRequest()`). Dans ce cas, le RTIG recueille les informations temporelles relatives à chaque fédéré et peut participer à l'avancée dans le temps en lançant des opérations de réduction.

Ce nouvel algorithme est simple : lorsqu'un fédéré appelle le service `nextEventRequest()`, son RTIA va envoyer un message NULL PRIME au RTIG et celui ci va pouvoir ensuite calculer la valeur globale du LBTS[15]. Chaque fois que la valeur du LBTS calculée par le RTIG augmente, le RTIG génère, lui-même, un message NULL anonyme qui est diffusé à tous les fédérés contraints. Quand un fédéré (en fait son RTIA) reçoit un message NULL anonyme, il pourra réaliser le calcul

[15] *Lower Bound on Time Stamp*

de sa valeur locale du LBTS. Nous pouvons reprendre le cas du *time creep* présenté précédemment (*C.f.* figure 4.14, page 86), dans le cas de l'application de l'algorithme CMB ONERA, la séquence de message correspondante est donnée dans la figure 6.27.

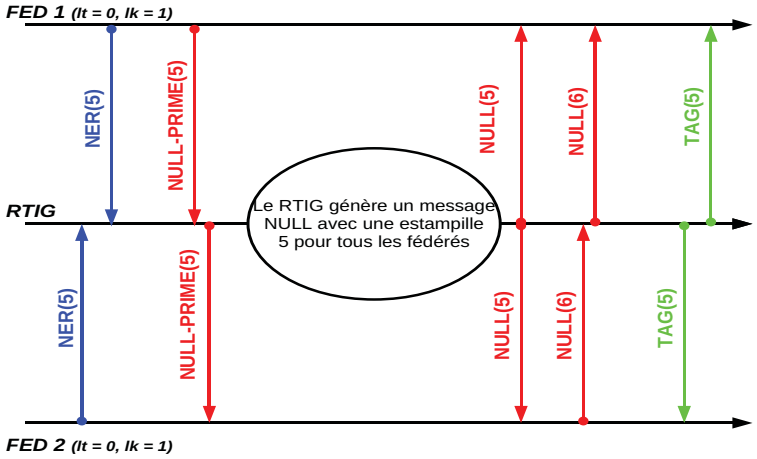


FIG. 6.27 – Fonctionnement de l'algorithme des messages NULL PRIME avec deux fédérés

Le nombre de messages additionnels échangés avant l'acquittement de l'avancée dans le temps (messages NULL et NULL PRIME) passe de 12 à 6. Le nombre de messages additionnels utilisés par l'algorithme CMB ONERA est indépendante de la valeur du lookahead alors que l'algorithme CMB classique nécessite un nombre de messages additionnels qui est proportionnel au lookahead et aux temps demandés dans les avances des fédérés dans le temps simulé. L'algorithme CMB ONERA peut co-exister avec l'algorithme classique dans le CERTI et ne génère que de nouveaux messages anonymes uniquement lorsque le RTIG a recueilli assez d'information pour lancer une opération de réduction. Le nouvel algorithme a été intégré dans la version actuelle du CERTI et il peut être activé à l'aide d'une option de compilation dédiée. Les expérimentations sur le simulateur de vol ont montré que lorsque cet algorithme est utilisé, les calculs supplémentaires réalisés par le RTIG pour la réduction peuvent nuire au respect des cycles très petits pour les fédérés situés sur le même nœud que le RTIG. Cependant, nous pensons que ce nouvel algorithme est plus performant sur des simulations distribuées de plus grande échelle par rapport à l'algorithme CMB classique.

6.4.3 Vers une démarche de validation formelle

Nous pensons que notre algorithme CMB ONERA peut être vu comme un équivalent des algorithmes de réduction d'un état global comme celui de Mattern [Mat89, Mat93]. En effet, ces algorithmes sont basés sur la notion d'état des fédérés qui doivent rendre compte de leurs différents états locaux à un processus de contrôle qui est le seul capable de connaître l'état global de l'ensemble de la simulation. C'est aussi le cas dans notre algorithme dans lequel le processus RTIG est au courant de l'avancement de chaque fédéré et peut ainsi lancer sa réduction pour déterminer un état global. De plus, nous pensons que notre algorithme propose plusieurs avantages :

1. L'opération de réduction se déclenche automatiquement dès que quelque chose vaut la peine de la faire et il ne faut pas déterminer l'instant opportun de commencer cette réduction.

2. L'indeterminsime de l'algorithme de Mattern dû aux messages transients sur le réseau au moment du calcul du LBTS n'est pas un problème dans notre cas. En effet, le LBTS calculé à partir des messages NULL anonymes est toujours valide quels que soient les messages de simulation en transfert sur le réseau au moment du calcul de l'état global.
3. Le nombre de messages additionnels générés par l'algorithme est constant et indépendant de la valeur du lookahead.

De la même façon que pour les fédérés *Time Stepped*, nous proposons d'utiliser l'outil UPPAAL pour modéliser et vérifier les différentes propriétés de ce nouvel algorithme. L'objectif final est de déterminer des preuves formelles des différents avantages et d'assurer des calculs de temps de réception et de synchronisation déterministes pour les fédérés *Event Driven*. Nous pourrions ensuite introduire ces résultats dans notre démarche de validation du mode mixte en les combinant avec l'approche envisagée pour les fédérés *Time Stepped*.

Chapitre 7

Conclusions et Perspectives

7.1 Conclusions

7.1.1 Rappel des contributions

Cette thèse pour une "*Architecture de Simulation distribuée temps réel*" s'inscrit au carrefour de la simulation distribuée, des systèmes temps réel et des intergiciels (respectant des normes pour la distribution). Nous avons effectué une étude complète pour la mise en œuvre de simulations distribuées temps réel modélisant des systèmes embarqués. Pour cela, nous nous sommes basés sur les principes et les spécifications de la norme HLA et de l'intergiciel CERTI : la RTI open-source de l'ONERA. Sur ces bases, nous avons envisagé, étudié et expérimenté différentes pistes et ces travaux ont amené à diverses contributions :

- Premièrement, nous nous sommes efforcés de fournir une bibliographie solide et relativement complète des différents domaines relatifs à une architecture de simulation distribuée temps réel. Nous possédons une version électronique ou papier de la majorité des documents cités dans ce manuscrit que nous pouvons distribuer ou prêter selon les licences relatives à chaque document.
- Ensuite, nous avons défini un ensemble de modes d'exécution génériques basés sur les différentes techniques de synchronisation utilisées. Le mode *Data Flow* est un mode simple qui repose sur l'hypothèse d'une horloge globale commune. Le mode synchronisé par envoi d'une interaction est un mode semblable au mode *Data Flow* dans lequel les fédérés se réfèrent aux pulsations envoyées par un fédéré unique qui joue le rôle d'horloge globale. Lorsque la RTI offre les services de gestion du temps, il est possible d'utiliser les modes d'exécution *Time Stepped* et *Event Driven*. Ces mécanismes particuliers à la norme HLA permettent ainsi de synchroniser les fédérés en leur offrant une échelle de temps logique commune : le temps simulé. Une découverte a été la particularité de l'utilisation des mécanismes de gestion du temps pour interconnecter des fédérés assurant la résolution d'équations différentielles. Cela a amené à l'ajout d'un mode d'exécution mixte qui permet de conserver une cohérence entre le temps simulé et le temps réel.
- Nous avons spécifié, implémenté et évalué des cas tests pour des simulations de systèmes embarqués. Deux simulations sont relatives au vol en formation de satellites selon une problématique issue d'une collaboration avec le laboratoire du CNES. Nous avons aussi conçu notre propre simulateur de vol sous la forme d'une agrégation de fédérés HLA (*C.f.* partie 7.1.3, page 143). Ces simulateurs utilisent les modes d'exécution génériques définis

dans cette thèse. Les expérimentations nous ont permis de vérifier la tenue de contraintes temporelles très petites pour assurer des simulations avec des calculs et des communications haute fréquence.

- Même si la version actuelle du CERTI présentait des performances acceptables, nous avons ajouté un ensemble de fonctionnalités qui offrent un fonctionnement optimisé et compatible avec la démarche proposée pour la conception et l'exécution de simulations temps réel.
- Nous avons aussi proposé une méthodologie complète pour l'étude, la conception et la mise en œuvre de simulations HLA temps réel (*C.f.* partie 7.1.2).
- Enfin, nous avons étudié un ensemble de techniques pour l'aide à la validation formelle du comportement temporel d'une simulation HLA. Ces techniques dépendent des modes d'exécution choisis et nous avons fourni une étude approfondie des mécanismes de gestion du temps mis en œuvre dans le CERTI (*C.f.* partie 7.1.4).
- Certaines parties de ces travaux ont fait l'objet de communications dans des conférences internationales [CSA10, CSN11, CSSA11, CSAN11] ainsi que la soumission d'un article de journal [CSSA12].

7.1.2 Démarche méthodologique

Nous avons abordé ce travail de thèse avec une démarche méthodologique précise. Cette démarche peut être adoptée pour la conception et la mise en œuvre de tout type de simulation HLA temps réel ainsi que pour l'élaboration de n'importe quel type de système temps réel utilisant un intergiciel pour interconnecter ses composants :

1. **Prendre en compte le matériel :** Ici, l'important est de prendre en compte l'infrastructure matérielle qui va permettre les exécutions et les communications du système distribué pour réaliser un calcul global. Il faut appréhender quels sont les types de processeurs supportant les exécutions et au point de vue du réseau il faut déterminer la technologie utilisée, la topologie du réseau ainsi que la répartition géographique du système distribué (réseau local LAN ou réseau de plus grande échelle type WAN). Tous ces paramètres ont un impact sur les contraintes temporelles qu'il sera possible d'assurer et de vérifier pour l'application que l'on souhaite mettre en œuvre. A ce niveau, on peut aussi vérifier s'il existe une référence temporelle matérielle globale (comme la technologie RCIM dans le cas de la plate-forme PRISE) et quelle est sa précision.
2. **Évaluer les logiciels :** Au dessus de l'infrastructure matérielle, c'est la partie logicielle de bas niveau qui va jouer un rôle prédominant. Il faut, à ce niveau, appréhender les techniques de conception de l'application (les langages d'implémentation) et déterminer si les systèmes d'exploitation des nœuds sont temps réel et s'ils supportent des priorités différentes pour les tâches (processus ou thread). En ce qui concerne les communications, le protocole MAC joue un rôle déterminant et il faut aussi étudier les couches supérieures (Réseau et Transport) qui s'occupent du transport effectif des données. On peut aussi vérifier s'il existe un support logiciel pour assurer une référence de temps globale (par exemple, utilisation du protocole NTP) et quelle est la précision de cette horloge logicielle globale.
3. **Expertiser l'intergiciel et la norme :** L'utilisation d'un intergiciel (et de la norme sous-jacente) présente plusieurs avantages. Il permet aux concepteurs d'applications de s'affranchir de la prise en compte de la complexité des niveaux logiciel et matériel. Pour cela, la norme définit un ensemble de services qui seront fournis par l'intergiciel à l'application. La

sémantique associée à chaque service permet aux applications interconnectées de bénéficier des différents algorithmes distribués. Cependant, pour la conception d'une application temps réel, ce niveau d'abstraction ne permet pas aux concepteurs de garantir le déterminisme et la prédictibilité de leurs applications. Dans ce travail, nous avons démontré que la norme HLA (et les services qu'elle fournit) permet de mettre en œuvre des exécutions et des communications entre des simulateurs de systèmes embarqués. Nos expérimentations ont permis de vérifier que notre implémentation de cette norme (le CERTI) permettait d'assurer l'exécution et l'interconnexion de simulateurs avec des contraintes temporelles très petites. Nous avons aussi démontré que les services de gestion du temps spécifiques à la norme HLA fournissait une excellente alternative pour synchroniser des processus temps réel lorsqu'il n'existe pas d'horloge globale de référence (logicielle ou matérielle).

4. **Définir les contraintes temporelles de l'application :** La structure et les objectifs de l'application vont définir les contraintes auxquelles elle sera soumise. Par exemple, les applications multimédia sont soumises à des contraintes temps réel souples et on peut être tolérant sur les éventuelles erreurs temporelles. Inversement, les systèmes temps réel durs, doivent toujours vérifier leurs contraintes temporelles. Ces systèmes peuvent être soumis à des contraintes plus ou moins exigeantes. Par exemple, les environnements virtuels mettant en œuvre des opérateurs humains, sont soumis à des contraintes temporelles, de l'ordre de la dizaine de millisecondes, dépendantes des limites physiologiques de l'être humain. Les systèmes embarqués, quant à eux, sont soumis à des contraintes beaucoup plus restrictives pouvant descendre au niveau de la microseconde. Toujours au niveau de l'application, il faut aussi *choisir un mode d'exécution* dépendant des ressources disponibles et fournies par les niveaux précédents (les services de l'intergiciel, l'existence d'une horloge globale ou encore les mécanismes de synchronisation possibles). Dans cette thèse, nous avons identifié un ensemble de modes d'exécution génériques et applicables avec tout simulateur (fédéré) conforme aux spécifications de la norme HLA.
5. **Intégrer une validation formelle :** L'objectif est de fournir des outils pour permettre la vérification du comportement temporel et de la fiabilité de l'application sous-jacente. Les méthodes de validation possibles dépendent de toutes les caractéristiques évaluées aux niveaux précédents. Il faut prendre soin d'avoir un modèle dont la granularité permet de décrire de façon cohérente les mécanismes mis en œuvre dans l'application. Généralement, pour les systèmes temps réel, on s'oriente vers la théorie de l'ordonnancement ainsi que des techniques de validation formelles additionnelles telles que le *model-checking*. En ce sens, dans cette thèse, nous avons fourni des pistes de validation formelle par des techniques issues de la théorie de l'ordonnancement pour les fédérés en fonction des modes d'exécution utilisés. Nous avons aussi étudié en détail les mécanismes de gestion du temps implémentés dans le CERTI (basés sur l'algorithme CMB), nous avons proposé des métriques de quantification, un algorithme additionnel et envisagé une validation par l'outil de *model-checking* : UPPAAL [BDL04] [76].

7.1.3 Le simulateur de vol

Une contribution importante de cette thèse a été la conception, l'implémentation et l'expérimentation d'un simulateur de vol en collaboration avec David Saussié [Sau10]. Il se présente sous la forme d'une agrégation de fédérés HLA où chaque fédéré simule une partie spécifique et essentielle au vol d'un avion. Par conséquent, il est entièrement modulable et extensible et complètement compatible avec les spécifications de la norme HLA. De plus, nous possédons les spécificités techniques de chaque partie qui le compose aussi bien du point de vue des algorithmes

de calcul des fédérés, de la gestion des communications (le CERTI) ainsi que des techniques de déploiement sur une plate-forme.

La version actuelle de ce simulateur est opérationnelle sur la plate-forme PRISE. Les premières expérimentations présentées dans ce manuscrit ont montré que les techniques de conception et de mise en œuvre utilisées permettaient d'assurer un comportement temps réel. Une contribution intéressante sur ce simulateur a été la résolution d'équations différentielles en utilisant les mécanismes de gestion du temps pour la synchronisation. Le fait que nous possédions la sémantique des codes a permis de mettre en exergue l'importance d'un mode mixte comprenant deux phases de réception sur le même cycle de calcul (*C.f.* partie 4.3.6, page 86). En effet, certains fédérés ont besoin de la nouvelle entrée pour calculer leur état et la sortie lors du même pas de calcul. Si l'on souhaite conserver une concordance entre la période réelle du fédéré (en millisecondes) et son expression dans le temps simulé, chaque cycle de calcul du fédéré doit être composé de deux phases de réception. Ainsi, cela permet au fédéré de recevoir les données relatives à ses entrées et publier ses sorties pour le même pas de temps. Ce mode de fonctionnement a été mis en avant par l'implémentation de méthodes de résolution des équations différentielles. Une description détaillée des spécificités de la résolution de ces équations différentielles est disponible dans le rapport technique de ce simulateur [SCS11].

De plus, pour normaliser les différentes communications mises en œuvre dans ce simulateur, nous avons développé un modèle objet OMT dédié. Nous souhaitons encourager l'utilisation de ce modèle objet pour normaliser les communications essentielles dans tout type de simulateur de vol. Les éventuelles extensions du simulateur et l'ajout de fonctionnalités devraient permettre de compléter ce modèle objet et nous espérons que'il deviendra ainsi une référence dans le domaine des simulateurs de vol.

7.1.4 Utilisation de méthodes formelles et études des mécanismes de gestion du temps

Dans ce travail de thèse, nous nous sommes aussi concentrés sur la définition et la mise en œuvre d'un certain nombre de principes qui permettent la validation du comportement temporel d'une simulation distribuée. Ces techniques de validation ont été envisagées en fonction des modes d'exécution choisis pour les fédérés temps réel. Afin d'être compatible avec les méthodes de la théorie de l'ordonnancement, nous avons ajouté certaines fonctionnalités au CERTI (*C.f.* partie 5.1, page 89) afin de permettre l'affectation des différents processus sur des processeurs particuliers et aussi de pouvoir affecter une priorité différente à chaque processus. De plus, dans les cas de fédérés ayant une référence de temps globale (mode *Data Flow*), nous avons proposé une méthode de discrétisation, une extension de l'holistique de Tindell et Clark [TC94] associée à une méthode d'affectation des priorités par l'algorithme de Audsley [Aud91]. Ces techniques peuvent être adaptées et utilisées pour valider n'importe quel système distribué temps réel.

Nous nous sommes aussi particulièrement intéressés aux mécanismes de gestion du temps spécifiques de la norme HLA. De tels mécanismes permettent de synchroniser un ensemble de processus (les fédérés) lorsqu'il n'existe pas d'horloge globale. Ces services assurent le respect de la cohérence entre les cycles de calculs et les communications en se basant sur une échelle commune de temps simulé. De plus, les modes d'exécution que nous avons choisis permettent de garder une cohérence entre le temps réel et l'expression de ce temps réel dans le temps simulé. L'intergiciel CERTI se base sur l'algorithme CMB[1] [Bry77, CM79], assurant cette synchronisation à l'aide

[1] *Chandy-Misra-Bryant*

de messages additionnels contenant uniquement une estampille temporelle du temps simulé. Nos expérimentations ont montré que ce mécanisme de synchronisation permettait de synchroniser efficacement des simulateurs soumis à des contraintes temporelles très faibles. Nous avons proposé un ensemble de métriques pour pouvoir quantifier le nombre de messages additionnels générés par ce mécanisme de synchronisation lors de l'utilisation de fédérés *Time Stepped*. De plus, nous avons proposé un nouvel algorithme résolvant le problème du *time creep* (saturation du réseau en messages additionnels) dans le cas des fédérés *Event Driven*. Enfin, le mode d'exécution mixte, utilisant les principes de ces deux modes de fonctionnement dans le même cycle de calcul, permet de conserver une cohérence entre le temps réel et le temps simulé pour la résolution d'équations différentielles dans une simulation distribuée.

7.2 Perspectives

7.2.1 Les évolutions du CERTI et les extensions de la norme HLA

Le CERTI est l'intergiciel fournissant les services distribués aux applications et assurant la communication entre toutes ces applications interconnectées. Des évolutions de ce logiciel libre sont permises et plusieurs pistes peuvent être envisagées, implémentées et testées :

- Cette thèse repose sur l'hypothèse que notre réseau est surdimensionné et qu'il est par conséquent assez performant pour assurer des communications hautes fréquences dans le contexte du temps réel (déterminisme de ces communications). Une piste majeure à considérer serait d'intégrer au CERTI des supports logiciels pour prendre en compte des réseaux et des protocoles MAC déterministes (*C.f.* partie annexe B.2.2, page 165).
- Remplacer l'algorithme CMB classique du CERTI par l'algorithme de Mattern [Mat93]. Nous pourrions ainsi mesurer les différences de performances par rapport à notre algorithme CMB, quantifier les sources d'indéterminisme pour l'utilisation de cet algorithme dans le contexte temps réel et enfin comparer explicitement cet algorithme avec notre variante de l'algorithme CMB classique.
- Intégrer les mécanismes optimistes de gestion du temps (*C.f.* partie 2.2.3.2, page 16 et partie 2.5.2, page 33) dans le CERTI. Cela permettrait de fournir des services supplémentaires à l'intergiciel et évaluer leur utilisation dans le contexte des simulations temps réel mais cette dernière piste est encore très incertaine.

Le RTIG est le processus central du CERTI qui agit en tant que passerelle afin d'acheminer les messages de la simulation aux fédérés concernés. Néanmoins, dans le contexte du temps réel, son placement sur l'infrastructure distribuée peut avoir un impact non négligeable sur le déterminisme de l'application. Par exemple, reprenons une des mises en œuvre de la simulation CNES complexe décrite dans le document [NdS08]. Lors d'une expérimentation sur une architecture distribuée à deux nœuds, les fédérés avec les cycles les plus petits (les fédérés 5 et 6) occupaient un nœud dédié qui était isolé du processus central RTIG se trouvant sur l'autre nœud. Ainsi, le schéma de communication le plus simple et idéal aurait amené les fédérés 5 et 6 à communiquer directement par la carte réseau de leur nœud (*C.f.* Figure 7.1). Mais, dans le CERTI toutes les communications passent par le RTIG et donc dans ce cas, le message transitait deux fois sur le réseau pour être acheminé (*C.f.* Figure 7.2). Ainsi, sur des réseaux mal dimensionnés, le placement du processus RTIG peut être crucial pour assurer des communications hautes fréquences.

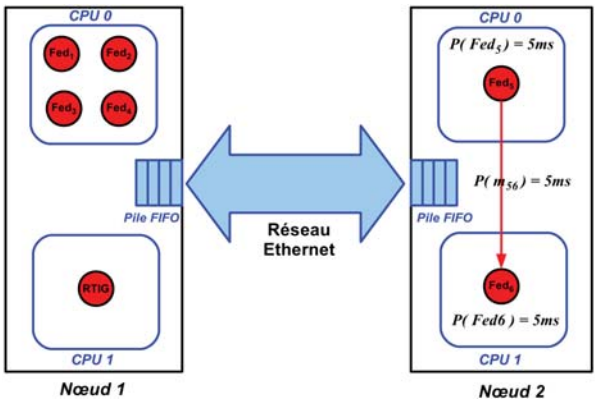


FIG. 7.1 – Communication idéale

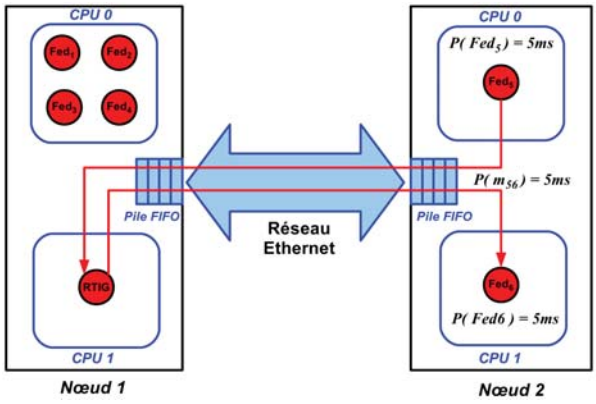


FIG. 7.2 – Communication réelle

Pour pallier à cela et toujours dans un souci de déterminisme, nous pourrions envisager, dans un premier temps, d'augmenter la réactivité et le déterminisme du processus central RTIG en le reprogrammant selon le paradigme multi-thread. Cela permettrait de prendre en compte des priorités plus ou moins grandes pour les différents messages transitants. Ainsi nous pourrions accorder plus d'importance à certains messages de plus grande fréquence ou d'importance cruciale pour les applications. Une autre technique à envisager consisterait à s'affranchir complètement du processus RTIG. Cela serait possible pour assurer des services simples comme la publication et l'abonnement mais les algorithmes plus complexes tels que ceux la gestion du temps seraient plus compliqués à exporter sur une architecture de RTI complètement distribuée. L'algorithme CMB classique serait utilisable et exportable sur une architecture distribuée. Cependant la variante CMB ONERA ou l'algorithme de Mattern, nécessitant idéalement la présence d'un processus central pour réaliser leurs calculs et déterminer un état global, seraient beaucoup plus compliqués à implémenter sur une architecture totalement distribuée. D'autres travaux récents se sont, quant à eux, intéressés aux possibilités de migration du composant central d'une RTI pendant l'exécution de la simulation [Len09] mais dans notre cas cette piste n'est pas envisagée.

Nous pensons que ce travail est assez mature pour réfléchir à d'éventuelles propositions d'extensions de la norme HLA pour son utilisation dans les contexte des systèmes temps réel et des systèmes embarqués. Ces extensions pourraient prendre plusieurs formes. Dans un premier temps, nous pourrions nous inspirer des mécanismes additionnels du CERTI pour les rendre compatible avec n'importe quel intergiciel HLA :

- Affectation des priorités pour les fédérés, les composants locaux de la RTI (LRC) et le composant central (CRC) ;
- Choix d'une affinité processeur pour les fédérés, les LRCs et le CRC ;
- Configuration mémoire pour l'allocation dynamique des fédérés et de la RTI ainsi que la configuration sans pagination des fédérés, des LRCs et du CRC ;
- Ajout d'un service `tick2()` bloquant et spécifiquement dédié aux simulations temps réel ne souhaitant pas saturer le système d'exploitation par des appels répétitifs (sous un nom compatible avec la norme actuelle *i.e.* `evokeCallback()`).

Ces ajouts dans les fonctionnalités assurées par les RTIs devront être compatibles sur tous les systèmes d'exploitation et toutes les configurations, ce qui implique qu'il faudra faire une étude afin de définir des structures de données génériques et applicables sur différents systèmes d'exploitation.

On pourrait aussi imaginer des extensions dans les communications supportées, en s'inspirant des extensions proposées par Hui Zhao dans sa thèse [Zha01] :

- Un mode de transport `SHM` définissant les échanges par mémoire partagée (*C.f.* partie 5.1.5, page 99) et utilisé pour optimiser les communications sur le même nœud.
- Des modes de transports additionnels selon les nouveaux supports de communications implémentés, testés et supportés par les RTIs sur les plate-formes.

7.2.2 Les évolutions du simulateur de vol

Comme nous l'avons précisé précédemment, notre simulateur de vol se présente sous la forme d'une agrégation de fédérés HLA communiquant selon un modèle objet spécifique. Cette architecture particulière et ses principes de fonctionnement fournissent une architecture modulaire et extensible à souhait. Plusieurs pistes peuvent être envisagées pour les évolutions de ce simulateur :

- **Intégrer du matériel réel dans la boucle de simulation :** Le but est de remplacer certaines parties simulées du simulateur par du matériel réel. Nous pourrions ainsi utiliser des capteurs réels pour remplacer certains capteurs simulés par le fédéré du même nom, utiliser de vrais jauges pour certaines parties de la visualisation PFD, ...

- **Intégration sur des supports matériels opérationnels :** Une fois que nous aurons fait fonctionner des parties du simulateur avec du matériel réel, l'étape suivante sera de faire fonctionner certaines parties du simulateur (ou de nouvelles parties additionnelles) sur du matériel opérationnel tels que des drones. Cette perspective pourrait amener à des projets multi-disciplinaires intéressants.

- **Études sur le vol en formation :** Une autre perspective intéressante serait de combiner les deux cas test utilisés pour cette thèse afin de faire fonctionner plusieurs simulateurs de vol au sein de la même simulation. Cela imposerait des études complémentaires afin de déterminer les interactions entre les simulateurs entre eux et avec leur environnement commun, etc...

- **Collaborations avec des partenaires extérieurs :** Le principal avantage de l'utilisation d'une norme est la facilité de la collaboration. En effet, l'utilisation de la norme HLA permet de remplacer facilement un fédéré existant par un autre ayant le même rôle ou encore de rajouter un fédéré additionnel dans la boucle de simulation. Dans le cadre d'une collaboration extérieure (industriel, autre université,...), l'éventuel collaborateur devra simplement spécifier les données (correspondantes au modèle objet OMT) qu'il souhaite échanger avec le reste de la fédération. Ainsi, il conservera la propriété intellectuelle sur la sémantique de ses codes de calculs tout en participant à l'exécution d'une simulation globale. Cela permet d'envisager des projets multi-disciplinaires impliquant des spécialistes de différents domaines tels que des spécialistes propulsion pour le moteur ainsi que des simulations collaboratives de très grande échelle (plusieurs participants industriels ou universitaires répartis dans plusieurs pays).

7.2.3 Vers des simulations dirigées par le temps

Une perspective intéressante de cette thèse serait de mener des études complémentaires sur des simulations dirigées par le temps. En effet, dans cette thèse nous avons fourni un effort particulier pour définir des bases solides pour la validation de simulations distribuées utilisant le mode d'exécution *Data Flow*. Dans notre cas, nous supposons que les fédérés partagent la même

référence de temps : l'horloge matérielle pour les fédérés situés sur le même nœud et encore le support RCIM pour les nœuds Red Hawk de notre plateforme. Une évolution intéressante serait d'évaluer et d'intégrer des techniques de synchronisation logicielles supplémentaires telles que l'utilisation des protocoles PTP ou NTP (*C.f.* partie annexe B.4, page 170). Cela permettrait de tester la mise en œuvre du mode d'exécution *Data Flow* sur des supports distribués hétérogènes et des infrastructures distribuées à plus grande échelle. Pour cela, il faudrait évaluer la précision des protocoles de synchronisation utilisés pour vérifier qu'ils seront compatibles avec les contraintes temporelles spécifiques de l'application envisagée. Au niveau du modèle formel (*C.f.* partie 6.2, page 117), cela se traduirait par une prise en compte du délai de synchronisation dans les temps de calcul des tâches (*i.e.* les fédérés).

7.3 Bilan Général

La simulation distribuée permet la collaboration entre plusieurs disciplines d'études et de recherches par la mise en œuvre de simulations fédératrices et interopérables. La problématique traitée dans cette thèse est riche en impliquant différentes techniques et métiers du domaine de l'informatique. Les études présentées dans ce document ouvrent la voie vers des études annexes et complémentaires. Par ces travaux sur le temps réel, nous souhaitons encore accroitre l'utilisation des techniques de simulation, de la norme HLA ainsi que de l'infrastructure d'exécution du CERTI autour de projets fédérateurs et multi-disciplinaires tel que celui initié, par ce travail, dans le domaine des systèmes embarqués.

Troisième partie

ANNEXES

Annexe A

Les supports exécutifs temps réel

Préambule

Les contraintes du temps réel imposent aux systèmes informatiques de garantir le déterminisme temporel de l'ensemble des couches qui les composent. Cette annexe présente les différents efforts techniques de la littérature afin de fournir des supports exécutifs prédictibles assurant le déterminisme pour l'exécution d'une application temps réel. Pour cela, nous présenterons dans un premier temps les différents systèmes d'exploitation temps réel existants ainsi que leurs principes et spécificités. Dans un deuxième temps, nous évoquerons les techniques de conception et de programmation pour les systèmes temps réel en étudiant les différents langages ainsi que les compilateurs associés. Enfin, nous terminerons en évoquant les techniques déterministes d'allocation dynamique de mémoire mises en œuvre dans certains langages de programmation.

A.1 Les systèmes d'exploitation temps réel

A.1.1 Généralités

Un système d'exploitation temps réel est un système d'exploitation destiné spécifiquement à l'exécution d'applications qui doivent s'affranchir des incertitudes temporelles. Il n'est pas forcément plus rapide (plus performant) qu'un système d'exploitation multi-tâches classique mais il devra satisfaire les contraintes temporelles auxquelles est soumise l'application. Ainsi, il doit prendre en considération les fonctionnalités classiques des systèmes d'exploitation ainsi que certaines fonctionnalités supplémentaires essentielles pour garantir le déterminisme temporel aux applications. Cela consiste donc à gérer le parallélisme des tâches, les synchronisations et communications entre les différents processus (ou les threads), la protection de la mémoire et la gestion des interruptions. La composante principale est l'ordonnanceur à priorité qui permet de mettre en œuvre un ordre d'exécution entre les tâches en concurrence selon les principes de la théorie de l'ordonnancement (*C.f.* Annexe C, page 173). Une autre composante importante pour les systèmes d'exploitation temps réel est la gestion fine du temps à l'aide de primitives qui permettent de prendre en compte des délais avec une grande précision. Toute ces fonctionnalités additionnelles doivent donc être adaptées pour pouvoir permettre aux applications de bénéficier de services prédictibles.

Plusieurs systèmes d'exploitation temps réel ont été conçus, avec des particularités techniques propres à chacun. Les avancées des travaux de recherche sur le domaine ont, de plus, amené à

l'intégration de nouvelles fonctionnalités ainsi qu'à l'exportation des systèmes et des applications d'un matériel vers un autre (changement de processeur par exemple). Ainsi, pour assurer une cohérence et une compatibilité entre les différents systèmes d'exploitation, des APIs communes à tous les systèmes d'exploitation ont été introduites puis normalisées.

A.1.2 Vers des APIs normalisées

La complexité croissante des systèmes informatiques et de leurs fonctionnalités a nécessité très tôt des efforts de normalisation pour maintenir et exporter plus facilement les applications informatiques. La norme la plus connue est la norme POSIX[1] qui définit une API standard pour les différents services offerts par un système d'exploitation ainsi que les interfaces utilisables par les langages de programmation. Cette norme est, de plus, constituée de plusieurs documents pour la prise en compte des spécificités du temps réel [IEE93b, IEE95c] :

1. **1003.1b** est la première extension temps réel de POSIX incluant l'ordonnanceur à priorité, les horloges et les timers, les sémaphores ou encore les entrées/sorties synchrones et asynchrones.
2. **1003.1c** est l'extension spécifique de POSIX pour la définition des concepts et des sémantiques pour la gestion des processus légers : les threads (attributs, mutex avec inversion de priorité...).
3. **1003.1d** et **1003.1j** sont des extensions additionnelles de POSIX pour des fonctionnalités supplémentaires telles que la prise en compte de serveurs sporadiques, les verrous de lecture et d'écriture ou encore les nanosleeps plus précis.

Tous ces différents documents ont été regroupés dans le standard actuel POSIX 1.2008 dont une copie libre peut être téléchargée à l'adresse [49]. Le lecteur intéressé pourra trouver des résumés des spécifications POSIX dans les documents [CLG90, Har93]. Nous pouvons aussi citer l'alternative japonaise à POSIX, le standard ITRON [Sai91] [75], les efforts de normalisation pour le langage Java temps réel (RTSJ[2]) [GB00] [65] ou encore le standard OSEK[3] [OSE04, OSE05] [46] incontournable dans l'industrie automobile. De façon générale, les APIs temps réel proposent d'inclure dans les systèmes d'exploitation des fonctionnalités supplémentaires qui permettent une gestion plus fine des ressources. Cela inclut des primitives pour :

- les processus légers (*threads*) ;
- les horloges (les temporisateurs et les timers) ;
- l'ordonnancement par priorité (principalement des mécanismes statiques) ;
- la mémoire partagée ;
- les outils de synchronisation (sémaphores et mutex) ;
- les files de messages ;
- les événements asynchrones ;
- le verrouillage de zone mémoire (*memory locking*) ;
- les entrées/sorties asynchrones et synchrones.

A.1.3 Les systèmes d'exploitation temps réel existants

Il existe de très nombreux systèmes d'exploitation totalement ou partiellement temps-réel, qui tentent de fournire un support pour une des APIs normalisées aux applications (la plus

[1] *Portable Operating System Interface for uniX*
[2] *Real-Time Specifications for Java*
[3] *Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug (Allemand)*

couramment utilisée est POSIX). Nous donnerons ici, une liste non exhaustive des systèmes d'exploitation les plus utilisés :

- **VxWorks** [Win08] développé par la société Wind River [82] est le système d'exploitation temps réel commercial le plus utilisé dans le monde industriel. VxWorks propose un support exécutif multi-tâche préemptif, fiable, totalement configurable et compatible sur un grand nombre de processeurs tels que Power PC, Pentium ou encore ARM. Il est complètement compatible avec l'ensemble des spécifications POSIX et les langages utilisés sont le `C`, le `C++` et Ada. Il est notamment employé par la NASA pour la conception des systèmes pour les missions spatiales (Mars Pathfinder, Stardust, ...).
- **QnX** [Hil92] est un système d'exploitation temps réel commercial développé par QNX Software Systems [54]. Sa structure est de type Unix et il est compatible avec la norme POSIX. Son micro-noyau Neutrino lui confère des capacités temps-réel en fournissant un support d'exécution multi-tâche préemptif avec un contrôle fin du temps de réaction notamment pour la gestion des interruptions. C'est un système compatible avec les processeurs `x86` et SMP et le langage de programmation supporté est le `C`.
- **LynxOS** [Fra05] a été développé par la société LynuxWork [38]. Il est conforme à la norme POSIX et offre une compatibilité avec Linux. Il s'emploie surtout dans des systèmes embarqués tels que les logiciels critiques dans l'aviation ou le domaine militaire. En 2003, LynuxWorks a lancé une version LynxOS-178 complètement dédiée aux applications aéronautiques critiques qui doivent posséder la certification au standard DO-178B.
- **Windows NT**[4] [KRSS98] désigne la série de systèmes d'exploitation temps réel de Microsoft [85]. Ces systèmes sont multitâches préemptifs, multi-utilisateurs et multiprocesseurs et ils ont la particularité de ne pas reposer sur le noyau classique MS-DOS des systèmes Windows.
- **OSE**[5] est un système d'exploitation multitâche préemptif créé par la société ENEA [17]. Bien qu'il fut créé à l'origine pour un processeur particulier[6], il est aujourd'hui compatible avec un grand nombre de processeur dont ARM, PowerPC et MIPS mais ne fournit aucun support pour la norme POSIX. Une étude intéressante des pires temps d'exécution sur ce système d'exploitation peut être trouvée dans le manuscrit de Master de Martin Carlsson [Car02].
- **SHARK** [GAGB01] [66] est un noyau de recherche dédié aux applications temps réel partiellement compatible avec la norme POSIX. Le point fort de ce noyau est d'être entièrement configurable en termes de politiques d'ordonnancement, de serveurs apériodiques ainsi que de protocoles de contrôle de concurrence. Ces fonctionnalités ne sont généralement pas modulaires dans les autres systèmes d'exploitation temps réel.
- **MaRTE** [RH02] [39] est un noyau de recherche dédié pour les applications temps réel embarquées. Il fournit un sous-ensemble partiel de compatibilités avec la norme POSIX. Il a la particularité d'avoir été conçu et programmé principalement en Ada avec certaines parties C et en assembleur. Par conséquent, il supporte les langages Ada, `C` et `C++` et les travaux sont en cours pour intégrer un support pour le langage Java.
- **UNIX** est un système d'exploitation propriétaire développé à l'origine par Sun Microsystems. UNIX n'était pas nativement un système d'exploitation temps réel, mais différentes optimisations du noyau d'origine ont permis d'exporter ce système d'exploitation vers le monde du temps réel. Historiquement, SVR 4.2 [Zea97] fut la première version d'UNIX

[4] *Windows New Technology*
[5] *Operating System Embedded*
[6] *Le processeur Zilog 80*

à intégrer des modifications pour le temps réel notamment avec l'intégration d'un noyau complètement préemptible ainsi que d'un ordonnanceur à priorités fixes (60 niveaux de priorités). Ces modifications furent ensuite intégrées dans les versions courantes des systèmes SOLARIS [SUN00].

- **μITRON** [Sak99] est un noyau temps réel dur développé par les membres du standard ITRON et par conséquent il est complètement compatible avec les spécifications de la norme ITRON.
- **Jbed** [33, 34] est une machine virtuelle JAVA dédiée au système temps réel. Cette machine virtuelle est associée un système d'exploitation temps réel dédié. Il supporte dix niveaux de priorités pour les threads JAVA ainsi qu'un ordonnancement dynamique EDF.

Il existe un très grand nombre de systèmes d'exploitation temps réel qu'ils soient commerciaux, open-sources ou dédiés au monde universitaire. Le lecteur intéressé pourra se référer aux états de l'art sur les systèmes d'exploitation temps réel [RPA02, BM05] pour avoir une vision plus détaillée de l'ensemble des systèmes d'exploitation temps réel ainsi que de leurs spécificités. Une liste des systèmes temps réel (RTOS) existants est disponible sur le site de Wikipédia [81].

A.1.4 Le cas de LINUX

Linux, dont le noyau fut conçu en 1991 par Linus Torvalds, est le système d'exploitation open-source le plus utilisé au monde. Linux est un système généraliste qui n'est pas nativement dédié pour les systèmes temps réel. Néanmoins, les avantages de l'open-source et la fiabilité légendaire de Linux ont amené la communauté à améliorer le comportement du noyau afin qu'il puisse être compatible avec les contraintes d'un système temps réel. En ce sens, plusieurs contributions ont permis de faire évoluer Linux et le rendre compatible avec les exigences du temps-réel [FK03].

En effet, depuis l'intégration des patchs de préemption dans les noyaux Linux [37], plusieurs projets ont émergé afin de rendre les systèmes d'exploitation Linux prédictibles. RTAI [87] et XENOMAI [Ger04] [87] sont basés sur un noyau auxiliaire (indépendant du noyau Linux), intégrant un gestionnaire de tâches. Pour XENOMAI, ce partage des ressources matérielles entre les noyaux repose sur la technologie ADEOS[7] [Yag02] [4]. Le projet RT LINUX [59] était à l'origine un projet complètement open-source devenu industriel lorsqu'il fut racheté par la société Wind River[8]. Désormais, deux versions sont disponibles : une open-source (sous certaines conditions) et une autre entièrement industrielle. Dans ce travail de thèse, les expérimentations ont été réalisées en utilisant une version commerciale de Linux nommée Linux Red Hawk [BKBH08] de la société Concurrent [12]. Le système d'exploitation Linux Red Hawk est entièrement compatible avec la norme POSIX. De façon générale, les communautés de recherche pour la mise en œuvre de Linux dans le monde des systèmes temps réel et embarqués sont toujours très actives [36, 60]. Par exemple, nous pouvons évoquer un effort récent pour inclure un ordonnanceur dynamique (EDF) dans le noyau Linux [FBC09] qui ne supportait jusqu'à lors que des mécanismes statiques.

A.2 Conception et Implantation

A.2.1 Programmation

Les langages de programmation sont la base d'une implémentation d'un système informatique. Ils permettent au concepteur de définir les ensembles d'instructions logiques ou mathématiques

[7] *Adaptative Domain Environment for Operating System*
[8] *Les développeurs de VxWorks*

qui seront effectuées par l'ordinateur lors de l'exécution du programme [Dow97]. Le programme est décrit dans le langage de programmation par le concepteur et ensuite il est implanté dans le système. Historiquement, les premiers langages de programmation étaient très proches du matériel et donc peu compréhensibles pour un utilisateur novice. Avec le temps, les langages sont devenus de plus en plus abstraits pour s'approcher des concepts plus compréhensibles pour le programmeur non initié.

Il existe une grande diversité de langages de programmation mais, cependant, tous les langages ne permettent pas de développer des applications temps réel ou des applications relatives aux systèmes embarqués [BW01]. Pour la conception de systèmes temps réel, nous verrons les différents langages relatifs à une programmation bas-niveau (proche du matériel) dans la partie A.2.2 et les approches de plus haut niveau ensuite (partie A.2.3 et partie A.2.4).

A.2.2 Langages de bas-niveau

Historiquement, la technique la plus répandue pour programmer des systèmes temps réel et embarqués était d'utiliser le langage le plus proche du matériel : le langage Assembleur. Cependant, ce langage est tellement proche du matériel qu'il dépend fortement du type de processeur. Ainsi il n'existe pas un unique langage assembleur, mais un langage assembleur par type de processeur comme le langage assembleur pour les processeurs x86 [Car05] . En utilisant ce langage, il est donc nécessaire de connaître un minimum le fonctionnement du processeur utilisé pour pouvoir concevoir correctement une application. La création de nouveaux langages de programmation au début des années 70 tels que `C` [Ker88][9] ou Pascal [Wir71] avait pour but de simplifier la tâche du programmeur en lui proposant des instructions prêtes à l'emploi pour les fonctionnalités usuellement utilisées par les programmes informatiques. Plus tard, l'avènement du paradigme de programmation orienté objet a amené des nouveaux langages encore plus hauts niveau tels que ADA [BW98] [2, 3], `C++` [Str97] [7] ou en encore JAVA [GJSB05] [31]. Néanmoins, à part le langage Ada, ces nouveaux langages de programmation n'étaient pas nativement destinés à la programmation de systèmes temps réel.

De plus, ce haut niveau d'abstraction ajoute une étape supplémentaire lors de la conception d'un programme exécutable. En effet, un programme assembleur était directement converti en langage machine. Ainsi, le programmeur implémentait directement son programme sur le processeur et pouvait effectuer des optimisations directement sur ce code. Cela n'est plus vrai pour les langages de plus haut niveau où cette conversion se fait par l'utilisation d'un outil intermédiaire : le compilateur. Le compilateur est un programme informatique qui transforme le code source (le programme) en instructions pour le processeur (le langage machine). Cette transformation n'est pas toujours aussi optimisée que pourrait le souhaiter le concepteur. Différents compilateurs existent tels que le compilateur GNAT [24] pour Ada et les compilateurs GCC [23] ou TCC [72] pour le `C/C++` ou encore JAVAC [32] pour Java. Cependant, ils ne permettent pas tous de fournir des codes exécutables temps réel et/ou embarqués. Par exemple, dans le cas du `C` et du `C++` seulement quelques compilateurs sont admis comme propices à la compilation d'applications temps réel et/ou embarquées [Gau05]. Dans le cas du langage Java, le lecteur est invité à se référer au mémoire de Licence de Anders Nilsson dans lequel les différentes techniques de compilation pour Java temps réel sont décrites [Nil04].

[9] *La première édition de ce livre de référence date de 1978*

A.2.3 Les langages synchrones

Par la suite, la conception de systèmes temps réel s'est orientée vers des approches spécialisées avec un niveau d'abstraction encore plus élevé. Cela permet au concepteur de programmer un système temps réel sans tenir compte des détails de programmation complexes des langages de bas niveau (mécanismes de synchronisation, gestion de la mémoire, etc...). Ces détails techniques sont alors pris en compte lors de la traduction automatique du programme en langage machine par un compilateur ou un interpréteur. Généralement, les approches de haut niveau dédiées à la conception de système temps réel se basent sur une sémantique forte associée au langage qui permet de garantir la correction des programmes produits lors de l'opération de compilation. Toutefois, le code machine généré dépend une nouvelle fois de la qualité de l'outil de compilation utilisé.

L'approche synchrone propose une démarche de haut niveau très connue et utilisée pour la conception et la validation de systèmes temps réel [BB02]. Cette approche, reposant sur une abstraction du temps réel par un temps logique dans les modèles, est très utilisée pour sa simplicité, sa rigueur et ses bases formelles bien définies. Selon l'hypothèse synchrone, le concepteur fait abstraction de la durée des traitements d'un même instant et considère que tous les traitements effectués au cours d'un instant sont simultanés (donc synchrones). La validité de cette hypothèse est assurée tant que tous les traitements d'un même instant se terminent avant le début de l'instant suivant. Les langages les plus connus et utilisés sont les langages synchrones équationnels LUSTRE [Ber86, HCRP91] et le langage synchrone impératif ESTEREL [BG92, Ber00]. Un état de l'art sur les travaux relatifs aux langages synchrones peut être trouvé dans le document [BCE+03]. A l'ONERA, Julien Forget a développé pendant son doctorat [For09] un langage synchrone nommé PRELUDE. Il repose sur les mêmes hypothèse sémantiques que les langages synchrones classiques et hérite par conséquent de leurs propriétés formelles. Il inclut en plus des primitives pour permettre la programmation de systèmes multi-périodiques notamment en formalisant un concept pour des horloges périodiques de temps logique. Un compilateur associé à ce langage a aussi été implémenté et peut être téléchargé à partir de l'adresse [50].

A.2.4 Autres approches

Simulink [67] est un langage de haut niveau pour la modélisation de systèmes dynamiques. Il est utilisé dans de nombreux domaines et fournit un environnement graphique ainsi qu'un ensemble de bibliothèques très utiles pour la spécification et la simulation des systèmes (notamment grâce à son intégration dans l'outil MATLAB). Il permet de concevoir rapidement et de mettre en œuvre des systèmes temps réel tels que des systèmes de lois de contrôle et de commande. Néanmoins, le manque de propriétés sémantiques amènent les concepteurs des systèmes à bouder les compilateurs associés à Simulink lors de l'implantation du programme sur du matériel réel (et éventuellement opérationnel). D'autres approches, telles que les langages de description d'architecture, existent aussi pour la conception haut niveau de systèmes temps réel. Nous pouvons citer AADL[10] [FGH06] ou encore GIOTTO [HHK03, Hor94]. Enfin, nous pouvons terminer en citer l'outil SCADE[11] qui est un environnement de développement diffusé par ESTEREL Technologies [19]. Il est destiné à la conception de systèmes critiques. Le formalisme SCADE est une notation graphique formelle, basée sur le langage Lustre, qui permet de générer du code en langage `C` ou Ada. Le code généré peut être qualifié DO-178B (niveau A), ce qui explique sa popularité en aéronautique (il est notamment utilisé par Airbus).

[10] *Architecture Analysis and Design Language*
[11] *Safety Critical Application Development Environment*

A.3 Allocation mémoire déterministe

A.3.1 Position du problème

Lorsque l'on utilise la gestion dynamique de la mémoire, la mémoire est allouée pendant l'exécution du programme ce qui est plus souple que la gestion statique où la quantité mémoire doit être prévue et dimensionnée avant cette l'exécution. On peut classer les techniques de gestion de la mémoire dynamique en deux catégories :

1. ***Les mécanismes automatiques*** fournissent une gestion automatique de la mémoire dynamique intégrée au langage de programmation (par exemple dans les langages Java ou Lisp). Les gestionnaires automatiques de mémoire dynamique (souvent appelés les *Garbage Collectors*) recyclent des blocs qui deviennent inaccessibles à partir de variables de programme. De façon générale, ces gestionnaires permettent une gestion correcte de la mémoire dynamique. Cependant, ces services additionnels au programme peuvent consommer une grande quantité de temps processeur pour gérer cette mémoire dynamique [Zor93, SADLCJ01].

2. ***Les mécanismes manuels*** sont un ensemble de primitives destinées au programmeur qui a ainsi un contrôle direct sur le moment où la mémoire est allouée et quand elle pourra être libérée. Habituellement, cela se fait par des appels explicites du langage de programmation (tels que `malloc()` en `C` ou `new()` en `C++`). Ces mécanismes offrent la capacité au programmeur de contrôler toutes les opérations relatives à la gestion de la mémoire dynamique. Toutefois, cela oblige les programmeurs à implémenter leurs codes avec une grande rigueur car des problèmes de gestion de mémoire sont fréquents.

Ainsi, la gestion dynamique de la mémoire amène certains problèmes pour une utilisation dans le contexte des systèmes temps réel. D'une part, elle peut consommer une quantité non négligeable de la ressource processeur. De plus, elle peut aussi mettre en œuvre des mécanismes qui ne peuvent pas être bornés dans le temps. A cause de ces inconvénients, la gestion dynamique de la mémoire a été, jusqu'à une date récente, soigneusement évitée dans la conception des logiciels temps réel. Cependant, l'accroissement constant de la complexité des fonctionnalités ont poussé les concepteurs à gérer leurs ressources mémoire de manière dynamique en fournissant ainsi des mécanismes automatiques ou des mécanismes manuels d'allocation plus performants et adaptés aux contraintes temps réel.

Dans la suite, nous nous intéresserons aux différentes solutions relatives aux mécanismes manuels. En effet, notre middleware CERTI étant implémenté en `C` et `C++`, ce sont les langages que nous avons utilisés pour nos travaux. Nous nous sommes, par conséquent, intéressés aux différents travaux menés pour rendre prédictibles les mécanismes d'allocations dynamique de mémoire mis en œuvre dans les langages `C` et `C++`. Plus formellement les exigences auxquelles on souhaite répondre, sont des exigences de deux types : (1) temporelles et (2) spatiales. Temporelles, d'une part, car il faut pouvoir borner une opération de gestion mémoire (allocation, libération) ; spatiales, d'autre part, car il faut garantir qu'à un instant donné, le nombre d'allocations effectuées (et associées aux tailles demandées), permet de déterminer le nombre de requêtes supplémentaires de taille donnée qui pourront être satisfaites. Nous invitons le lecteur à se référer à l'article [WJNB95] pour un état de l'art complet sur les différents algorithmes d'allocations mémoires et leurs spécificités. Dans cette partie, nous verrons les approches classiques d'allocation manuelle de mémoire dynamique (*C.f.* partie A.3.2), puis les techniques d'allocation temps réel (*C.f.* partie A.3.3) puis pour finir une comparaison entre ces allocateurs (*C.f.* partie A.3.4).

A.3.2 Les allocateurs classiques

Les algorithmes d'allocation mémoire implantés dans les fonctions d'allocation telles que `malloc()` en `C` ou `malloc()` en `C++`, utilisent des approches First-Fit ou Best-Fit. Ces approches sont des approches séquentielles basées sur une liste unique des blocs mémoires disponibles quelles que soientt leurs tailles respectives.

Dans l'approche First-Fit, l'allocateur sélectionne le premier bloc libre de taille suffisante pour satisfaire la requête. Si l'espace restant est d'une taille suffisante, un nouveau bloc libre est inséré dans la liste des blocs libres. Lors de la libération d'un bloc il est possible ou pas de fusionner deux blocs libres, ce qui permet de réduire la fragmentation. Dans l'approche Best-Fit, l'allocateur sélectionne le bloc libre dont la taille est la plus proche (et suffisante) de celle du bloc requis. L'inconvénient de cette approche est une fragmentation [HC05] plus importante dues à l'existence de nombreux blocs mémoire de faible taille. Des études intéressantes (et relativement anciennes) sur des estimations des pires cas de fragmentation possibles avec ces deux approches est disponible dans les documents [Sho75, Rob77]. Une autre étude [Pua02] s'intéresse aux pires temps d'exécution de ces algorithmes First-Fit et Best Fit ainsi que leurs variantes optimisées (BTree Best-Fit, Quick-Fit, ...).

A.3.3 Les allocateurs temps réel

Pour répondre aux exigences et aux besoins du temps réel et en se basant sur les avancées des algorithmes d'allocation mémoire, [WJNB95] plusieurs allocateurs assurant des mécanismes en temps borné ont été proposés :

- **TLSF** est un allocateur de mémoire dynamique spécialement conçu pour répondre aux exigences du temps réel [MT07, MBR+07, MBRC08]. Les durées d'exécution dans le pire des cas pour les opérations d'allocation et de libération mémoire sont connues, bornées et indépendantes des applications. La complexité de l'algorithme sous-jacent (algorithme de type *Segregate-Fit*) est $O(1)$. Ainsi, le nombre d'instructions exécutées est très faible ce qui assure un niveau de réactivité élevée. La description algorithmique de TLSF complexe est détaillée dans [CMR05, MRR+08]. Les sources de cet allocateur (en langage C) sont également disponibles à partir du site [74].
- **DL malloc** est un allocateur implémenté par Doug Lea qui est en constante évolution depuis sa création 1987. C'est l'un des algorithmes d'allocation les plus utilisés (ou une de ses variantes comme le ptmalloc() [52]) dans certaines bibliothèques `C` et `C++` de GNU. Il combine de très nombreux mécanismes afin de trouver un compromis entre les différentes exigences des allocateurs temps-réel en termes de prédictibilité spatiale et temporelle. Le code de cet allocateur (en langage C) est également disponible à partir du site [15]. Le gestionnaire mémoire utilise des structures de données hybrides pour ranger les blocs libres. Un vecteur est ainsi utilisé pour référencer les blocs libres dont la taille est inférieure à une valeur prédéfinie. Les blocs de taille supérieure sont référencés par une liste doublement chaînée. Les premiers indices du vecteur référencent des blocs de même taille ce qui accélère la recherche. Les autres indices du vecteur référencent des blocs mémoire rangés par taille décroissante. L'algorithme utilise ensuite des stratégies différentes pour la recherche des blocs libres selon les tailles demandées pour les nouvelles allocations.
- L'allocateur **Half-Fit** [Oga95] propose une approche similaire à celle de TLSF en maintenant plusieurs listes des blocs libres selon leurs tailles. Les coûts des opérations d'allocation et de libération sont bornés et la complexité est aussi $O(1)$. La structure de donnée utilisée est un vecteur de listes ; chacune d'elles adressant des blocs libres dont la taille est com-

prise entre la puissance 2 de l'indice de la liste courante dans le vecteur et la puissance 2 de l'indice de la liste suivante.

- L'allocateur **Compact-Fit** a été développé à l'Université de Salzburg [Pay07, CKP+08]. Il est basé sur un principe de partitionnement de la mémoire en pages de 16 Koctets, chaque page étant elle-même découpée en blocs de taille identique. Cet allocateur est disponible en deux versions CFM[12] et CFNM[13].
- Certains travaux abordent le problème d'une gestion mémoire directement intégrable dans les classes conteneurs de la STL[14] (vecteurs, listes ...). Pete Isensee propose quelques exemples d'allocateurs personnalisés dont certains codes sont téléchargeables gratuitement [68]. Ces mécanismes pourraient être utilisés pour implémenter des versions temps réel des allocateurs utilisés et faire hériter l'ensemble des conteneurs STL de propriétés déterministes.

A.3.4 Comparaisons

Plusieurs études tentent de comparer les performances de ces gestionnaires de la mémoire dynamique [CMR06, Pay07]. Ces études comparatives posent le problème des métriques de référence pour lesquelles il n'existe pas de consensus réel. L'efficacité temporelle (performance) d'un gestionnaire mémoire est souvent évaluée par deux facteurs :

1. Le nombre de cycles processeur, difficile à mesurer et qui dépend de la nature du processeur ;
2. Le nombre d'instructions exécutées par le gestionnaire pour effectuer les opérations.

Et pour mesurer l'efficacité spatiale d'un gestionnaire mémoire (fragmentation), on s'accorde sur 4 métriques [JW98] :

1. La ressource mémoire utilisée par le gestionnaire par rapport à la quantité de mémoire requise par l'application ;
2. La ressource mémoire utilisée par le gestionnaire par rapport à la quantité de mémoire requise par l'application à l'instant où la demande est maximale ;
3. La ressource mémoire maximale utilisée par le gestionnaire par rapport à la quantité de mémoire requise par l'application à l'instant où cette ressource est maximale ;
4. La ressource mémoire maximale utilisée par le gestionnaire par rapport à la quantité de mémoire maximale requise par l'application.

L'étude la plus poussée sur la comparaison des différents allocateurs peut être trouvée dans [CMR06, MRR+08]. Les auteurs présentent une évaluation approfondie sur les performances temporelles et spatiales de ces différents allocateurs mémoire. Cette étude montre que le gestionnaire mémoire TLSF offre le meilleur compromis entre efficacité spatiale et temporelle. De plus, des études annexes montrent que le taux moyen de fragmentation pour TLSF est inférieur à 25% dans le pire des cas [MBR+07, MBRC08]. Dans ce travail de thèse, c'est l'allocateur temps réel que nous avons choisi pour intégrer au CERTI des mécanismes déterministes pour la gestion dynamique de mémoire.

[12] *Compact-Fit Moving*
[13] *Compact-Fit Non-Moving*
[14] *Standard Template Library*

Annexe B

Les supports de communication temps réel

Le temps réel distribué implique des systèmes informatiques dans lesquels plusieurs unités d'exécution communiquent par un médium de communication pour accomplir un calcul global. De la même façon que pour les systèmes temps réel mono-processeurs, le déterminisme de l'ensemble des parties du système distribué est à étudier et donc il faut aussi prendre en considération la ressource de communication. Pour cela, cette annexe présente les différents points essentiels à prendre en compte sur les technologies du réseau afin d'assurer des supports déterministes pour les communications dans un système temps réel distribué. Dans un premier temps, nous présenterons les efforts de normalisation des technologies du réseau ainsi que les caractéristiques importantes de l'infrastructure physique ayant un impact sur la qualité et les performances temps réel du réseau. Dans un deuxième temps, nous évoquerons les différents protocoles MAC existants dédiés à arbitrer l'accès au réseau entre les différents nœuds du système distribué. Nous poursuivrons ensuite, dans un troisième temps, sur les différents protocoles de transport chargés, quant à eux, du transport effectif des données. Enfin, nous terminerons sur les protocoles de synchronisation existants qui sont utilisés afin de fournir une horloge globale pour tout un système distribué.

B.1 Normalisation et supports physiques

B.1.1 Les efforts de normalisation

Les technologies des réseaux informatique sont apparues dans les années 70-80 avec la mise en œuvre du réseau ARPANET [BI81]. Depuis, de nombreux supports physiques ainsi que de nombreux protocoles pour la gestion de ces supports (accès, transfert,...) sont apparus. La très grande diversité des technologies a nécessité des efforts de normalisation et actuellement la norme en vigueur est la norme OSI[1] [Bur85]. Le modèle OSI, basé sur le modèle plus ancien SNA[2] [McF76] de IBM, décompose les technologies réseaux en 7 couches distinctes (*C.f.* tableau B.1). Ce découpage va de l'infrastructure physique (le support matériel) jusqu'à l'application considérée (le programme souhaitant communiquer).

[1] *Open Systems Interconnection*
[2] *Systems Network Architecture*

Nom de la couche	Explications
1. La couche « physique »	Transmission effective des signaux sur le support physique Émission et la réception d'un ou plusieurs bit(s)
2. La couche « liaison de données »	Encodage des données pour le transport par la couche physique Détection d'erreur de transmission et synchronisation
3. La couche « réseau »	Transport, adressage et routage des paquets encodés Le protocole réseau le plus connu est IP.
4. La couche « transport »	Communication effective d'un bout à l'autre fragmentation des données en petits paquets, contrôle de transmission Les protocoles de transport les plus connus sont TCP et UDP.
5. La couche « session »	Établissement et maintien des sessions
6. La couche « présentation »	Représentation des données du codage des données applicatives Conversion entre données applicatives et octets effectivement transmis
7. La couche « application »	Point d'accès aux services réseaux logiciels : navigateur, logiciel d'email, FTP, chat...

TAB. B.1 – Liste des couches du modèle OSI

Ce modèle OSI, fondé sur ce découpage en différents niveaux où les protocoles sont indépendants, souffre de sa trop forte normalisation. Généralement, les couches 5, 6 et 7 sont difficiles à distinguer pour le concepteur et elles sont donc rarement implémentées séparément. Ainsi, on fusionne couramment ces niveaux dans la couche applicative pour se retrouver avec un modèle hybride à cinq couches (*C.f.* Figure B.1)

FIG. B.1 – Modèle hybride à 5 couches

En nous basant sur ce découpage hybride, nous présenterons brièvement les caractéristiques du support physique (Niveau 1) dans la partie suivante (*C.f.* partie B.1.2). Le respect de la ponctualité d'un système distribué nécessitant un ordonnancement approprié des communications, nous étudierons ensuite les différentes protocoles MAC (Niveau 2) chargés de garantir cette propriété (*C.f.* partie B.2). Au dessus de ces protocoles MAC, ce sont les protocoles des niveaux *Réseau* et *Transport* (Niveau 3 et 4) qui sont chargés du transport explicite des données entre les différents nœuds. Donc, nous étudierons les différents protocoles existants ainsi que les solutions envisagées pour garantir une certaine qualité de service à ces niveaux-là (*C.f.* partie B.3). Dans notre cas, l'application (Niveau 5) est composée des différents fédérés qui communiquent par le CERTI au travers des différentes couches présentées précédemment ; il convient donc de choisir un des modes d'exécutions (*C.f.* partie 4.3, page 80) pour chaque application interconnectée.

B.1.2 La couche physique *[Niveau OSI 1]* et La Qualité de service

Les nouvelles contraintes des systèmes temps réel ont amené des évolutions dans les technologies existantes afin de prendre en compte et surtout d'assurer certains paramètres de qualité de service. Le *débit* ou la *bande passante* représente le volume de données qui peut transiter sur le réseau en une seconde. Généralement, cette bande passante est mesurée en octets par secondes (ou bits par seconde). La *latence* désigne le temps nécessaire pour le transport d'un paquet de données de la source jusqu'à la destination. La variation de la latence pour un même trajet est appelée le *jitter* ou la *gigue*. Une gigue élevée, traduisant des délais fortement variables, peut être pénalisante pour des applications temps réel souhaitant des communications déterministes. La *fiabilité* représente l'assurance que toutes les données émises sont bien reçues. La fiabilité est surtout gérée au niveau des protocoles de réseau et de transport (*C.f.* partie B.3). De façon générale, les protocoles fiables garantissent l'arrivée d'un paquet en mettant en œuvre des mécanismes additionnels qui peuvent limiter la performance. Les mécanismes non fiables sont plus rapides mais il n'y a aucun moyen de savoir si l'information est bien arrivée et il faut évaluer le *taux de perte de paquet*.

Les paramètres de qualité de service du débit et de la latence reposent en premier lieu sur la couche physique choisie pour les communications (niveau 1 du modèle OSI). Plusieurs facteurs sont à prendre en compte au niveau de la couche physique :

1. Le *support physique* de transmission : câbles dans lesquels circulent des signaux électriques, des fibres optiques qui propagent des ondes lumineuses ou encore l'atmosphère où circulent des ondes radio (*Wi-Fi* par exemple).
2. La *portée géographique* du réseau (distance entre les nœuds) qui peut aller du réseau local LAN[3] au réseau mondial WAN[4]
3. La topologie (la configuration spatiale) a aussi une influence sur la qualité des transmissions. Plusieurs topologies existent selon le type des supports physiques et la portée géographique. Par exemple, la topologie en étoile (*C.f.* Figure B.2) permet d'interconnecter les nœuds par un intermédiaire commun (le concentrateur ou le hub). Dans une topologie en anneau (*C.f.* Figure B.3), les nœuds sont situés dans une boucle et communiquent à tour de rôle.

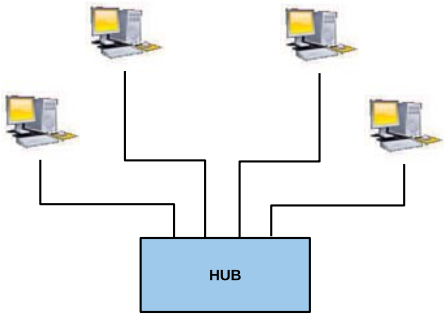


FIG. B.2 – Topologie du réseau en étoile

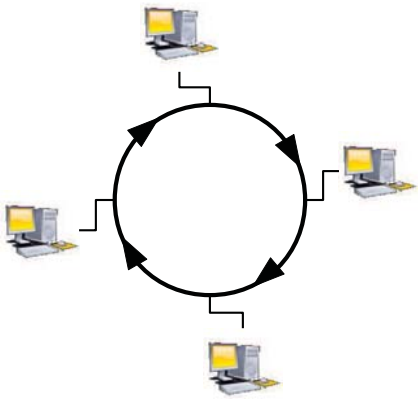


FIG. B.3 – Topologie du réseau en anneau

Ethernet [Spu00], créé à l'origine par Xerox, est le support physique réseau le plus répandu et le plus utilisé pour les LANs et il est aussi adapté pour les WANs depuis l'essor des supports

[3] *Local Area Network*
[4] *World Area Network*

gigabits plus performants [Ten02]. De plus, ces dernières années, son déploiement industriel a connu un essor grandissant notamment pour la mise en œuvre de communications temps réel critiques. Des exemples connus sont le réseau AFDX [CES03, Cha07] embarqué dans les Aibus A380 et le réseau PROFINET, créé par la société PI [51] qui est un réseau pour l'automatisation industrielle basé lui aussi sur Ethernet.

Pour notre plate-forme PRISE, nous avons choisi d'utiliser du matériel réseau standard dans un premier temps (*C.f.* partie 4.1.2, page 70). En ce sens, notre système distribué repose sur un réseau Eternet LAN qui met en œuvre une topologie en étoile. Nous possédons aussi une connexion extérieure pour étendre nos simulations sur des plus grandes portées géographiques (réseau WAN).

B.2 Les protocoles MAC *[Niveau OSI 2]*

B.2.1 Généralités et classification

Au-dessus de la couche physique, c'est la sous-couche MAC[5] (correspondant à la couche 2 du modèle OSI), qui gère l'accès à la ressource de transmission (ressource commune à toutes les stations, et donc à toutes les tâches). Le choix de ce protocole est essentiel pour assurer le déroulement temps-réel des applications [KSY84, MZ95]. La classification des protocoles MAC temps réel distingue deux processus pour assurer la gestion de la communication :

1. le processus d'arbitrage d'accès qui détermine *quand* un nœud a le droit d'utiliser la ressource de transmission (d'émettre un message sur le réseau).
2. le processus de contrôle de transmission qui détermine *combien de temps* le nœud a le droit d'utiliser la ressource de transmission.

De façon générale, les protocoles MAC mettent en œuvre chacun de ces deux processus pour la gestion de la ressource de transmission. Cependant, les mécanismes mis en œuvre dans chaque protocole reposent principalement sur l'un ou l'autre des deux processus qui joue alors un rôle prépondérant. Ainsi, même si cela n'est pas tout à fait correct, la classification la plus courante divise les protocoles MAC en deux catégories selon l'importance relative au rôle de chaque processus dans l'accès au média de communication.

Francisco Vasques de Carvalho, dans le troisième chapitre de son manuscrit de doctorat [VdC96], propose une classification plus élégante basée sur l'ordonnancement :

1. Une catégorie où l'accent est mis sur un ordonnancement basé sur l'assignation de priorité aux flux des messages. Les protocoles MAC de cette catégorie travaillent à partir des contraintes temporelles auxquelles sont soumis les différents flux de messages. Le protocole MAC interprète ces contraintes et réalise un ordonnancement global pour assurer les transferts.
2. Une catégorie où l'accent est mis sur un ordonnancement basé sur une garantie d'accès de temps borné aux stations. Les protocoles MAC de cette catégorie offrent un accès en temps limité et en exclusion mutuelle à la ressource de communication. L'ordonnancement global des flux de messages est alors déterminé en fonction de ces temps d'accès (ordonnancements locaux).

Pour la suite, sans tenter de classifier les différents protocoles MAC, nous décrirons les principaux protocoles MAC temps réel dans un premier temps (*C.f* partie B.2.2) puis nous discuterons

[5] *Medium Access Control*

des limitations du protocole MAC mis en œuvre sur les réseaux Ethernet : le protocole CSMA/CD (*C.f* partie B.2.3).

B.2.2 Les différents protocoles MAC temps réel

Les différentes techniques MAC temps réel pour assurer une gestion de l'accès au réseau peuvent être résumées et illustrées par la figure B.4.

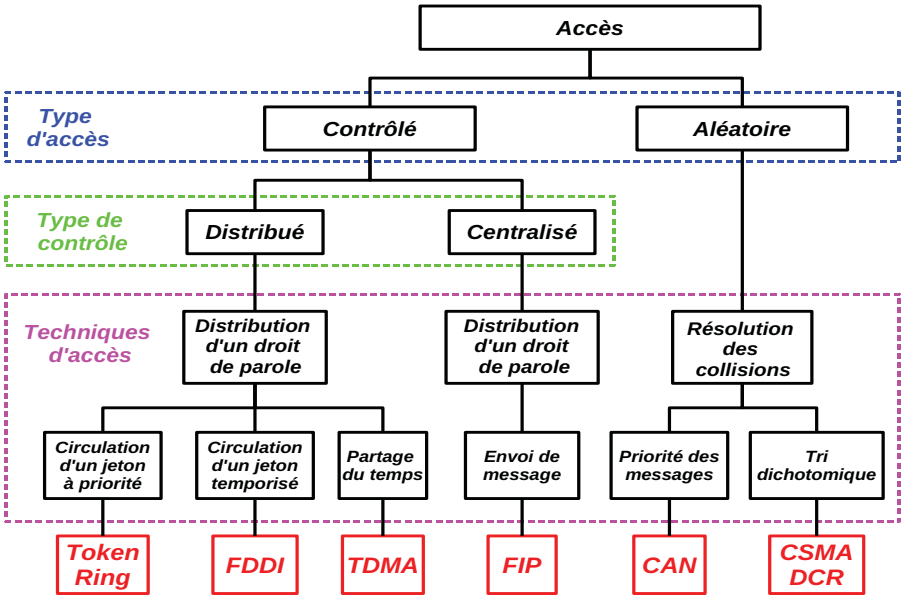


FIG. B.4 – Les techniques d'accès au réseau de différents protocoles MAC

- **Token Ring** est un protocole MAC, initié par les travaux de Farmer et Newhall [FN69], puis adopté et normalisé par l'IEEE sous le nom IEEE 802.5 [IEE98]. Ce protocole est basé sur la circulation d'un jeton à priorité qui autorise le nœud qui le détient à émettre sur le support de communication. Le protocole Token Ring utilise une trame spéciale de trois octets, appelée *jeton*, qui circule de processeur en processeur pour distribuer une priorité à chaque nœud de l'application. Token Ring supporte huit niveaux de priorité différents et les techniques d'assignation de ces priorités peuvent avoir un impact sur les communications [CS92].
- **FDDI**[6] est un protocole qui repose sur un support physique spécifique selon une topologie en anneau et une fibre optique multimode comme support de transmission. Ce protocole MAC, normalisé par l'ISO[7] [ISO89], est basé sur la circulation d'un jeton temporisé qui autorise le nœud qui le détient à émettre. Le principe du protocole de gestion du jeton est d'allouer à chaque nœud une durée pré-définie qui représente le temps maximal pendant lequel il peut transmettre chaque fois qu'il reçoit le jeton. Cette durée est déterminée à partir du temps maximal de rotation nécessaire au jeton pour faire un tour complet de l'anneau (le paramètre TTRT[8]). Ainsi, chaque nœud dispose d'une fraction de la transmission exprimée en pourcentage de ce paramètre TTRT qui a une importance sur les temps de communication entre les sites [SJ87, ZS95].

[6] *Fiber Distributed Data Interconnect*
[7] *International Organization for Standardization*
[8] *Target Token Rotation Time*

- **TDMA**[9] [IEC00] est un protocole MAC basé sur une division du temps global en une séquence d'intervalles. Chaque intervalle est assigné à un seul nœud (le nombre d'intervalles assignés est égal au nombre de nœuds) et ainsi le temps global était partagé équitablement. Une séquence périodique d'intervalles est appelée un *cycle*. Il est aussi envisageable d'assigner un nombre d'intervalles supérieur pour les stations ayant des contraintes spécifiques (communication dense et de haute fréquence). Une variante FDMA[10] est un mode de multiplexage destiné à la téléphonie mobile. Il s'agit d'un découpage en bandes de fréquences de manière à attribuer une partie du spectre (une bande de fréquence distincte) à chaque utilisateur. La variante CDMA est basée sur un système de codage des transmissions, utilisant la technique d'étalement de spectre. Il permet à plusieurs liaisons numériques d'utiliser simultanément la même fréquence porteuse. Des comparaisons entre ces différentes variantes sont disponibles dans les documents [FN98, Kol06].
- **FIP**[11] [Cot96] est un protocole MAC utilisé sur le réseau de terrain du même nom assurant des transferts d'informations entre différents équipements opérationnels (capteurs, actionneurs ou encore des unités de commande). Dans ce protocole MAC, l'ordonnancement des échanges de messages est basé sur un contrôleur central appelé *arbitre de bus*. La démarche consiste à construire, hors-ligne, un ordonnancement (noté *table de scrutation*) qui sera suivi, en-ligne, par l'arbitre de bus pour déterminer l'accès au réseau. L'arbitre de bus diffuse par envoi d'un message l'indentifiant du producteur qui peut émettre sur le réseau.
- **CAN**[12] est un protocole MAC (et aussi un support physique), initialement proposé par l'industriel BOSH [BOS91], normalisé par l'ISO[13] [ISO99]. Il est utilisé pour gérer la communication entre les systèmes embarqués dans les véhicules automobiles. Ce protocole repose sur l'assignation d'une priorité pour les messages transitant sur le réseau et, par conséquent, la politique choisie pour cette assignation est importante pour le déterminisme du réseau. Un schéma d'assignation des priorités, basé sur une variante non preemptive de l'algorithme Deadline Monotonic, a été proposé dans [THW94, DBBL07].
- **CSMA/DCR**[14] est une variante du protocole MAC classique d'Ethernet (*C.f.* partie B.2.3, page 167) proposée par l'INRIA [Le 93]. Ce protocole MAC est basé sur la résolution des conflits au bus (les collisions) par tri dichotomique. Le mécanisme de résolution du conflit est établi grâce à un découpage dichotomique basé sur les adresses des sites. Lors d'une occurrence de collision (deux stations émettent en même temps), appelée une *époque*, tous les sites vont pouvoir émettre dans un ordre fixé. Cela permet de connaître le délai maximum de transfert suivant la position de chacun (déterminisme).

Il existe de nombreux protocoles MAC dans la littérature plus ou moins adaptés au contexte des systèmes temps réel. Nous pouvons encore citer le protocole TTP[15] [KG94] qui constitue une variante de TDMA définie dans le cadre du système d'exploitation MARS. Dans ce protocole, le temps global est aussi divisé en une séquence de tranches de temps chacune étant assignée à un nœud particulier. Le protocole VT-CSMA [ZR87] est une autre variante de l'algorithme classique d'Ethernet (*C.f.* partie B.2.3) dédiée aux systèmes temps réel qui attribue l'accès au médium de communication selon le principe d'une fenêtre de génération. Ainsi la gestion des collisions se fait soit automatiquement en adaptant les fenêtres en fonction des nœuds en concurrence ou une

[9] *Time Division Multiple Access*
[10] *Frequency Division Multiple Access*
[11] *Factory Instrumentation Protocol*
[12] *Controller Area Network*
[13] *International Standard Organization*
[14] *Carrier Sense Multiple Access/Deterministic Collision Resolution*
[15] *Time Tiggered Protocol*

autre méthode consiste à positionner la fenêtre de façon arbitraire [KSY84]. Le réseau μITRON bus [MMTS96] repose aussi sur une variante du protocole classique d'Ethernet, basée sur une résolution des conflits par attribution de priorités, qui est utilisée pour interconnecter les systèmes compatibles avec la norme ITRON (*C.f.* partie A.1.3, page 152). Samia Saad-Bouzefrane dans son manuscrit de thèse [SB98] (dans la partie 3.1.1) propose une étude analytique intéressante pour le calcul des délais de propagation des messages en fonction de certains protocoles MAC cités précédemment.

B.2.3 Le protocole MAC CSMA/CD

Le protocole MAC CSMA/CD[16] a été proposé initialement en 1975 afin de résoudre les problèmes de collisions récurrents dans l'utilisation du protocole initial ALOHA [Abr70] puis fut ensuite normalisé par l'IEEE sous le nom IEEE 802.3 [IEE00a] [28]. C'est le protocole MAC qui gère le partage de l'accès au support de communication sur le réseau Ethernet. Si un nœud veut émettre, il écoute le réseau pour savoir s'il y a déjà une autre émission en cours. S'il n'y a pas de collisions le nœud peut émettre son message. En revanche s'il y a une collision, le nœud doit attendre un délai aléatoire tiré dans un intervalle de temps borné. Si par la suite, une collision est encore produite à la prochaine émission, le même processus d'attente est renouvelé mais dans un intervalle de temps doublé. Ainsi, au fur et à mesure, la probabilité de réussite des transmissions est forte car le risque de collision est amoindri par l'augmentation de cet intervalle de temps possible pour l'attente. Ce mécanisme empêche ainsi l'interruption d'une communication active mais ne permet pas d'éviter des accès simultanés au bus. Dès que des transmissions sont initiées au même instant, une collision est inévitablement produite traduisant une destruction puis une reconstruction des informations envoyées.

Ces collisions rendent ce protocole probabiliste inadapté aux contraintes de prédictibilité des systèmes temps réel car on ne peut généralement pas borner le temps nécessaire à une station pour pouvoir émettre sur le réseau. En effet, au début de son utilisation, le réseau Ethernet était utilisé pour interconnecter (sur des LANs) différents nœuds selon des topologies en bus. Dans cette topologie (la plus simple) tous les ordinateurs sont reliés à la même ligne de communication et les collisions sont fréquentes. Des études sur l'évaluation des performances [BMK88] ou encore sur des évaluations pour les temps d'accès dans le contexte du temps réel [Hai83] ont donc été réalisées très tôt sur ces topologies en bus. De nos jours, les technologies Ethernet proposent des solutions très performantes (augmentation importante du débit) et, de plus, l'utilisation des commutateurs permet de diminuer les domaines de collisions sur un réseau local entre les différents nœuds du système distribué [Geo05]. Dans notre cas, les mesures des temps de transfert, montrent que les performances sur notre réseau local sont acceptables pour répondre aux contraintes temporelles des simulations de systèmes embarqués (*C.f.* partie 5.2.1, page 101). Néanmoins, l'utilisation d'un protocole MAC déterministe est à envisager dans les perspectives de travail de cette thèse. Dans la partie suivante, nous étudierons les protocoles et les techniques pour gérer la qualité de service au niveau des couches *Réseau* et *Transport* sur un réseau Ethernet.

[16] *Carrier Sense Multiple Access with Collision Detection*

B.3 Couches Réseau et Transport *[Niveaux OSI 3 et 4]*

B.3.1 Les Protocoles de Transport classiques d'Internet

Les protocoles de réseau et de transport définissent la manière dont les informations sont échangées entre les équipements du réseau. Des logiciels spécifiques pour la mise en œuvre de ces protocoles sont installés sur les équipements d'interconnexion (commutateurs, routeurs, ...). Ces fonctions de contrôle permettent une communication entre les équipements connectés d'un système distribué par Ethernet tel que la plate-forme PRISE. Le protocole réseau (Niveau OSI 3) probablement le plus répandu est le protocole IP[17] et il est généralement associé aux protocoles de communication (niveau OSI 4) UDP[18] et TCP[19] :

- La couche **IP** [Inf81] gère la transmission de paquets d'un émetteur A à un destinataire B sans garantie d'acheminement ou d'ordre. Elle s'occupe du fractionnement des paquets (les données d'un message) en unités de petite taille en accord avec la capacité des supports physiques sous-jacents. Les protocoles de niveaux supérieurs s'appuient sur la couche IP pour le transport en encapsulant leurs données dans les paquets IP.
- Le protocole **UDP** [Pos80b] permet l'envoi de données d'une manière non sécurisée au-dessus de IP. Le rôle de ce protocole est de permettre la transmission de données de manière très simple entre deux entités, chacune étant définie par une adresse IP et un numéro de port (pour différencier différents utilisateurs sur la même machine). En utilisant ce protocole l'émetteur n'a aucun moyen de vérifier si toutes les données envoyées sont bien arrivées à destination, ni dans quel ordre elles sont arrivées. En revanche, pour un paquet UDP donné, l'exactitude du contenu des données est assurée grâce à une somme de contrôle (*i.e.* checksum). Ce protocole supporte des communications de type *unicast* (un émetteur et un récepteur) et des communications de type *multicast* (un émetteur et plusieurs récepteurs). Le principal avantage du multicast est de pouvoir envoyer le flux une seule fois quelque soit le nombre de récepteurs, ce qui fait baisser la charge du réseau.
- Le protocole **TCP** [Pos80a] permet au contraire une transmission fiable des données au-dessus de IP. Ce protocole garantit à l'émetteur qu'une information envoyée a bien été reçue ou alors le prévient d'un problème (après plusieurs essais infructueux). TCP s'occupe également de garantir l'ordre de livraison des paquets transmis sur la couche IP pour reconstruire le message initial. Ce protocole supporte des communications de type *unicast* mais ne supporte pas nativement des communications de type *multicast* sans hypothèses et implémentations additionnelles. Un exemple d'extension multicast du protocole TCP est le réseau *TCP Overlay* développé à l'INRIA [BCLR04].

Ces protocoles, bien que très couramment utilisés, ne sont pas adaptés pour assurer des communications dans les systèmes temps réel. Lorsque l'on utilise le protocole UDP, il n'y a aucun moyen de garantir l'arrivée des paquets et certains peuvent se perdre pendant les communications [CB01]. Le protocole TCP, quant à lui, est un protocole de transport dit "élastique" qui s'adapte au taux de transfert à la bande passante disponible sur le réseau. Il n'assure pas un taux minimum de transfert mais assure seulement la livraison des données de façon fiable. Pour cela, le TCP prévoit des mécanismes de gestion de la congestion[20] et il existe une multitude d'algorithmes d'évitement de congestion pour TCP. Le lecteur intéressé pourra se référer à l'article [Jac88] pour une description des algorithmes originaux de contrôle de congestion. De nos jours, les noyaux

[17] *Internet Protocol*
[18] *User Datagram Protocol*
[19] *Transmission Control Protocol*
[20] *La congestion intervient lorsque le réseau est saturé.*

Linux (depuis la version 2.6.19) utilisent l'algorithme TCP CUBIC [HRX08] alors que les systèmes d'exploitation Windows utilise TCP COMPOUND [Tan06]. Une étude comparative entre ces différents algorithmes de gestion de la congestion peut être trouvée dans [HS08].

Rappelons que dans la norme HLA, les modes de transport disponibles sont les modes *reliable* et *best-effort* qui reposent respectivement sur les protocoles TCP et UDP. Généralement, UDP (et donc le mode de transport *best-effort*) intervient lorsque le temps de remise des paquets est prédominant mais sans contrainte sur une réception de ces paquets dans tous les cas. Inversement, les applications nécessitant une transmission fiabilisée et ordonnée d'un flux de données implémenteront de préférence le protocole TCP (et donc le mode de transport HLA *reliable*). Les systèmes embarqués nécessitent des communications sûres, donc nous avons réalisé l'ensemble de nos expérimentations en utilisant le protocole TCP. Les mesures que nous avons effectuées montrent que les performances du CERTI utilisant des communications TCP sont compatibles avec des contraintes temps réel sur notre réseau LAN dédié (*C.f.* partie 5.2.1, page 101).

B.3.2 Les Protocoles Temps Réel

La qualité de service dans les réseaux filaires (tels que Ethernet) peut être garantie, dans un premier temps, par le surdimensionnement des ressources en conservant les protocoles classiques. Cela permet d'assurer une bande passante suffisante pour éviter la congestion, des délais des files d'attentes convenables ou encore des pertes des paquets limitées. Cette approche est facile à mettre en œuvre et permet aux protocoles classiques (en particulier TCP et ses mécanismes de contrôle de congestion) de mieux supporter les applications temps réel. Cependant, cela ne permet pas de traiter les émetteurs ou les messages selon leur importance : tous les nœuds et tous les flux ont la même priorité. Le fonctionnement du réseau, même surdimensionné, peut devenir incompatible avec le temps réel lorsque, par exemple, des parties du réseau subissent une surcharge. Plusieurs travaux ont donc été menés pour pallier aux inconvénients des approches classiques sur les réseaux filaires (en particulier Ethernet) :

- Le modèle IntServ[21] [Bra94] définit une architecture capable de prendre en charge la qualité de service sans toucher au protocole IP. Le modèle est basé sur la définition de classes de service et la réservation de ressources dans les différents éléments du réseau. Pour cela, IntServ définit deux classes de services : la classe *Controlled-load* [Wro97a] est dédiée aux applications temps réel souples et la classe *Guaranteed* [She97] est dédiée aux applications temps réel dures non tolérantes. La réservation de ressources sur le réseau peut se faire de façon statique ou dynamique. Une proposition de protocole de réservation dynamique appelé RSVP[22] a été faîte dans le document [Bra97] et son utilisation dans le contexte de IntServ est décrite dans le document [Wro97b]. RSVP est un mécanisme de reservation de ressources qui permet d'allouer dynamiquement de la bande passante aux applications unicast et multicast en s'adaptant aux évolutions du réseau (nombre de participants, changements de routes). Il demande des ressources dans une seule direction et traite l'émetteur et le récepteur de manière différente. Le modèle IntServ (et le protocole de réservation) sont dédiés pour les réseau LAN et sont donc difficilement applicables dans le cas de grands réseaux.
- Le modèle DiffServ[23] [Bla98] propose, quant à lui, de séparer le trafic par classes en traitant tous les paquets d'une classe donnée de la même manière. Pour cela, le modèle DiffServ

[21] *Integrated Services*
[22] *Resource ReSerVation Protocol*
[23] *Differentiated Services*

redéfinit le champ existant TOS[24] [Pos81] de l'entête IP (version 4) en y encodant les différentes classes de service. Cela permet de ne pas remettre en cause ce protocole IP et évite ainsi l'usage additionnel d'un protocole de signalisation à implémenter et mettre en œuvre (tel que le protocole RSVP).

– Nous pouvons citer aussi le protocole RTP[25] [Sch03] dont le but est de fournir un moyen uniforme de transmettre sur IP des données soumises à des contraintes de temps réel souple telles que des données audios ou vidéos. Pour assurer une qualité de service à ce type d'applications, le protocole fournit des mécanismes pour reconstituer la base de temps des différents flux multimédia, détecter les pertes de paquets ou encore identifier le contenu des paquets pour leur transmission sécurisée. Le protocole complémentaire RTCP[26] [Hui03] accompagne RTP en assurant un mécanisme de contrôle pour l'émetteur sur la qualité de transmission RTP ainsi que d'autres informations. RTP et RTCP reposent, en général, sur le protocole de transport UDP et peuvent ainsi être utilisés aussi bien selon les modes unicast et multicast. Cependant, RTP et RTCP n'ont pas été conçus pour effectuer des réservations de ressources et ne garantissent pas non plus la fiabilité de la livraison ni un délai de livraison.

B.4 Les protocoles de synchronisation

B.4.1 Nécessité d'établir une référence de temps globale

En l'absence de référence absolue, chaque noeud du système distribué est équipé de sa propre horloge locale physique. Les horloges de chaque noeud du système distribué ne sont généralement pas les mêmes et peuvent ainsi diverger les unes par rapport aux autres. L'observation de l'écoulement du temps par un ensemble d'horloges logiques nécessitent alors d'établir une référence : une horloge globale. Généralement on utilise alors une technique de synchronisation permettant d'établir une référence de temps commune à tous les nœuds du système. Deux types d'algorithmes peuvent alors être utilisés [KO87, Cri89] :

1. une synchronisation interne consiste à synchroniser mutuellement l'ensemble des horloges locales.
2. une synchronisation externe est fondée sur une horloge référence qui peut être accessible par tous les nœuds du système (exemple l'horloge GPS utilisée dans les simulations DIS). Chacune des horloges locales doit ainsi réaliser une phase de synchronisation interne par rapport à cette horloge globale.

Quelles que soient les techniques utilisées pour mettre en œuvre une synchronisation dans un système distribué, deux points restent essentiels :

1. la précision de la synchronisation ;
2. le déterminisme c'est-à-dire la garantie d'un majorant pour la précision de cette synchronisation.

Dans la suite, nous évoquerons deux protocoles permettant une synchronisation externe (à partir d'un horloge de référence) des horloges locales sur des systèmes distribués et connectés par internet : le protocole NTP (*C.f.* partie B.4.2) et le protocole PTP (*C.f.* partie B.4.3).

[24] *Type Of Service*
[25] *Real-time Transport Protocol*
[26] *Real-time Tranport Control Protocol*

B.4.2 Le protocole NTP

Le protocole NTP[27] [43] est un protocole de synchronisation particulièrement répandu basé sur l'adressage IP en mode non connecté avec UDP. Il permet de synchroniser l'horloge d'un ordinateur avec celle d'un serveur de référence. Depuis la version originale datant de 1985 [Mil85], plusieurs versions ont vu le jour ensuite. La version 2 a introduit un aspect sécuritaire permettant de crypter les paquets de communication grâce à un algorithme de chiffrage spécifique. Les performances de cette version ont d'ailleurs étaient testées sur des simulations DIS d'environnements virtuels [Mil90]. La version 3 [Mil92] a permis d'affiner les algorithmes et de supprimer les ambiguïtés des versions précédentes permettant ainsi une synchronisation de l'ordre de la dizaine de millisecondes ou mieux sur des réseaux locaux et avec des écarts inférieurs à 10 secondes sur des réseaux plus vastes (de type WAN) [Mil94]. La toute nouvelle version 4 [Mil10] devrait permettre d'atteindre une précision de l'ordre de la micro-seconde sur les réseaux locaux modernes et permettrait alors de fournir une horloge globale dont la précision serait compatible avec les contraintes temporelles des systèmes embarqués. Toutes les versions de NTP ont été principalement testées sur le réseau *DCnet Research Network* [14].

B.4.3 Le protocole PTP

Le protocole PTP[28] est un protocole défini dans la norme IEEE 1588 [IEE02]. L'emploi grandissant de réseaux Ethernet dans des applications industrielles a contribué à l'élaboration de ce protocole. PTP est basé sur une architecture logique de type maître-esclave. Cette architecture est automatiquement instanciée par le protocole. Le maître envoie périodiquement un message contenant son heure locale. Les processeurs esclaves en usent pour se synchroniser. Afin de limiter l'impact de pile de communication OSI et du système logicielle, l'horodatage des messages est entrepris généralement au niveau du support matériel. Avec cette conception, PTP atteint des précisions de synchronisation de l'ordre de la microseconde [Cal09]. Ce protocole serait compatible pour fournir une horloge globale entre des fédérés selon le mode d'exécution *Data Flow* avec les contraintes temporelles très petites (simulation de systèmes embarqués). Nous pouvons citer une implémentation Open source de ce protocole *PTPd* dont les détails de l'implémentation sont décrits dans l'article [CBB05] et les sources sont téléchargeables sur le site [53]. Ou encore, le projet *White Rabbit* [80] est un effort multi-laboratoire pour construire un réseau fiable et sûr basé sur Ethernet fournissant des transferts de données déterministes ainsi qu'un mécanisme de synchronisation précis des horloges basé sur PTP. Le lecteur intéressé pourra se référer au manuscrit de Master de Tomasz Wlostowski [Wlo11] pour plus de détails techniques sur ce réseau.

[27] *Network Time Protocol*
[28] *Precision Time Protocol*

Annexe C

La théorie de l'ordonnancement

Le comportement temporel exact et sûr d'une application temps réel repose sur les techniques formelles utilisées pour le valider. La maîtrise du comportement temporel passe par une théorie particulière appelée la théorie de l'ordonnancement qui consiste à organiser l'exécution de différentes tâches concurrentes sur un système donné en fonction de leurs contraintes temporelles respectives (délais, contraintes d'enchaînement). Pour cela, il faut prendre en compte les ressources mises en jeu dans le système pour l'exécution de l'application globale (principalement les processeurs et le réseau). Cette annexe présente donc les méthodes et les principes relatifs à la théorie de l'ordonnancement et pour cela, dans un premier temps, nous présenterons les bases formelles de cette théorie. Nous poursuivrons en étudiant, dans un deuxième temps, les différents techniques algorithmiques applicables aux systèmes monoprocesseurs. Dans un troisième temps, nous nous intéresserons au cas plus compliqué des systèmes multiprocesseurs. Ensuite, nous aborderons la problématique très complexe de l'ordonnancement temps réel dans le contexte des système distribué dans lesquels un média de communication est utilisé pour faire communiquer les différentes parties du systèmes. Nous étudierons les différents aspects (déploiement, synchronisation et validation), les solutions algorithmiques envisagées ainsi que leurs limitations. Pour finir, nous présenterons quelques outils logiciels dédiés à l'étude des systèmes temps réel.

Note : *Le but de cette annexe est de présenter de façon informelle la théorie de l'ordonnancement. Le lecteur est invité à consulter les références citées dans cette annexe pour la compréhension détaillée des aspects mathématiques et algorithmiques.*

C.1 Principes de l'ordonnancement

C.1.1 Présentation générale

Les systèmes temps réel ont acquis une importante place dans le développement des systèmes informatiques contemporains [Sta88, Sta92] et recouvrent un spectre d'applications très étendu (contrôle de systèmes de commande, contrôle du trafic aérien, installations informatiques nucléaires...). Un système informatique est qualifié de temps réel si sa correction ou sa validité ne dépend pas uniquement de la justesse des résultats obtenus mais aussi des instants de production de ces résultats. Dans une application informatique, plusieurs fonctionnalités concurrentes se partagent les ressources (processeurs, mémoires ou encore réseau) ; l'objectif du temps réel est de tenir compte des besoins d'urgence et d'importance de ces traitements en concurrence sur le système. Par conséquent, l'attribution des ressources nécessaires pour l'exécution de toutes les

fonctionnalités d'une application doit être planifiée pour garantir le respect de leurs contraintes temporelles respectives. C'est le rôle des mécanismes d'ordonnancement temps réel qui doivent garantir l'exécution temporellement correcte d'un ensemble de tâches sur un ensemble de ressources rattachées à un système donné.

A l'origine, les systèmes informatiques temps réel étaient équipés d'un seul processeur chargé de réaliser tous les traitements des applications. Ces applications sont devenues, au fil du temps, de plus en plus complexes nécessitant une grande puissance de calcul. Cependant, l'augmentation de la puissance d'un unique processeur est limitée et coûteuse. Les concepteurs se sont donc orientés vers des architectures parallèles composées de plusieurs processeurs pour satisfaire les échéances des applications temps réel complexes. De telles architectures proposent une puissance de calcul totale équivalente et moins chère que les architectures monoprocesseurs très coûteuses et volumineuses. De plus, l'émergence des techniques du réseau a aussi permis aux concepteurs de systèmes temps réel de s'orienter vers les technologies des systèmes distribués dans lesquels plusieurs nœuds (des systèmes monoprocesseurs ou multiprocesseurs) interagissent au travers d'un réseau de communication pour accomplir un calcul global. Par exemple, les systèmes embarqués s'appuient souvent sur des architectures distribuées où chaque processeur est spécialisé. Pour résumer, l'architecture d'un système temps réel est un critère important dans le choix de la stratégie d'ordonnancement et la détermination de sa validité. Trois types d'architecture informatique sont possibles :

- **Les systèmes monoprocesseurs :** toutes les tâches doivent s'exécuter sur un unique processeur. Les techniques pour assurer le comportement temporel de ce type de système seront décrites dans la partie C.2 page 181.
- **Les systèmes multiprocesseurs :** les tâches sont réparties sur plusieurs processeurs partageant une mémoire commune ainsi qu'une horloge matérielle. Les techniques pour assurer le comportement temporel de ce type de système seront présentées dans la partie C.3 page 188.
- **Les systèmes distribués :** les tâches sont réparties sur plusieurs nœuds (monoprocesseurs ou multiprocesseurs) dépourvus de mémoire et d'horloge commune. Ils sont reliés par un médium de communication par lequel ils peuvent communiquer par échange de messages. Les techniques pour assurer le comportement temporel de ce type de système seront décrites dans la partie C.4 page 193.

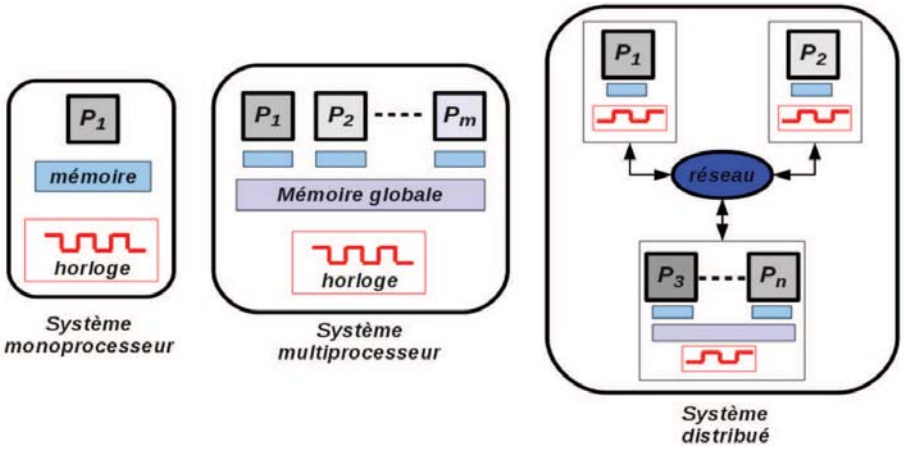


FIG. C.1 – Les différentes architectures des systèmes temps réel

C.1.2 La notion de tâche

Les différents traitements d'une application temps réel sont appelés des tâches. Une tâche est une séquence d'instructions à exécuter sur le système qui permet de réaliser les différentes fonctionnalités d'une application ; c'est l'unité à prendre en considération dans un problème d'ordonnancement. En pratique, une tâche peut être assimilée à deux entités informatiques qu'il faut distinguer : (1) le processus ou (2) le thread (*C.f.* figure C.2) :

1. Un **processus** (dit processus lourd) est un flot d'exécution séquentiel unique qui ne peut pas partager sa mémoire et ses données[1]. Chaque processus lourd possède son propre espace d'adressage virtuel et ses propres variables. Par conséquent, les commutations de contexte[2] sont coûteuses en temps à cause des transferts de mémoire qu'elles engendrent.
2. Les **threads** sont nés avec l'essor des machines parallèles et sont des flots d'exécution internes à un processus. Ainsi l'exécution du code du processus peut être réalisé par différents fils d'exécution distincts. Un thread est définit dans le contexte d'exécution d'un processus (avec les données fournies par ce processus). Un processus lourd peut contenir plusieurs centaines de processus légers constituant autant de flots d'exécution parallèles indépendants. Les threads proposent une solution pour éviter les commutations de contexte coûteuses aux processus. En revanche, tous les threads d'un même processus partagent les mêmes zones mémoire ce qui peut amener à des conflits lors de l'utilisation concurrente de ces ressources (*C.f.* partie C.2.4, page 185).

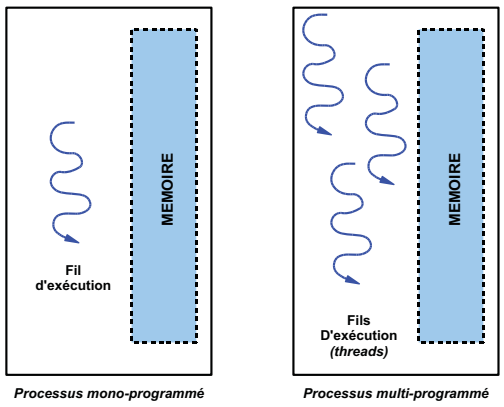


FIG. C.2 – Une tâche informatique : processus ou thread

C.1.3 Comportement temporel et description analytique des tâches

Les tâches mises en concurrence sur un système informatique temps réel n'ont pas toutes le même comportement temporel. En effet, certaines tâches sont dédiées à la réalisation de calculs itératifs et donc répétitifs du point de vue temporel. D'autres sont dédiées à des traitements particuliers et ponctuels. Il est possible de classer les tâches mises en jeu dans un système temps réel en trois catégories (*C.f.* figure C.3) :

[1] *Sauf si le concepteur le définit explicitement en utilisant un segment de mémoire partagée*
[2] *L'arrêt de l'exécution d'un processus sur le processeur pour en exécuter un autre*

1. les tâches périodiques ont des activations régulières et le délai entre deux activations successives est connu et constant.
2. les tâches sporadiques ont des activations irrégulières mais on connait le délai minimum entre deux activations successives (appelé le *délai inter-arrivée* et noté DI).
3. les tâches apériodiques sont des tâches sur lesquelles nous n'avons aucune information.

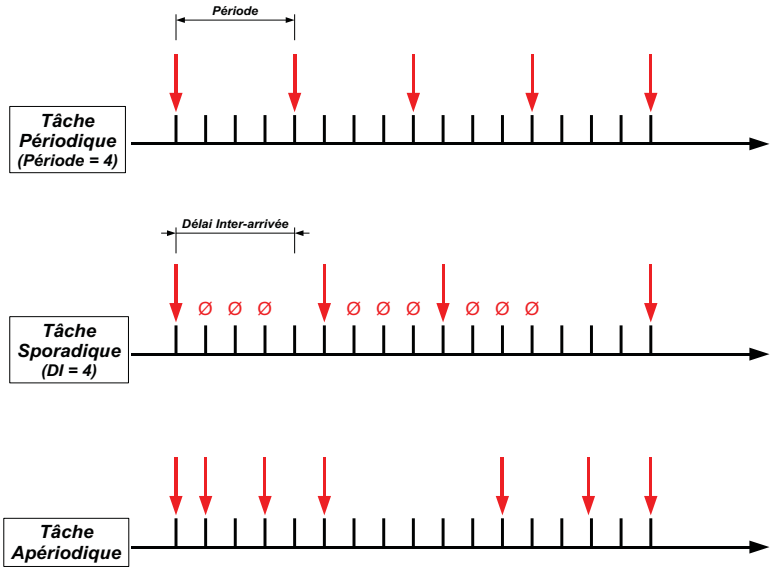


FIG. C.3 – Les différents comportements temporels possibles pour une tâche

La définition des modèles de tâches permet de décider de la politique d'ordonnancement à mettre en place sur le système. Pour analyser de manière rigoureuse la séquence d'exécution des tâches par une politique d'ordonnancent, il est nécessaire d'avoir un modèle mathématique précis permettant de décrire ces tâches. Ce modèle doit permettre de prendre en compte les caractéristiques opérationnelles et temporelles de chaque tâche. Les algorithmes d'ordonnancement sont donc extrêmement dépendants du modèle analytique choisi. Le modèle des tâches périodiques est le plus utilisé car c'est celui qui assure le comportement le plus prédictible. La plupart des techniques pour la prise en compte des tâches sporadiques et apériodiques visent à se ramener à des extensions des méthodes pour les tâches périodiques (*C.f.* partie C.2.3, page 184). Serlin [Ser71] et Liu et Layland [LL73] ont proposé un modèle de tâche périodique basé sur les attributs temporels de ces tâches. Depuis, ce modèle a été étendu mais, de façon générale, une tâche périodique τ_i est décrite par un simple quadruplet $< r_i, C_i, D_i, T_i >$ (*C.f.* figure C.4) :

- r_i est la **date d'activation** de la tâche qui correspond à l'instant à partir duquel la tâche peut commencer son exécution ;
- C_i est la **durée d'exécution** de la tâche qui est égale à la borne supérieure du temps processeur nécessaire à l'exécution de la tâche ;
- D_i est le **délai critique** à respecter pour la tâche qui est l'intervalle de temps, à partir de la date de réveil, durant lequel la tâche doit s'exécuter ; en dehors de cet intervalle, l'exécution de la tâche devient non valable ;
- T_i est la **période** de la tâche qui correspond à l'intervalle de temps fixe qui sépare les arrivées successives d'une tâche. Une **instance** d'une tâche périodique est une de ces itérations périodiques.

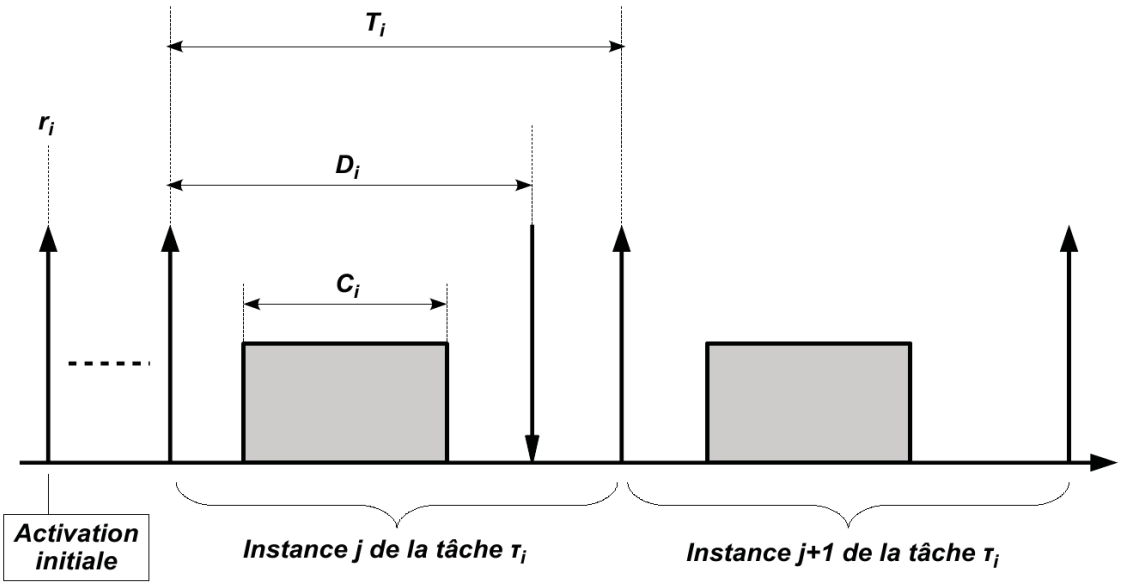


FIG. C.4 – Illustration graphique du modèle de tâche périodique τ_i

La durée d'exécution C_i de la tâche correspond au temps processeur maximum (au pire des cas) nécessaire pour que la tâche puisse réaliser son traitement. Ce temps peut donc varier en fonction de l'architecture support sur laquelle la tâche va être exécutée. En général, la tâche est caractérisée par son pire temps d'exécution WCET[3] pour un processeur donné, c'est-à-dire un majorant de tous les temps d'exécution possibles de la tâche seule sur ce processeur. L'estimation du WCET se trouve confrontée à de nombreux problèmes dus entre autres aux technologies déployées sur les nouveaux processeurs. Par exemple, avec l'utilisation des pipelines super-scalaires, l'exécution des instructions de chaque tâche peut s'effectuer dans le désordre. Ou encore, sur les systèmes actuels à plusieurs cœurs partageant une mémoire cache (généralement L2 ou L3), les processeurs peuvent passer leur temps en conflit pour accéder à cette mémoire cache. La détermination du WCET est donc essentielle et constitue un domaine de recherche à part entière [PK89, PS97, WEE+08, CPRS02].

C.1.4 La priorité et la préemption

Comme expliqué précédemment, tous les algorithmes d'ordonnancement fonctionnent à partir de la notion de **priorité**. Cette priorité, notée $\Pi(\tau_i)$ pour une tâche τ_i, permet d'établir une relation d'urgence entre les tâches. La plupart des politiques d'ordonnancement sont basées sur ce principe de priorité pour gérer la concurrence entre les tâches d'un système temps réel. C'est-à-dire qu'à chacune des tâches (ou des instances de ces tâches) une priorité est associée et le processeur est toujours attribué à la tâche de plus haute priorité. La valeur de cette priorité peut être constante au cours du temps et fixée avant l'exécution du système[4] (*i.e.* hors-ligne) ou dynamique et peut évoluer constamment durant l'exécution du système[5] (*i.e.* en-ligne). Ces types de priorité permettent de distinguer les classes de politiques d'ordonnancement statiques et dynamiques (*C.f.* partie C.1.6, page 179).

[3] *Worst-Case Execution Time*
[4] *La priorité est alors liée à la tâche.*
[5] *Généralement, la priorité est alors liée à l'instance de la tâche c'est-à-dire une de ces exécutions périodiques.*

La préemption est la capacité du système informatique d'interrompre une tâche en cours en faveur d'une tâche de priorité supérieure. Chaque élection d'une nouvelle tâche s'accompagne d'un changement de contexte, permettant de passer de l'exécution de l'ancienne tâche (en cours et pas encore terminée) à la nouvelle plus prioritaire. La préemption peut être avantageuse mais elle amène des délais de changement de contexte qu'il faut prendre en compte lors de la conception [Meu09]. La prise en compte ou non de la préemption permet de distinguer les classes de politiques d'ordonnancement préemptives et non-préemptives (*C.f.* partie C.1.6, page 179).

Les comportements préemptif et non-preemtif sont illustrés respectivement sur les figures C.5 et C.6. Dans ces deux illustrations, deux tâches τ_1 et τ_2 sont en concurrence pour l'obtention du processeur et τ_1 est plus prioritaire que τ_2.

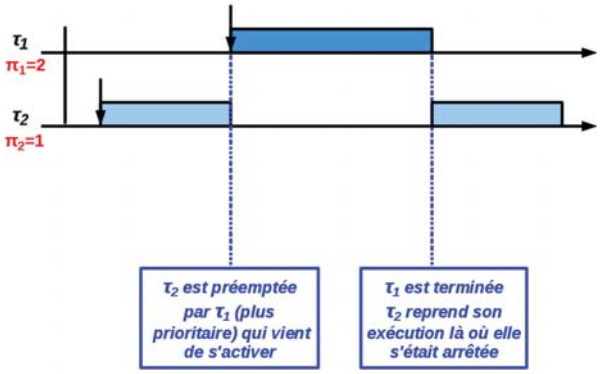


FIG. C.5 – Comportement préemptif

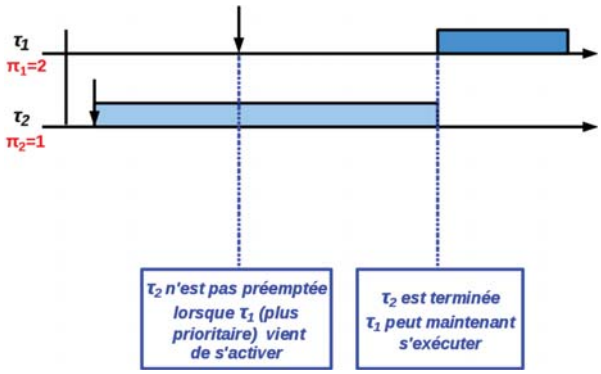


FIG. C.6 – Comportement non-préemptif

C.1.5 Les politiques d'ordonnancement et la complexité

En général, un algorithme d'ordonnancement est nécessaire pour pouvoir garantir l'ordonnançabilité d'un système donné. Un algorithme d'ordonnancement est un algorithme qui utilise les informations sur l'ensemble des tâches (le modèle analytique) et détermine la séquence d'ordonnancement propice au problème. Pour cela, le concepteur fixe des règles qui permettent à l'ordonnanceur de choisir à un instant donné à quelle tâche prête il accorde la ressource. L'ensemble de ces mécanismes est appelé politique d'ordonnancement. Pour un système donné et une politique d'ordonnancement particulière, l'ordonnanceur produit au cours du fonctionnement de l'application, une séquence d'exécution des instances des tâches. Le but du concepteur est de fournir une politique d'ordonnancement qui garantit le respect des contraintes temporelles. Plusieurs définitions servent à qualifier une politique d'ordonnancement vis-à-vis d'un système :

- La politique d'ordonnancement d'un ensemble de tâches est dite **valide** si et seulement si toutes les tâches du système respectent leurs contraintes temporelles (contrainte de périodicité, contrainte d'échéance, ...) ;
- Un ensemble de tâches est dit **faisable** (donc ordonnançable) si et seulement si il existe une politique d'ordonnancement valide pour le système considéré ;
- Une **classe** de politiques d'ordonnancement regroupe les politiques d'ordonnancement qui ont les mêmes caractéristiques (*C.f.* partie C.1.6, page 179) ;
- Une politique d'ordonnancement est **optimale** pour une classe de politiques d'ordonnancement donnée et une classe de systèmes donnée si et seulement si tous les systèmes de

la classe qui sont ordonnançables par une politique d'ordonnancement de la classe donnée, sont ordonnançables par la politique d'ordonnancement optimale. Par conséquent, si un système n'est pas ordonnançable par une politique d'ordonnancement optimale d'une classe donnée, alors il ne l'est par aucune autre politique de la même classe. L'optimalité est un critère de comparaison entre les politiques d'ordonnancement.

Pour résoudre un problème d'ordonnancement, il faut déterminer une méthode algorithmique permettant de vérifier que l'ensemble de tâches considéré est faisable sur le système. Un problème d'ordonnancement est donc aussi caractérisé par la **complexité** des méthodes permettant de vérifier qu'il existe un ordonnancement faisable (et/ou optimal) sur le système. La complexité d'un problème est la quantité de temps et d'espace (mémoire) qui est nécessaire à la recherche de la solution du problème posé. Nous savons qu'un bon nombre de problèmes classiques sont solvables algorithmiquement (lorsque cet algorithme existe). Par exemple, un problème peut être résolu par énumération de toutes les solutions possibles, mais en revanche le temps ou l'espace nécessaire à la mise en œuvre de certains algorithmes rendent leur utilisation non réaliste. Généralement, la complexité relative à un problème d'ordonnancement est exprimée par rapport à la dimension du modèle c'est-à-dire par rapport au nombre de tâches à ordonnancer et à l'architecture du système (par exemple, le nombre des processeurs). Différentes solutions sont possibles pour étudier l'ordonnançabilité d'un système temps réel :

- L'approche par **simulation** consiste à modéliser le comportement du système à l'aide d'un formalisme adéquat puis à l'exécuter avec un outil de simulation sur des scénarios définis par l'utilisateur. Cette approche peut permettre de prouver qu'un système n'est pas ordonnançable en montrant l'existence d'un scénario non ordonnançable. Elle permet aussi de montrer qu'il existe un scénario faisable mais ne peut pas prouver que le système est ordonnançable (même si tous les scénarios testés le sont). De plus, il est généralement assez difficile de réaliser une étude exhaustive de tous les scénarios possibles.
- L'analyse par **model-checking** est une méthode permettant l'exploration exhaustive de l'espace d'états du système. Le formalisme utilisé est souvent celui des systèmes de transition étendus avec la notion de temps tels que les automates temporisés ou les réseaux de Petri temporisés. Dans cette approche, le principal problème est l'espace d'états possibles à prendre en compte qui peut être de taille trop importante ceci limitant, par conséquent, la taille des modèles analysables.
- L'étude par **approche analytique** consiste à produire une expression analytique, une condition de faisabilité (*i.e.* test de faisabilité), permettant de garantir l'ordonnançabilité du système. Ces conditions peuvent être des conditions suffisantes (CS) ou des conditions nécessaires et suffisantes (CNS), qui sont établies à partir des caractéristiques temporelles des tâches et sont généralement associées à l'algorithme utilisé pour affecter les priorités à ces tâches (*C.f.* partie C.2.2, page 182). Par la suite, nous nous intéresserons principalement à ce type de technique.

C.1.6 Les classes d'algorithmes

Généralement, on distingue plusieurs classes de politiques d'ordonnancement : (1) Statiques *vs* Dynamiques, (2) Préemptifs *vs* Non-préemptifs et (3) Oisif *vs* Non-Oisif :

1. **Statiques *vs* Dynamiques** : Ces caractéristiques correspondent au moment où sont effectués les choix d'allocation de la priorité pour chaque tâche. Dans le cas **statique**, les priorités des tâches sont calculées une seule fois avant l'exécution du système et sont basées sur des paramètres qui ne changent pas pendant l'exécution du système. L'avantage

de cette classe d'algorithmes est la possibilité d'identifier, avant l'exécution, les tâches qui dépasseront leurs échéances. En revanche, l'ordonnancement statique prend difficilement en considération les évolutions du système qui peuvent survenir (par exemple l'ajout d'une tâche). Si le système évolue, il est alors nécessaire de reconstruire l'ensemble du cycle d'exécution pour pouvoir obtenir un nouvel ordonnancement valide. Dans le cas **dynamique**, les priorités des tâches sont calculées plusieurs fois pendant l'exécution du système et sont basées sur des paramètres qui peuvent varier pendant l'exécution. En général, malgré les importants surcoûts liés à l'exécution même de l'algorithme d'ordonnancement pendant l'exécution du système, l'avantage de cette classe d'algorithmes est la possibilité de prendre en compte des événements susceptibles de se produire de manière apériodique. Ce type d'approche permet au système de s'adapter et de prendre en compte des tâches non prévues lors de la spécification. Néanmoins, en pratique, dans le cadre d'applications temps réel dur telles que les systèmes embarqués, les algorithmes dynamiques sont souvent boudés car ils sont difficilement prédictibles et moins contrôlables. De plus, ils sont très sensibles aux *effets domino* ; si on surcharge le système dans le cas dynamique, toutes les tâches violent leur échéance dans une plage de temps très réduite. Alors que, dans le cas statique, seules les tâches ayant les priorités les plus faibles violeraient leur échéance.

2. **Préemptifs *vs* Non-préemptifs :** Conformément à la définition vue dans la partie C.1.4, dans le cas préemptif, une tâche en cours d'exécution peut être interrompue pour céder le processeur à une autre tâche de priorité supérieure sans avoir à recommencer son exécution depuis le début. En général, malgré la difficulté d'évaluer le coût lié à la préemption [Meu09], l'avantage dans cette classe d'algorithmes est l'amélioration du temps de réponse des tâches les plus prioritaires du système ainsi que le nombre plus important de systèmes ordonnançables. Dans le cas non préemptif, une fois qu'elle a démarré son exécution, une tâche ne peut pas être interrompue pour céder le processeur à une autre tâche quelle que soit la priorité de celle-ci. En général, malgré le nombre moins important de systèmes ordonnançables, l'avantage de cette classe d'algorithmes est la limitation des surcoûts induits par la préemption : (1) coût du changement de contexte entre les tâches ainsi que (2) le coût de l'ordonnanceur demandant ce changement de contexte.
3. **Oisifs *vs* Non-Oisifs :** Dans le cas d'un algorithme d'ordonnancement non-oisif, c'est-à-dire avec insertion de temps creux, dès qu'au moins une tâche est prête à être exécutée, alors l'ordonnanceur en élit forcément une parmi elles. Dans ce cas, on dit aussi que l'ordonnanceur fonctionne sans insertion de temps creux. En général, bien qu'il conduise dans le cas non-préemptif à un nombre plus important de systèmes non ordonnançables, l'avantage de cette classe d'algorithmes est qu'on peut atteindre des charges processeur plus élevées. En revanche dans le cas préemptif le nombre de systèmes ordonnançables est plus important.

C.2 Ordonnancement d'un système monoprocesseur

C.2.1 Le rôle de l'ordonnanceur

Si l'on se place à l'échelle d'un système mono-processeur, la fonction d'ordonnancement est gérée par un élément de l'exécutif temps réel (*i.e.* le système d'exploitation) appelé l'ordonnanceur. Plusieurs tâches peuvent réclamer simultanément un processeur afin de s'exécuter. Ces conflits peuvent être résolus par des mécanismes d'affectation de priorité pour les tâches pour l'accès au(x) processeur(s). L' ordonnanceur élira la tâche (active) la plus prioritaire, c'est-à-dire la tâche considérée comme la plus urgente. Les priorités sont déterminées par un algorithme d'ordonnancement. Le but de l'ordonnanceur est de fournir une séquence d'ordonnancement valide pour le respect des contraintes temporelles de chaque tâche. La problématique de l'ordonnancement consiste à définir une politique d'élection adéquate qui garantit le respect des contraintes temporelles pour toutes les tâches mises en concurrence dans le système. Cette politique d'élection peut s'exprimer par une simple politique d'ordonnancement basée sur des priorités ou par un algorithme plus complexe et doit permettre de définir une séquence d'ordonnancement valide. Pour résumer, l'ordonnanceur est chargé de l'organisation temporelle des tâches en concurrence (*C.f* figure C.7).

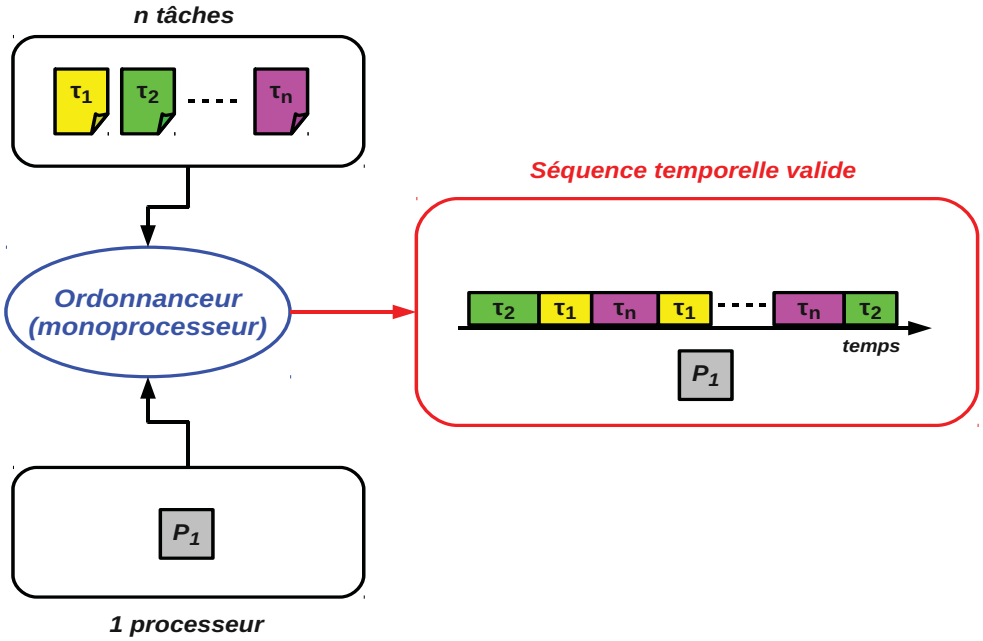


FIG. C.7 – Ordonnanceur monoprocesseur

De plus, plusieurs tâches peuvent demander simultanément l'accès à la même ressource mémoire telle qu'une donnée non partageable (unité d'entrée/sortie, fichiers, disques...). Dans ces cas, l'ordonnanceur met en œuvre des mécanismes de verrouillage (*C.f* partie C.2.4, page 185) mais l'ordonnancement doit prendre en considération les temps de blocage dus à ces conflits de ressource afin d'éviter que certaines tâches qui tardent à avoir accès à cette donnée voient leurs contraintes temporelles violées.

Nous nous plaçons donc ici dans le contexte de l'ordonnancement monoprocesseur. Les algorithmes pour ordonnancer des tâches périodiques et indépendantes seront présentés dans un pre-

mier temps (*C.f* partie C.2.2). Les techniques pour la prise en compte de tâches non-périodiques seront présentées (*C.f* partie C.2.3, page 184). Enfin, nous présenterons les techniques pour gérer les relations de dépendances entres les tâches : (1) le partage des ressources mémoire (*C.f* partie C.2.4, page 185) et (2) les relations de précédence (*C.f* partie C.2.5, page 187).

C.2.2 Les algorithmes classiques

Les algorithmes d'ordonnancement[6] les plus courants sont les suivants :

- **Rate Monotonic (RM)** [LL73, LSD89] : algorithme d'ordonnancement préemptif à priorités fixes. La tâche est d'autant plus prioritaire que sa période est petite. Cependant, si les périodes de plusieurs tâches sont identiques, il faut alors décider d'une politique additionnelle pour distinguer ces tâches.
- **Deadline Monotonic (DM)** [LW82, LSST91] : algorithme d'ordonnancement préemptif à priorités fixes. La tâche est d'autant plus prioritaire que son échéance est courte. DM équivaut à RM lorsque l'échéance relative de la tâche est égale à sa période.
- **Algorithme de Audlsey** [Aud91] : algorithme d'ordonnancement préemptif à priorités fixes combiné à un test de faisabilité pendant les calculs d'affectation des priorités. C'est le seul algorithme statique optimal lorsque les tâches ont des dates de réveil différentes de zéro.
- **Earliest Deadline First (EDF)** [LL73, Der74] : ordonnancement préemptif à priorités dynamiques. Une instance de tâche est d'autant plus prioritaire que sa date d'échéance est proche de la date courante. Cette politique est similaire à celle proposée par Jackson [Jac55] issue de l'ordonnancement pour les chaînes de production et appelée **EDD**[7]. Cet algorithme concernait des systèmes de tâches indépendantes, apériodiques et exécutées une seule fois.
- **Least Laxity First (LLF)** [Sor74, Mok83] : ordonnancement préemptif à priorités dynamiques. Une instance de tâche est d'autant plus prioritaire que sa laxité[8] est petite. En pratique, il est complexe à mettre en œuvre car il nécessite de connaître à tout instant la durée d'exécution restante pour chacune des tâches actives. De plus, LLF conduit à plus de préemptions que l'algorithme EDF. Si deux activations de tâches ont la même échéance absolue, alors elles se préempteront sans arrêt jusqu'à la fin de leurs exécutions avec LLF ; alors qu'avec EDF, une des deux sera exécutée d'abord sans préemption. Ce nombre important de préemptions explique qu'il soit en général très peu utilisé par rapport à EDF [Bim07]. Oh et Yang [OY98] ont proposé une variante de LLF (notée MLLF[9]) qui autorise les inversions de laxité entre les tâches afin de réduire le nombre excessif des préemptions. Une étude comparative des algorithmes LLF, MLLF et EDF est disponible dans [SNT+06]

Il existe des variantes non-préemptives des algorithmes ci-dessus. La littérature est beaucoup moins importante et les techniques s'appliquent principalement pour séquencer les messages sur un réseau informatique. **NP-EDF**[10] [KN80, JSM91, ZS94] est une variante non-préemptive de l'algorithme EDF dédiée à gérer l'accès dans un réseau. Francisco Vasques de

[6] *Rappel : Ces algorithmes sont tous non-oisifs.*
[7] *Earliest Due Date*
[8] *La laxité est l'échéance absolue moins le temps courant et moins la durée d'exécution restante*
[9] *Modified LLF*
[10] *Non-Preemptive Earliest Deadline First*

Carvalho [VdC96] utilise aussi cette version de EDF et il propose en plus une variante non-préemptive de l'algorithme RM (notée NP-RM[11]) en basant le test d'ordonnancabilité sur une adaptation du protocole de gestion de ressource à priorité héritée (*C.f.* partie C.2.4, page 185). **CAN-DM**[12] [THW94, TBW95, DBBL07] est une variante non-préemptive de l'algorithme DM proposées pour ordonnancer les flux de messages sur le réseau CAN où l'attribution du médium de communication se fait par affectation de priorité. Bien que LLF implique, lors de son utilisation, un grand nombre de préemptions entre les tâches, une variante non-préemptive a été proposée dans [LK99] pour ordonnancer les messages à contraintes souples sur le réseau CAN. Le lecteur intéressé peut se référer au manuscrit de thèse de Nicolas Rivierre [Riv98] qui présente une étude approfondie et comparative des algorithmes préemptifs et non-préemptifs ainsi que différentes études de faisabilité associées.

Afin de vérifier que les tâches respecteront leurs échéances temporelles, ces algorithmes d'affectation de faisabilité sont donc associés avec des tests de faisabilité (*i.e* condition d'ordonnancabilité). Ces tests reposent principalement sur trois principes :

- L'étude du **facteur d'utilisation processeur** (PUA[13]) : le facteur d'utilisation est le pourcentage de temps que le processeur passe à exécuter les tâches du système. Il est calculé par la formule : $U_{\mathcal{T}} = \sum_{i=0}^{n} \frac{C_i}{P_i}$ pour un ensemble de tâches $\mathcal{T} = [\tau_1, \tau_2, \ldots, \tau_n]$. Dans certains cas, l'étude de ce facteur d'utilisation permet de calculer des conditions d'ordonnançabilité. Mais, dans tous les cas, si le processeur a un facteur d'utilisation supérieur à 1, le système ne sera pas ordonnançable. Cette approche est souvent appréciée pour la simplicité des tests de faisabilité associés et parce qu'elle donne une réponse sur l'ordonnançabilité globale du système.
- L'analyse de la **demande processeur** (PDA[14]) : cette technique repose sur le calcul de la demande cumulée des exécutions des tâches activées dans un intervalle de temps d'étude. L'analyse de la demande processeur revient à tester pour tout intervalle de temps $[t_1, t_2]$ que la durée maximale cumulée (ou borne supérieure) des exécutions des requêtes qui ont leur réveil et leur échéance dans l'intervalle est inférieure à $t_2 - t_1$ (c-à-d., n'excède pas la longueur de l'intervalle). Cette approche permet aussi de donner une réponse sur l'ordonnançabilité globale du système.
- L'analyse des **temps de réponse** (RTA[15]) : cette analyse est basée sur le calcul des temps de réponse des tâches afin d'établir qu'ils sont inférieurs à leurs échéances respectives. Le temps de réponse d'une tâche est établi en fonction du temps nécessaire pour terminer son exécution en considérant les latences dues aux exécutions des tâches plus prioritaires ainsi que des diverses causes de blocage (ressources, précédences,...). Cette approche est très utilisée dans les systèmes monoprocesseurs mais aussi dans la validation de systèmes distribués (*C.f* partie C.4.5, page 199). L'analyse du temps de réponse doit être appliquée à chacune des tâches du système.

Le tableau C.1 résume les différentes références[16] sur l'analyse de faisabilité en fonction des différents algorithmes vus précédemment. Nous rappelons qu'une condition suffisante est notée CS et qu'une condition nécessaire et suffisante est notée CNS.

[11] *Non-Preemptive Rate Monotonic*
[12] *Controller Area Network Deadline Monotonic*
[13] *Processor Utilization Analysis*
[14] *Processor Demand Analysis*
[15] *Response Time Analysis*
[16] *De façon non exhaustive*

Nom de l'algorithme	Facteur utilisation	Demande Processeur	Temps de réponse
RM	[LL73] (CS)	[LSD89] (CNS)	[JP86] (CNS)
DM	[LW82] (CS)	[LSD89] (CNS)	[JP86, LSST91] (CNS)
Audlsey	X	[Aud91] (CNS)	X
EDF	[LL73, Der74] (CNS)	[BMR90] (CNS)	[Spu96] (CNS)
LLF	[Mok83] (CNS équiv. EDF)	X	X
NP-RM	X	X	[VdC96] (CNS)
NP-EDF	X	[KN80, JSM91, ZS94] (CNS)	X
CAN-DM	X	X	[THW94, DBBL07] (CNS)

TAB. C.1 – Résumé des conditions d'ordonnançabilité

C.2.3 La prise en compte de tâches non périodiques

Sachant que les algorithmes d'ordonnancement s'appliquent principalement aux configurations de tâches périodiques, le problème est d'ordonnancer des tâches apériodiques sans remettre en cause l'ordonnançabilité des tâches périodiques. Dans un système temps réel, les tâches apériodiques peuvent être prises en compte en utilisant deux types de techniques :

1. les techniques de **réquisition des temps creux**[17] : ces techniques consistent à utiliser les temps pendant lesquels le processeur ne traite pas les tâches périodiques pour les consacrer aux traitements des tâches apériodiques. Généralement, il faut maintenir un calendrier dans la mémoire du système, établi hors-ligne et recouvrant une hyperpériode du système[18]. À chaque instant, le calendrier indique la capacité d'exécution du processeur autorisé pour les tâches apériodiques tout en garantissant que les échéances des tâches périodiques restent respectées. L'objectif de ces méthodes est d'améliorer les temps de réponse des tâches apériodiques sans déranger le séquencement des tâches périodiques. Pour optimiser le temps de réponse des tâches apériodiques, l'algorithme peut se baser sur des méthodes de recherche gloutonnes[19] [LRT92, DTB93, LT95] ou non-gloutonnes [TLS96]. Un résumé incluant les formules et les algorithmes peut être trouvé dans le document de Johan Hägg [Hä08]. Toutefois, cette approche n'est pas optimale pour les tâches apériodiques dures car il est impossible de garantir que le temps de réponse de toutes les tâches apériodiques sera inférieur à leurs échéances respectives. Les méthodes de réquisition des temps creux sont des approches hors-ligne dédiés à des politiques d'ordonnancement statiques. Tia et al [TLSH94] ont proposé une variante de ce type de méthodes pour un ordonnancement dynamique avec EDF ainsi que des tests de faisabilité associé.
2. les techniques de **tâche serveur :** Le serveur est une tâche périodique spécifique qui est créée pour ordonnancer les tâches apériodiques. Il s'agit de créer un processus serveur dédié à l'ordonnancement des tâches apériodiques qui surviendront pendant l'exécution. Il existe plusieurs types de serveurs qui diffèrent par la manière dont ils répondent aux arrivées des tâches apériodiques :
   - le **serveur de traitement par scrutation** (*i.e.* Polling Server PS) : ce type de serveur est conçu comme une tâche périodique quelconque. Une fois ce serveur lancé, s'il existe des tâches apériodiques en attente, elles sont exécutées dans la limite de la période du

[17] *Slack Stealing*
[18] *L'intervalle d'étude*
[19] *Un algorithme glouton est un algorithme qui suit le principe de faire, étape par étape, un choix optimum local, dans l'espoir d'obtenir un résultat optimum global.*

serveur. Le cas échéant, le serveur se suspend jusqu'à sa prochaine occurrence périodique. L'introduction du serveur ne modifie par les conditions d'ordonnançabilité du système car le serveur est une tâche périodique qui est ordonnancée même s'il n'existe pas de tâche apériodiques à traiter. En revanche, les tâches apériodiques peuvent ne pas être traitées directement lorsqu'elles sont activées.

- le **serveur ajournable** (*i.e.* Deferable Server DS) [LSS87, SLS95] : l'idée est similaire à la précédente mais le serveur n'est activé que lorsqu'il existe une tâche apériodique à traiter. Cela augmente la réactivité du traitement pour la tâche apériodique. En revanche, cette technique perturbe l'ordonnancement du système ce qui peut modifier les conditions de faisabilité selon les cas (le taux d'utilisation processeur supporté est pessimiste).
- le **serveur sporadique** (*i.e.* Sporadic Server SS) [SSL89, Spr90] : Cette technique est plus simple que le serveur DS et ne modifie pas les tests de faisabilité. Le serveur est vu comme une tâche sporadique dont la seule contrainte est d'assurer que les instants de remplissage de la capacité du serveur sont séparés par un délai d'inter-arrivée.
- le **serveur par échange de priorité** (*i.e.* Priority Exchange Server PES) [LSS87, SLS88] : ce type de serveur est une variante de SS dans lequel on attribue à la tâche affectée au serveur une priorité plus élevée que celles de toutes les tâches périodiques. Ainsi, le serveur prend la main après chaque exécution d'une tâche périodique et, dans le cas où une tâche apériodique survient, elle peut ainsi être exécutée immédiatement. Le cas échéant, le serveur échange sa priorité avec la tâche périodique de plus haute priorité. Le serveur va reprendre son temps et la plus haute priorité une fois cette tâche périodique achevée. Les conditions de faisabilité sont complexes lorsque ce type de serveur est utilisé sur le système.

Ces techniques de serveurs de tâches apériodiques sont principalement utilisées dans le cas des algorithmes d'ordonnancement statiques. Certaines techniques telles que les techniques d'échange de priorité ou de serveur sporadique sont aussi adaptées pour définir un comportement de serveur compatible avec un ordonnancement par EDF. Certaines autres méthodes de serveurs sont spécialement dédiées aux algorithmes dynamiques comme les serveurs à bande passante donnée [SB94]. La condition d'ordonnançabilité revient toujours à considérer le serveur comme une tâche périodique quelconque du système.

C.2.4 La prise en compte du partage des ressources

Une ressource est une structure logicielle pouvant être utilisée par une tâche pour avancer dans son exécution (ex : ensemble de variables, mémoire partagée, fichier etc.). Elle est dite privée lorsqu'elle est dédiée à une tâche en particulier, partagée si elle est utilisée par plusieurs tâches. On définit une section critique comme étant une partie de code exécutée en exclusion mutuelle. Une tâche est dite bloquée si elle attend une ressource exclusive. Lorsqu'elle la prend, elle entre en section critique. Une ressource exclusive est dite libre si aucune tâche n'est dans une section critique utilisant cette ressource. Le partage de ressources entre les tâches du système amène deux problèmes : (1) l'inversion de priorité et (2) l'interblocage.

Le problème d'inversion de priorité (*C.f.* figure C.8 tirée de [Mai96]) résulte de l'occupation d'une ressource par une tâche moins prioritaire, et lorsqu'une tâche de plus haute priorité arrive en demandant la même ressource critique, elle sera bloquée jusqu'à sa libération par la tâche moins prioritaire. Donc, on dit que la tâche de priorité élevée a été victime d'une inversion de priorité.

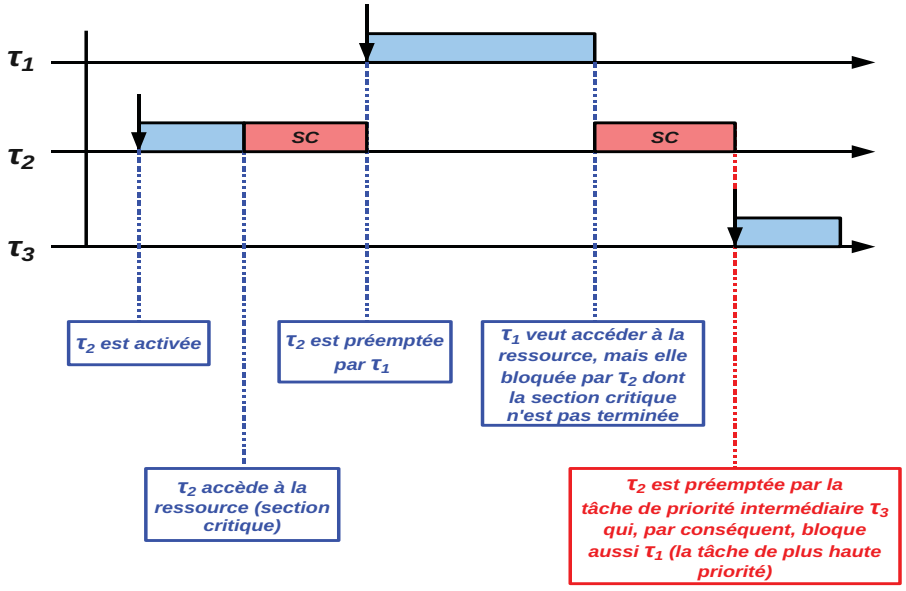


FIG. C.8 – Problème d'inversion de priorité

Le problème d'interblocage peut se produire lorsque toutes les tâches sont bloquées en attente de certaines ressources détenues par d'autres tâches, qui sont elle aussi bloquées. Ce problème est donc dû aux demandes circulaires pour des entrées en section critique (*C.f.* figure C.9 tirée de [Mai96]).

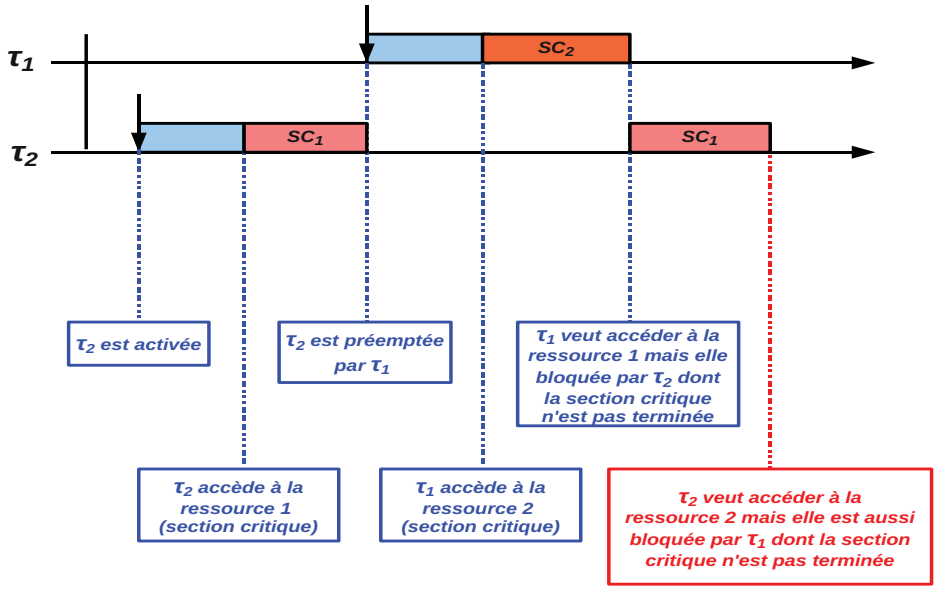


FIG. C.9 – Problème d'interblocage

Pour résoudre ce problème, différents protocoles d'accès aux ressources existent et sont intégrés à l'exécutif temps réel afin de maîtriser le phénomène d'inversion de priorité et de borner le temps de blocage parmi les protocoles d'accès aux ressources en priorités statiques :

– Le protocole à héritage de priorité **PIP**[20] [SRL90] : la tâche ayant la ressource hérite pour la durée de la section critique de la plus grande priorité des tâches qui demandent aussi

[20] *Priority Inheritance Protocol*

cette ressource. Ce protocole permet d'éviter l'inversion de priorité mais pas l'interblocage.

- Le protocole à priorité plafond **PCP**[21] [GS88, SRL90] : ce protocole s'inspire du PIP et permet, en plus, d'éviter l'interblocage. Pour qu'une tâche puisse entrer en section critique, il faut qu'elle soit strictement plus prioritaire que toutes les tâches pouvant avoir accès aux ressources actuellement bloquées. Une version modifiée du protocole PCP permet d'utiliser EDF pour avoir des priorités dynamiques [CL90].
- Le protocole à priorité de pile **SRP**[22] [Bak91] : le principe de base de ce protocole consiste à bloquer les tâches avant qu'elles ne le soient sur une ressource critique dans le but de minimiser le nombre de changements de contexte. Une tâche ne peut préempter une autre tâche que si elle est plus prioritaire que toutes les tâches prêtes (règle normale de l'ordonnancement à priorité) et si sa priorité est strictement supérieure à toutes les priorités des tâches actuellement bloquées (règle du PCP). Ce protocole permet d'éviter l'interblocage et l'inversion de priorité en réduisant légèrement le nombre de préemptions (par rapport aux autres protocoles précédents).

C.2.5 La prise en compte des contraintes de précédences

En général, dans les systèmes embarqués, les tâches ne sont pas indépendantes mais dépendantes dans le sens où une tâche peut produire des données pour une autre tâche qui les consommera pour réaliser ses calculs. Une dépendance impose une relation de précédence entre la tâche qui produit et celle qui consomme et cette contrainte de précédence impose un ordre dans l'exécution de ces tâches. En effet, si l'une a besoin du résultat issu de l'exécution de l'autre pour pouvoir être exécutée à son tour, cela impose, bien évidemment, que la tâche qui produit la donnée soit exécutée avant. On appelle la tâche qui produit les données une tâche *productrice* (*i.e.* prédécesseur) et la tâche qui consomme les données la tâche *consommatrice* (*i.e.* successeur). Ces relations de précédence entre les tâches peuvent être modélisées avec un graphe orienté où les nœuds représentent les tâches et les arcs les relations de dépendance entre les tâches.

L'objectif des techniques de prise en compte des relations de précédence est de transformer un ensemble de tâches avec des relations de précédence en un ensemble de tâches indépendantes. Le but des techniques peut se résumer à une affectation des priorités[23] aux tâches soumises à ces contraintes de précédence de façon à ce qu'une tâche ne s'exécute jamais avant la tâche la précédant. On distingue deux types de précédence : (1) les précédences simples et (2) les précédences complexes :

1. les **précédences simples** sont les contraintes de précédence qui impliquent que la tâche productrice et la tâche consommatrice (et donc le message) aient la même période. Bien que les dépendances entre les tâches d'un système temps réel ne se résument pas à ce type de précédence, ce modèle est le plus utilisé. Blazewicz [Bla76] a développé une technique pour prendre en compte ce type de précédence dans le cas de l'ordonnancement statique (avec DM) et Chetto [CSB90] dans le cas d'un ordonnancement dynamique (avec EDF).
2. les **précédences complexes** sont les contraintes de précédence générales qui impliquent que la tâche productrice et la tâche consommatrice (et donc le message) ne s'exécutent pas forcément avec la même période. Généralement, ces précédences sont traitées en discrétisant le graphe de précédence de manière à se ramener à un problème avec des précédences simples [RCK01]. Récemment, un travail réalisé à l'ONERA [FBG+10] a permis de prendre en compte ces contraintes directement dans la méthode d'affectation des priorités et l'analyse d'ordonnançabilité.

[21] *Priority Ceiling Protocol*
[22] *Stack Resource Policy*
[23] *compatible avec la stratégie de l'algorithme d'ordonnancement utilisé*

C.3 Ordonnancement d'un système multiprocesseur

C.3.1 Caractéristiques et classification des systèmes multiprocesseur

Sur une plate-forme multiprocesseur, les applications disposent de plusieurs processeurs simultanément pour réaliser leurs calculs. Nous pouvons distinguer deux types de plate-forme multiprocesseur :

- les **plate-formes homogènes** sont constituées de plusieurs processeurs identiques ayant la même puissance de calcul ;
- les **plate-formes hétérogènes** sont constituées de plusieurs processeurs différents n'ayant pas la même puissance de calcul.

L'analyse d'ordonnançabilité des systèmes multiprocesseurs s'applique en général à des plate-formes homogènes partageant une même référence temporelle ainsi qu'une mémoire commune. Les techniques décrites par la suite s'appliquent aux systèmes comprenant des tâches périodiques ou sporadiques indépendantes. Si ces hypothèses ne sont plus vraies (divergence des horloges ou communications interprocesseur non négligeables) le problème se ramène aux techniques de l'ordonnancement distribué (*C.f.* partie C.4, page 193). Pour résumer, la problématique de l'ordonnancement multiprocesseur est un problème à deux dimensions dans lequel il faut déterminer (1) une **organisation spatiale** des tâches sur les processeurs et (2) une **organisation temporelle** des tâches sur chaque processeur. Cela revient à devoir considérer un ensemble de n tâches sur une architecture composée de m processeurs identiques. Pour résoudre ce problème il existe deux grands types de méthodes :

1. Dans l'**ordonnancement global**, il s'agit d'appliquer sur tous les processeurs une stratégie unique d'ordonnancement et de faire en sorte qu'à chaque instant, les n tâches les plus prioritaires soient attribuées aux m processeurs. Dans ce cas, les migrations des tâches d'un processeur à un autre sont autorisées. Ces techniques sont décrites dans la partie C.3.2.

2. Dans l'**ordonnancement partitionné**, le but est de trouver un partitionnement de l'ensemble des n tâches en m sous-ensembles (m étant le nombre de processeurs), puis **ensuite** d'ordonnancer chaque sous-ensemble sur un processeur distinct. Dans ce cas, les migrations des tâches d'un processeur à un autre ne sont pas autorisées. Ces techniques sont décrites dans la partie C.3.3

C.3.2 Les techniques d'ordonnancement global

Dans ce cas, on se place dans le cas d'un algorithme centralisé qui tente de résoudre la dimension spatiale et temporelle du problème d'ordonnancement de manière conjointe (*C.f.* figure C.10). Pour cela, il doit choisir la tâche à exécuter, puis le processeur sur lequel cette tâche doit être exécutée. C'est un approche en-ligne qui est peu utilisée en particulier à cause du coup effectif des migrations et des préemptions des tâches (négligé lors des hypothèses initiales).

De nombreuses recherches ont été menées sur l'ordonnancement multiprocesseur selon l'approche globale. En premier lieu, il est important de noter que les algorithmes d'ordonnancement monoprocesseur ont leurs variantes globales pour les systèmes multiprocesseurs. Dans le cas des ordonnanceurs statiques, nous pouvons citer les variantes de l'algorithme RM telles que TkC [AJ00] et RM-US[ξ] [ABJ01] ou encore la variante DM-US[ξ] [BCL05] de l'algorithme DM.

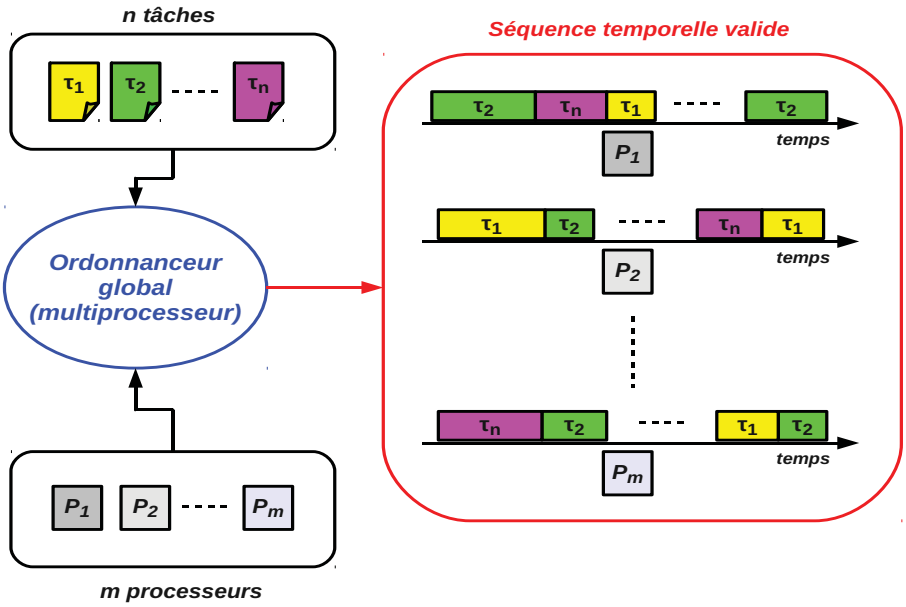


FIG. C.10 – Ordonnancement multiprocesseur global

Le lecteur est invité à se référer à la thèse de Björn Adersson [And03] pour une vision plus complète et détaillée des méthodes et des algorithmes pour l'ordonnancement global statique appliqué au contexte des systèmes multiprocesseurs. Dans le cas dynamique, l'utilisation des algorithmes classiques a été évaluée assez tôt dans le contexte multiprocesseur [DM89]. Dans ce contexte, il existe aussi de nombreuse extensions des algorithmes classiques telles que des variantes de l'algorithme EDF [Bak03, FB09] ainsi que des nouvelles approches assez complexes telles que l'algorithme LLREF[24] [CRJ06] ou encore EDLZ[25] [Lee94]. Le lecteur est invité à se référer à la thèse de Marko Bertogna [Ber08] décrivant les différents algorithmes dynamique pour l'ordonnancement global dynamique et proposant des études comparatives avec les approches statiques. L'algorithme d'ordonnancement Pfair[26] [BCPV93, AS00] est un algorithme présentant une approche nouvelle avec des résultats théoriques intéressants. Cependant, dans la pratique, cet algorithme n'est pas applicable car le mécanisme impose que les tâches soient préemptées très souvent pour en garantir l'optimalité, ce qui conduit à une surcharge système pour les préemptions et les migrations.

C.3.3 Les techniques d'ordonnancement partitionné

Dans ce type de méthode, on traite chaque dimension du problème d'ordonnancement de manière séparée. Dans un premier temps, une technique de placement des tâches est appliquée afin d'organiser spatialement le système, c'est-à-dire répartir les tâches sur les différents processeurs. Ensuite, dans un deuxième temps, une stratégie d'ordonnancement monoprocesseur est appliquée localement à chacun des processeurs pour assurer le séquencement temporel correct des tâches qui lui sont assignées. C'est donc une approche hors-ligne très utilisée particulièrement dans le contexte des systèmes embarqués. Inversement, de la même façon que les techniques statiques d'ordonnancement monoprocesseur, les méthodes d'ordonnancement partitionné ne permettent

[24] *Largest Local Execution Time First*
[25] *Earliest Deadline with Zero Laxity*
[26] *Proportionate Fair*

pas de prendre en compte les évolutions du système pendant l'exécution.

La première étape afin de trouver une répartition adéquate des tâches sur les processeurs, est équivalente au problème du **Bin-Packing**. L'objectif est de déterminer la manière de diviser l'ensemble des tâches en un nombre de sous-ensembles égal au nombre des processeurs (*C.f.* figure C.11). Les tâches qui appartiennent au même sous-ensemble doivent être ordonnançables sur le processeur concerné.

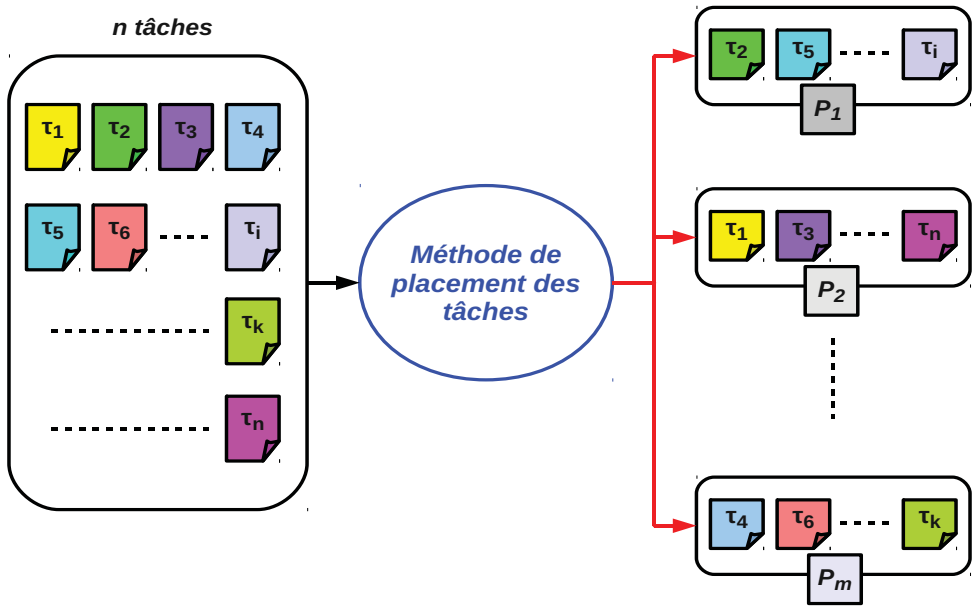


FIG. C.11 – Ordonnancement multiprocesseur partitionné : Étape d'organisation spatiale

Généralement, si l'application n'a pas de contraintes spécifiques de placement des tâches telles que celles que l'on peut voir dans les systèmes distribués (*C.f.* partie C.4.3, page 195), on peut utiliser les heuristiques classiques de mise en sac telles que : FF[27] avec laquelle chaque tâche est assignée au premier processeur qui peut l'accepter ; BF[28] avec laquelle chaque tâche est assignée au premier processeur pouvant l'accepter et qui a sa capacité la plus faible (il est peu chargé) ou encore NF[29] et WF[30]. Il existe de nombreuses variantes de ces méthodes classiques ainsi que des méthodes hybrides telles que HI-BP[31] [ARGA04].

Une fois qu'une des méthodes heuristiques a été appliquée et que les tâches sont assignées sur les différents processeurs du système, il faut assurer l'ordonnancement temporel (donc une séquence temporelle valide) de ces tâches sur chacun des processeurs (*C.f.* figure C.12).

La littérature est abondante sur les techniques d'ordonnancement partitionné impliquant les mises en œuvre de certaines heuristiques combinées avec des algorithmes d'ordonnancement monoprocesseur sur chacun des processeurs. Nous pouvons citer une étude utilisant l'algorithme

[27] *First Fit*
[28] *Best Fit*
[29] *Next Fit*
[30] *Worst Fit*
[31] *Hybrid Improvement heuristic for Bin-Packing*

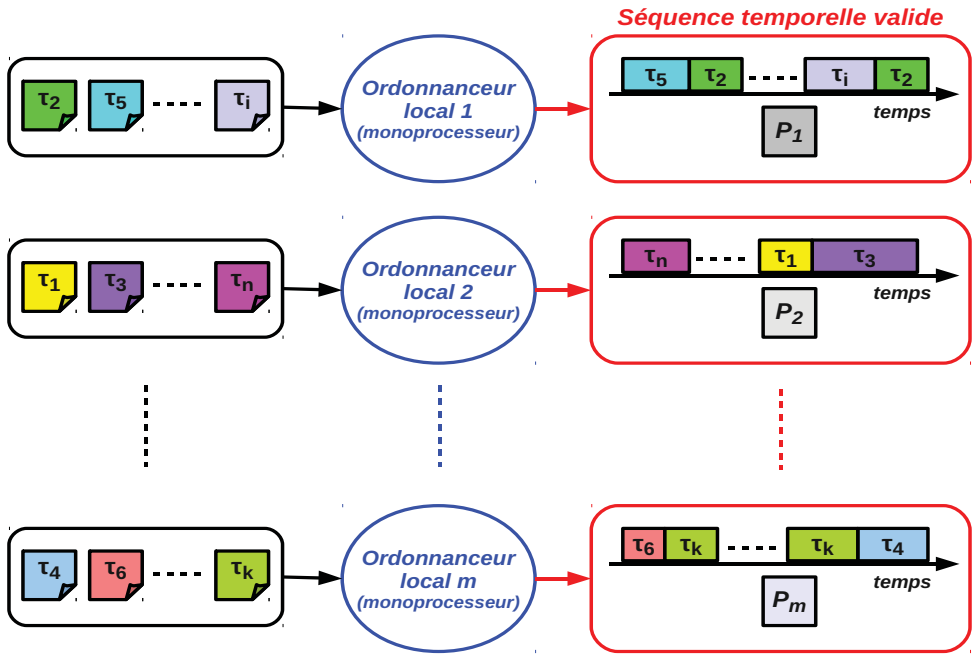


Fig. C.12 – Ordonnancement multiprocesseur partitionné : Étape d'organisation temporelle

RM combiné avec les heuristiques FF et BF [OS93] et une autre associant RM avec l'heuristique FFDU[32] [OS95] pour les cas statiques. Et dans le cas de l'utilisation d'ordonnanceurs dynamiques, nous pouvons citer une étude utilisant l'algorithme EDF combiné avec FF [LGDG00] ainsi qu'une étude complète évaluant l'utilisation de EDF avec la plupart des heuristiques connues [LDG04].

C.3.4 Les techniques de gestion des ressources

Dans un système multiprocesseur, tous les processeurs partagent une mémoire commune. Ainsi, de la même façon que pour les systèmes monoprocesseurs, les tâches peuvent être en conflit sur les ressources mémoire du système (particulièrement lorsque ces tâches sont des threads).

Si l'on se place dans le cas des ordonnancements partitionnés, les ressources communes peuvent être gérées par des extensions des techniques utilisées dans le contexte monoprocesseur (*C.f.* partie C.2.4, page 185). Rajkumar et al [RSL88] ont introduit la variante mulitprocesseur MPCP[33] du protocole PCP ; notons que ce protocole a été utilisé dans certains travaux du standard RT-CORBA (*C.f.* partie 3.2.2, page 50). Chen et al [CTB94] ont aussi présenté leur variante de ce protocole, nommée MDPCP, adaptée aux systèmes multiprocesseurs ordonnancés par une technique de partitionnement combinée à EDF. Gai et al [GLN01] ont, quant à eux, proposé une extension du protocole SRP, nommé MSRP[34], utilisable pour les techniques d'ordonnancement partitionné combinées à des algorithmes statiques et dynamiques (EDF).

Pour le cas de l'ordonnancement global, d'autres techniques ont été envisagées comme l'utilisation de *Spin-based queue locks* [MCS91], notamment dans le cas de l'utilisation de l'algorithme Global-EDF [DLA06]. Nous pouvons enfin citer le protocole FMLP[35] [BLBA07] qui peut être

[32] *First-Fit Decreasing Utilization*
[33] *Multiprocessor PCP*
[34] *Multiprocessor SRP*
[35] *Flexible Multiprocessor Locking Protocol*

appliqué pour gérer les ressources lorsque l'on utilise des techniques d'ordonnancement global ainsi que celles d'ordonnancement partitionné.

C.3.5 Discussion

Nous avons essayé dans cette partie de fournir une vision globale des différentes techniques d'ordonnancement utilisées dans le contexte des systèmes multiprocesseurs homogènes. Les tests de faisabilité, les algorithmes et leurs spécificités ne sont pas détaillés ici. Le lecteur intéressé peut se référer à l'état de l'art récent [DB11] dans lequel la plupart de ces méthodes algorithmiques et les preuves mathématiques sont décrites.

Lorsque des contraintes plus complexes sont à prendre en compte (communications interprocesseurs non négligeables, divergence des horloges matérielles), peu de résultats existent concernant les approches présentées ci-dessus. En pratique, l'ordonnancement de tels systèmes repose sur les mêmes principes que ceux utilisés pour les systèmes distribués.

C.4 Ordonnancement d'un système distribué

C.4.1 Problématique des systèmes distribués temps réel

Comme nous l'avons vu dans l'annexe précédente (*C.f.* annexe B, page 161), l'évolution technologique des réseaux de communication a amené au développement des systèmes distribués temps réel. Le problème d'ordonnancement est beaucoup plus complexe dans le cas distribué : la politique d'ordonnancement doit prendre en compte la répartition des tâches sur différents nœuds du système ainsi que les communications entre les tâches sur le(s) réseau(x). De plus, généralement, les différents nœuds qui composent le système distribué ne partagent pas la même horloge matérielle et, par conséquent, le système est asynchrone ce qui pose un problème pour le temps réel. Dans ce type de système, il est généralement impossible de mettre en application un unique mécanisme d'ordonnancement qui centralise les informations et les décisions d'ordonnancement. En effet, par rapport à l'ordonnancement multiprocesseur, quatre difficultés supplémentaires apparaissent :

1. La difficulté de maintenir une vision globale et cohérente de l'état du système : centraliser les décisions d'ordonnancement en ligne introduirait un surcoût élevé (dû aux communications et aux calculs supplémentaires) qui impacterait le comportement temporel du système. De plus, les sites ne partagent pas la même horloge matérielle, il est donc nécessaire de mettre en place des techniques de synchronisation afin d'assurer : (1) un comportement fonctionnel cohérent et (2) une horloge matérielle globale (généralement indispensable pour la validation de bout en bout). La cohérence temporelle et fonctionnelle de l'application sera donc dépendante des caractéristiques et de la précision de ces mécanismes de synchronisation.
2. L'hétérogénéité des nœuds implique que les tâches ne seront pas exécutées dans les mêmes conditions sur chacun des nœuds. Elle peuvent par exemple avoir des temps d'exécution différents d'un nœud à un autre.
3. Les migrations des tâches d'un nœud vers un autre sont en général plus complexes et plus difficiles à mettre en œuvre que dans le cas des systèmes multiprocesseurs. La migration d'une tâche implique un transfert complet du contexte de la tâche (état, données,...) et une reconstruction de ce contexte sur un autre nœud pouvant être hétérogène [Ell94a] (par exemple, si le système d'exploitation est différent). Si les migrations sont autorisées, les coûts de transfert et de reconstruction ne peuvent plus être négligées.
4. Le coût des communications ne peut pas, non plus, être négligé et il est nécessaire de prendre en compte l'ordonnancement de la ressource de communication (réseau) entre les processeurs afin d'assurer des livraisons pour les messages compatibles avec les contraintes temporelles du système.

Pour un système distribué temps réel, l'analyse passe obligatoirement par une étude hors-ligne afin d'appréhender les différents aspects induits par la distribution. Cette étude repose sur cinq points essentiels :

1. L'établissement de **mécanismes de synchronisation** efficaces entre les applications distribuées pour assurer un ordonnancement fonctionnel et temporel correct (*C.f.* partie C.4.2).
2. Le **placement des différentes tâches** du système sur chacun des nœuds du système (*C.f.* partie C.4.3, page 195).
3. La **configuration de chaque ordonnanceur local** des différents nœuds. Cet ordonnanceur s'occupe de gérer un sous-ensemble identifié des tâches du système. Généralement, cette étape est prise en compte lors de l'affectation des tâches sur les nœuds.

4. La **configuration du média de communication** qui revient à mettre en place des mécanismes pour assurer que les communications seront toujours correctes pour le respect des contraintes temporelles (*C.f.* partie C.4.4, page 197).
5. Une **démarche de validation globale** pour assurer que l'ensemble du système distribué se comporte correctement (*C.f.* partie C.4.5, page 199). Généralement, cela suppose de mettre en place une méthode, prenant en considération les points précédents, qui permet d'assurer que le comportement de l'application sera correct de bout-en-bout (pour toutes les exécutions et toutes les communications).

C.4.2 La nécessité de la synchronisation

Dans un système distribué, les communications ne peuvent plus être négligées et une application temps réel est explicitement composée d'un ensemble de tâches communicantes. Dans ce cas, comme nous l'avons évoqué précédemment (*C.f.* partie 6.1.3, page 117), deux facettes complémentaires sont à considérer : (1) la cohérence fonctionnelle de l'application et (2) la cohérence temporelle de l'application. La mise en place de mécanismes de synchronisation est essentielle pour garantir le bon fonctionnement d'une application distribuée et il faut distinguer ces deux points complémentaires :

1. La **synchronisation fonctionnelle** : Dans une application distribuée il faut s'assurer que le cadencement des communications et des exécutions de l'application fournira des résultats qui seront toujours justes et pertinents. Pour cela, il faut mettre en place une méthode de synchronisation pour garantir un séquencement des exécutions et des communications entre les tâches applicatives de manière à assurer une exécution correcte fonctionnellement. Luc Maillet dans sa thèse [Mai96] propose une extension du modèle classique des tâches périodiques qui permet de prendre en compte les interdépendances fonctionnelles entre les tâches. Son approche est basée sur la notion de *point de synchronisation* défini comme un instant de l'exécution d'une tâche où son exécution est suspendue en attente d'un événement extérieur (typiquement la réception d'un message). Jun Sun, dans son travail de thèse [Sun97], a proposé cinq protocoles de synchronisation pour gérer l'interdépendance entre les tâches ainsi qu'une prise en compte dans les calculs des pires temps de réponse pour chaque tâche (pour une validation de bout en bout). Dans, le cadre de notre travail, nous avons proposé plusieurs modes d'exécution (*C.f.* partie 4.3, page 80) notamment l'utilisation des mécanismes de gestion de la norme HLA qui permet un cadencement fonctionnel correct d'une application de simulation distribuée. Le problème de la synchronisation fonctionnelle est particulièrement illustré par la résolution des équations différentielles dans le cadre de la mise en œuvre de notre simulateur de vol [SCS11].
2. La **synchronisation temporelle** : Comme nous l'avons expliqué précédemment, les systèmes distribués ne partagent généralement pas une horloge matérielle commune. Toutes les techniques de validation distribuée actuelles (*C.f.* partie C.4.5, page 199) font l'hypothèse de l'existence d'une horloge globale absolue pour tous les sites. Cette horloge commune est supposée précise et donc sans divergence. L'existence de cette horloge globale sert à définir les modèles analytiques des tâches et des messages par rapport à un même référentiel temporel. En se basant sur cette hypothèse, la méthode de validation (telle que l'étude du temps de réponse des tâches) garantit le bon ordonnancement temporel pour que l'application distribuée donne ses résultats dans les bons temps. Cependant, dans la réalité, cela n'est pas aussi simple et les techniques de synchronisation des horloges nécessitent des mécanismes matériels ou logiciels complexes. Dans le cas de la plate-forme PRISE, les différents nœuds de notre plate-forme sont synchronisés par une horloge matérielle précise :

la RCIM[36] [Con01]. Il existe aussi des techniques logicielles de synchronisation des horloges telles que celles présentées dans l'annexe précédente (*C.f.* partie B.4, page 170). Dans tous les cas, la pertinence de la technique de validation de bout en bout sera impactée par la précision de la méthode de synchronisation des horloges choisie.

C.4.3 Le placement des tâches et l'affectation de priorité

Par rapport aux systèmes multiprocesseurs, la problématique du placement des tâches dans le cas distribué est plus complexe. Les heuristiques classiques issues du Bin-Packing ne sont plus adaptées. Le placement ne se résume plus à distribuer n tâches sur m processeurs en cherchant à garantir le taux d'utilisation de chaque processeur. Il faut prendre en compte la validité de l'ordonnancement pour chaque nœud, la charge du réseau, les coûts dus à la duplication des différentes tâches ou encore l'utilisation mémoire de chacun des nœuds du système. En plus, certaines contraintes peuvent se rajouter pour l'affectation des tâches afin de répondre à certaines exigences applicatives précises :

- Les **contraintes de résidence** : une tâche doit d'exécuter sur un nœud particulier. Cela est dû à l'utilisation de ressources logicielles ou matérielles spécifiques et uniquement accessibles par certains nœuds (par exemple des capteurs ou des actuateurs).
- Les **contraintes de co-résidence** : cette contrainte impose que plusieurs tâches doivent être placées sur le même noeud (par exemple, plusieurs tâches qui doivent utiliser une même ressource).
- Les **contraintes d'exclusion** : il s'agit d'interdire que certaines tâches résident sur les mêmes nœuds, en particulier, lorsque des tâches sont des répliques d'autres tâches (redondance) pour assurer des propriétés de tolérance aux fautes (par exemple, lors d'une défaillance de l'un des nœuds du système).

Dans le cadre de la recherche d'un placement valide ou optimal assurant l'ordonnançabilité de chacun des nœuds, les concepteurs d'application ont eu alors recours à des méthodes algorithmiques plus complexes que les heuristiques classiques. Les méthodes de résolution couramment rencontrées se répartissent en deux classes :

1. Les **méthodes optimales ou exactes** produisent toujours une solution valide si elle existe mais elles sont très coûteuses en temps processeur ainsi qu'en espace mémoire. Cela amène souvent à les rendre inexploitables[37] sur des problèmes de grande taille malgré qu'il soit possible de les guider sans en perdre leur complétude[38] et ainsi de les mettre en œuvre de manière plus efficace. Parmi ces méthodes exactes, on trouve les techniques issues de la théorie des graphes qui sont utilisées dans [CL87, Hou90] pour optimiser l'allocation de tâches sur des systèmes temps réel distribués. Cette approche permet de déterminer la solution optimale minimisant les temps de réponse pour les tâches ainsi que la charge du réseau. En revanche, le formalisme de la théorie des graphes ne permet pas d'exprimer des contraintes temporelles et donc ces méthodes ne permettent pas de garantir une allocation ordonnançable. D'autres techniques existent comme les techniques issues de la programmation mathématique. Dans ce cas, le problème est vu comme un problème d'optimisation combinatoire permettant à travers une fonction de coût d'exprimer les différentes

[36] *Real-Time Clock & interruption Module*

[37] *Explosion combinatoire*

[38] *La complétude d'une méthode désigne sa propriété à pouvoir statuer que le problème qu'elle étudie n'a pas de solution. La méthode est alors dite complète. Une heuristique n'est pas complète car si aucune solution n'est trouvée, nous ne pouvons pas conclure que le problème n'a pas de solution.*

contraintes (et donc les contraintes de temps). On applique ensuite la méthode de résolution exacte choisie. Les plus connues sont les procédures par séparation et évaluation (dites procédures B&B[39]) qui consistent à explorer l'espace de recherche (le plus souvent sous forme d'arbre) qui est, généralement, composé des combinaisons possibles pour les allocations des tâches sur les processeurs. Peng et Shin [PS89] présentent une méthode pour l'allocation de tâches temps réel périodiques et dépendantes basée sur deux algorithmes B&B. Le premier, noté B&BA[40], détermine une affectation optimale des tâches sur les processeurs du système distribué. Le second algorithme, noté B&BS[41], détermine un ordonnancement optimal pour les tâches allouées sur chacun des nœuds. Cette méthode a été reprise dans [HS94] en y ajoutant une méthode de réplication des tâches pour garantir des propriétés de tolérance aux pannes. Enfin, nous pouvons citer les travaux de Richard et Cottet [RRC01] qui proposent une méthode complète permettant l'allocation des tâches sur les nœuds, l'affectation des priorités pour ces tâches ainsi que l'affectation des priorités pour les messages (sur le réseau CAN). Cette méthode est basée sur des techniques de B&B combinées avec une analyse holistique pour déterminer les temps de réponse des tâches et des messages.

2. Les **méthodes approchées** tentent de trouver une solution s'approchant de la solution optimale à l'aide d'algorithmes polynomiaux et sont donc plus rapides et plus simples que les méthodes exactes. Cependant, elles présentent l'inconvénient de perdre la proprété de complétude des méthodes optimales. Parmi ces techniques, nous pouvons citer la méthode dite du *recuit simulé*[42], inspirée de la physique des matériaux (sidérurgie) [Kir83], qui a été appliquée dans de nombreux travaux pour résoudre le problème de l'allocation de tâches aux processeurs [BM91, TBW92, CA95]. Ramamritham [Ram90] a proposé une heuristique, adaptée pour des tâches périodiques communicantes, qui est basée sur des techniques de regroupement des tâches. Pour cela, des paramètres calculés à partir des rapports entre les temps d'exécution et les temps de communication permettent de pondérer les arcs du graphe pour guider le parcours de l'heuristique. Enfin, nous pouvons citer l'utilisation des algorithmes génétiques qui sont des algorithmes de recherche stochastiques basés sur la théorie de Darwin (l'évolution biologique des espèces[43]). Ces méthodes proposent des propriétés particulièrement intéressantes dans la résolution de différents problèmes d'optimisation (*C.f.* chapitre 2 du livre de Thomas Weise [Wei09]). Une méthode très complète, basée sur ces algorithmes spécifiques, permettant l'allocation ainsi que l'ordonnancement valide d'un ensemble de tâches communicantes sur un système distribué a été proposé dans le travail de thèse de Leïla Baccouche [Bac95]. Son approche est basée sur un codage spécifique des individus[44] ainsi que des opérateurs génétiques (mutation et de croisement) adaptés au problème d'ordonnancement. Cette méthode permet aussi de prendre en compte la charge du réseau dans le problème d'optimisation ainsi que la duplication des tâches pour les propriétés de tolérance aux pannes.

   Toutes ces approches tentent de trouver un placement optimal au regard de plusieurs critères : principalement, l'ordonnançabilité du système mais aussi la charge du réseau, le taux de duplication des tâches ou encore l'utilisation mémoire des nœuds. Pour cela, ce

[39] *Branch and Bound*
[40] *Branch and Bound Allocation*
[41] *Branch and Bound Sceduling*
[42] *Simulated Annealing*
[43] *Les espèces vivantes évoluent et s'adaptent à leur milieu permettant ainsi la sélection naturelle et la survie de l'espèce la plus adaptée au milieu considéré.*
[44] *Les individus sont les solutions possibles (même mauvaises) d'un problème dans un algorithme génétique, le but étant d'obtenir les individus (et donc les solutions) les plus adaptés au problème à la fin de l'exécution de l'algorithme.*

problème d'optimisation multi-objectif est ramené à un problème d'optimisation classique (1 seul critère) à l'aide d'une somme associant un poids (une pondération) à chacun des objectifs. Dans certains cas, cette technique ne permet pas d'obtenir les solutions optimales au regard de chacun des critères pris individuellement. Les algorithmes génétiques présentant des résultats très intéressants dans le contexte de l'optimisation multi-objectif [KCS06], une extension des travaux de Leïla Baccouche à ce contexte conduirait à des études intéressantes.

C.4.4 La prise en compte des communications

Dans un système multiprocesseur, la relation de communication est locale et l'échange se fait via une mémoire partagée avec un coût temporel supposé négligeable devant les autres grandeurs du système (en particulier les temps d'exécution des tâches). Dans un système distribué, la relation de communication est distante ; alors cette communication nécessite l'utilisation du réseau entraînant un coût temporel non négligeable. Le respect des délais de transfert des messages est donc un des aspects à prendre en considération. Les communications en concurrence doivent se partager le(s) réseau(x) de la même façon que les tâches doivent se partager le processeur pour s'exécuter. Le message est considéré comme l'unité de communication pour les échanges entre les tâches [SB98, MMMM01] et il peut être vu de la même façon qu'une tâche. Ainsi, nous pouvons distinguer plusieurs types de messages :

- les **messages périodiques** sont caractérisés par un intervalle de production constant (*i.e* période). Ils sont souvent utilisés pour la communication entre tâches périodiques ou pour des captures périodiques des événements identifiés par les capteurs (tel que le joystick dans notre simulateur de vol (*C.f.* partie 4.2.2, page 74).
- les **messages apériodiques ou sporadiques** sont souvent utilisés pour véhiculer des informations complémentaires pour l'application telles que des alarmes ou encore des communications initiées par certaines tâches apériodiques. Ce type de message arrive à des instants non fixés. Le succès du transfert des messages apériodiques repose sur la connaissance des lois de production (message sporadique) et des échéances correspondantes.

Nous supposerons dans la suite que les messages apériodiques temps réel peuvent être traités par des variantes des méthodes décrites précédemment (*C.f.* partie C.2.3, page 184). Les messages périodique peuvent être décrits par un modèle analytique proche de celui utilisé pour les tâches périodiques. Un message $m_{i,j} \in \mathcal{M}$ traduisant une communication entre la tâche τ_i et la tâche τ_j est décrit par un quadruplet $< r_{i\rightarrow j}, C_{i\rightarrow j}, D_{i\rightarrow j}, T_{i\rightarrow j} >$ tel que :

- $r_{i\rightarrow j}$ l'instant de production du message par la tâche τ_i ;
- $C_{i\rightarrow j}$ est la durée de transmission du message c'est-à-dire le temps nécessaire pour acheminer le message de la tâche productrice vers la tâche consommatrice ;
- $D_{i\rightarrow j}$ est le délai critique à respecter pour le message qui est l'intervalle de temps, à partir de l'instant de production, durant lequel le message doit être transmis de la tâche productrice vers la tâche consommatrice (en dehors de cet intervalle, le message devient non valable) ;
- $T_{i\rightarrow j}$ est la période d'émission par la tâche productrice τ_i. Généralement, la période du message est un multiple de la période la tâche productrice : $T_{i\rightarrow j} = k \times T_i$ avec k un entier positif. Par exemple, dans le cas des précédences simples, la période de la tâche productrice, du message et de la tâche consommatrice sont égales : $T_j = T_{i\rightarrow j} = T_j$ (et donc $k = 1$).

De la même façon que pour le modèle des tâches, la durée de transmission $C_{i\rightarrow j}$ du message correspond au maximal (au pire des cas) nécessaire pour le transmettre de la tâche productrice vers la tâche consommatrice . En général, on essaye de déterminer le majorant des temps de

transmission possible noté : WCTT[45]. Ce temps doit prendre en compte les temps nécessaires à différentes étapes de la transmission (*C.f.* figure C.13) :

1. Le temps $\mathbf{t_a}$ est le temps nécessaire pour que le nœud transfère le message émis par la tâche τ_1 dans sa file d'émission. Ce temps dépend de la structure du nœud et en particulier la structure des matériels dédiés aux échanges entre les mémoires processeurs et celles de la carte réseau. Généralement, dans la plupart des applications distribuées temps réel, il est supposé nul par rapport aux autres grandeurs du système.
2. Le temps $\mathbf{t_b}$ est le temps nécessaire pour que le nœud produisant le message obtienne l'accès au réseau afin de transmettre ce message. Ce temps dépend du protocole MAC mis en œuvre sur le réseau (*C.f.* partie B.2.2, page 165)
3. Le temps $\mathbf{t_c}$ est le temps nécessaire à la transmission effective du message sur le réseau. Ce temps est impacté par deux paramètres principaux : (1) la longueur du message c'est-à-dire le nombre d'octets ou de bits le constituant et (2) la distance de la connexion entre le producteur et le(s) récepteur(s).
4. Le temps $\mathbf{t_d}$ est le temps inverse de $\mathbf{t_a}$ nécessaire pour faire parvenir le message de la file d'insertion vers la tâche réceptrice.

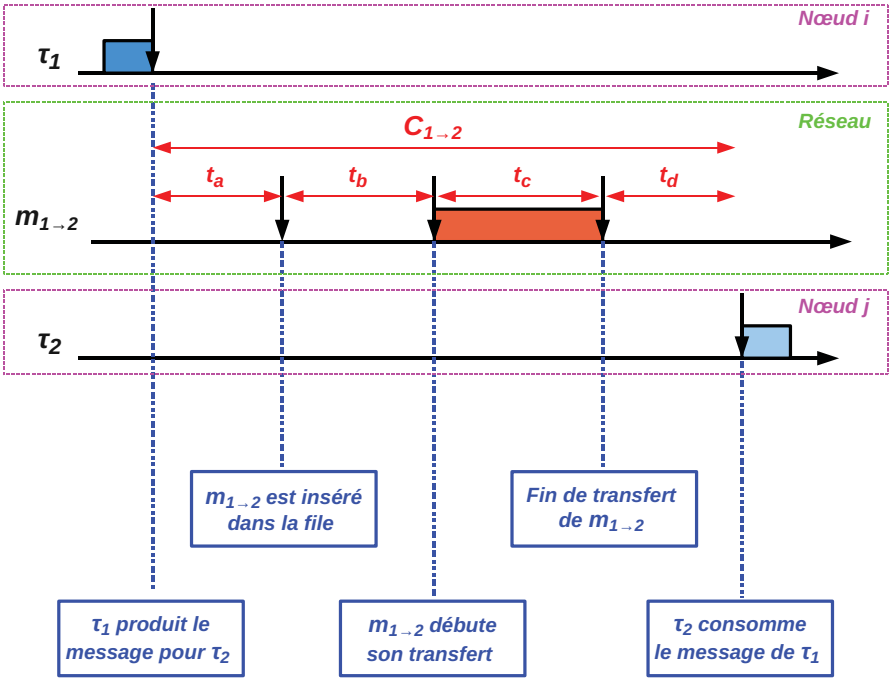


FIG. C.13 – Les étapes pour la transmission d'un message

Nous ne détaillerons pas les techniques pour déterminer ces temps. Toutefois, le lecteur intéressé est invité à se référer au manuscrit de thèse de Samia Saad-Bouzefrane [SB98] pour comprendre les détails des calculs des délais d'accès au réseau et de transfert des messages en fonction de certains protocoles MAC. L'ordonnançabilité d'un réseau peut être déterminé à l'aide des tests de faisabilité des variantes non-préemptives des algorithmes classiques si les protocoles d'accès sont basés sur l'attribution des priorités. Il existe aussi une méthode analytique complexe, nommée le *Network Calculus* [LBT11], qui est spécifiquement dédiée à l'étude des flux du réseau dans un système distribué.

[45] *Worst-Case Transit Time*

C.4.5 Les démarches pour la validation globale

La problématique de l'ordonnancement distribué est très complexe et la résolution d'un problème d'ordonnancement sur de telles architectures est dépendante de nombreux points que nous avons explicités dans les parties précédentes. Une méthode de validation globale est la dernière étape pour s'assurer que toutes les exécutions des tâches et toutes les communications inter-tâches se feront dans les bons temps : c'est la validation de *bout en bout*. La complexité du problème et la dépendance avec de nombreux paramètres liés à la structure matérielle (nombre de processeurs, topologie du ou des réseau(x), hétérogénéité,...) impliquent qu'il existe peu de démarches globales qui permettent de vérifier le comportement temporel correct d'une application distribuée.

La méthode la plus connue et la plus utilisée est l'analyse holistique de Tindell et Clark [TC94]. Cette méthode est basée sur les calculs des temps de réponse des tâches et des messages d'une application distribuée. Même si cela n'est pas clairement explicité dans l'article original, cette méthode suppose que toutes les communications entre les tâches sont sous la forme de précédences simples (*C.f.* partie C.2.5, page 187). Nous ne détaillerons pas cette méthode dans cette partie mais le lecteur intéressé peut se référer à la partie précédente de ce document consacrée à la validation de simulations distribuées selon le mode *Data Flow* (*C.f* partie 6.2, page 117). Le document de Richard et Cottet [RRC03b] permet aussi d'avoir une vision très détaillée de cette méthode ainsi qu'une description complète des différents algorithmes de calcul des temps de réponse aussi bien dans le cas d'un ordonnancement statique (RM ou DM) que dans le cas dynamique avec EDF. Jun Sun propose dans sa thèse [Sun97], une méthode complète pour la validation d'un système distribué où les nœuds sont ordonnancés par des politiques statiques. Sa méthode, basée aussi sur l'analyse des temps de réponse des tâches et des messages, est plus complète que celle de Tindell et Clark notamment par la distinction de différents protocoles possibles pour la synchronisation[46] entre les tâches. L'analyse holistique est ainsi englobée dans la méthode de Jun Sun et ses travaux ont mis exergue certaines lacunes dans les preuves fournies initialement par Tindell et Clark [SL96] avant qu'un complément de preuve soit apporté par Palencia et al [PGGH97]. Samia Saad-Bouzefrane [SB98], quant à elle, propose une méthode de mise à jour des dates d'activation des tâches et des messages. Cependant, dans le cas d'une politique statique, les ordonnanceurs (DM ou RM) ne sont plus optimaux et l'ordonnançabilité des sites peut être remise en question. Nous pensons toutefois que si cette approche est combinée avec un ordonnancement par l'algorithme d'Audsley[47] [Aud91] sur chacun des nœuds alors l'approche reste valable pour démontrer une validation de bout en bout.

C.5 Les différents outils existants

Il existe de nombreux logiciels pour l'aide à la conception, la simulation et la validation de systèmes temps réel. Nous donnerons ici, une liste non exhaustive des quelques logiciels existants :

- **PERTS**[48] [LRD+93] est un logiciel équipé d'un interface graphique d'aide à la validation d'applications monoprocesseur ou réparties. Il permet de visualiser la séquence d'exécution sur chaque processeur et d'évaluer les caractéristiques temporelles d'une application. Il offre aussi la possibilité de choisir les algorithmes d'ordonnancement RM et DM ainsi que

[46] *Il s'agit ici de synchronisation fonctionnelle entre les tâches, l'hypothèse de l'existence d'une horloge globale est conservée pour la pertinence des calculs des différents temps de réponse.*

[47] *L'algorithme de Audsley est le seul algorithme à être optimale pour les ensembles de tâches avec des dates de réveil différentes de* 0

[48] *Prototyping Environment for Real-Time Systems*

les protocoles d'allocation de ressources PCP ou SRP. Ainsi, l'analyse d'ordonnançabilité est établie pour chaque processeur en fonction de l'algorithme d'ordonnancement choisi ainsi que du protocole d'allocation de ressources. En revanche, dans le cas d'applications distribuées, la prise en compte du réseau est plus succincte et l'outil calcule simplement le taux d'utilisation du réseau. Il n'y a aucun moyen de prendre en compte certaines caractéristiques physiques du réseau ou encore différents protocoles de communication (en particulier les protocoles MAC). Cet outil a été utilisé pour la validation d'applications distribuées CORBA [LRD+93].

- **CHEDDAR** [SLNM04, Sin07] [10] est un outil d'ordonnancement open-source développé en Ada et compatible pour de nombreux systèmes d'exploitation (Windows, Solaris et Linux). Il est développé et maintenu au sein de l'équipe LISYC[49] à l'Université de Brest. Cet outil permet de simuler des ordonnancements d'ensembles de tâches par les algorithmes classiques RM, DM, EDF ou encore LLF. Il permet aussi de déterminer l'ordonnançabilité (la faisabilité) de la configuration par le calcul du taux d'utilisation du processeur ou le calcul du temps de réponse (uniquement si l'on utilise RM ou DM). CHEDDAR permet d'étudier des ensembles composés de tâches périodiques et apériodiques (indépendantes ou dépendantes). Il prend également en compte le partage de ressources grâce aux protocoles PIP et PCP. Des outils d'évaluation du temps de réponse de bout en bout pour un système distribué sont aussi disponibles et ils sont basés sur les calculs Holistiques de Tindell et Clark. C'est l'outil que nous avons utilisé dans le cadre de nos travaux sur la validation du comportement temporel de nos simulations distribuées HLA.
- **ARTISST**[50] est un outil open-source d'étude et de simulation de systèmes temps réel complexe développé en `C++` par David Decotigny dans le cadre de sa thèse de doctorat [Dec03]. Il s'agit d'une infrastructure de simulation modulaire et très complète qui repose sur les principes de la simulation distribuée à événement discret (*C.f.* partie 2.2.3, page 15). Cet outil est constitué de différents modules, notamment des modules de simulation dédiés pour les nœuds et un module de simulation pour réseau pour modéliser des systèmes distribués complexes. Un tutoriel ainsi que les sources sont disponibles à l'adresse [5]. Cependant, ce logiciel est uniquement compilable et utilisable sur des systèmes assez anciens (noyaux linux 2.4.x) et nous n'avons pas réussi, dans le cadre de nos travaux, à l'exporter sur les systèmes Linux actuels (et donc de l'utiliser).
- **STRESS** [ABRW94] est un outil dévelloppé par le *Real-Time Research Group* de l'Université de York. Plusieurs stratégies d'ordonnancement sont supportées notamment l'algorithme de Audsley. Il propose aussi un langage associé très riche pour la description des tâches.
- **MOSARTS** est un outil, développé par Samia Saad-Bouzefrane dans le cadre de son doctorat [SB98], pour la validation d'applications temps réel distribuées. Le principe de base repose sur la méthodologie d'évaluation conjointe des tâches et des messages développée dans sa thèse. Le logiciel est implémenté en `C` avec des interfaces graphiques codées en `Tcl/Tk`. Les algorithmes RM, DM et EDF sont pris en compte pour les nœuds ainsi que les protocoles de gestion des ressources PCP, PIP et SRP. Le principale avantage de cet outil est d'offrir une étude complète des différentes stratégies de communication basée sur l'analyse des protocoles MAC : FIP, CSMA/DCR, CAN, FDDI et TDMA.
- Nous pouvons aussi citer **gRMA**[51] [26] qui est un logiciel open-source basé sur Rate

[49] *Laboratoire d'Informatique des SYstèmes Complexes*
[50] *ARTISST is a Real-Time System Simulation Tool*
[51] *a Graphical tool for Rate Monotonic Analysis*

Monotonic dont les sources sont disponibles. **Rapid RMA** [56] est un outil commercial développé par Tri-Pacific Software fournissant une boîte à outils fondée sur PERTS. **Time Wiz** [73] développé par TimeSys Software est du même type se basant aussi sur l'analyse RMA. **ProtEx** [MBB00] et **SymTA/S** [69] sont des outils pour l'étude et le prototypage de certains systèmes distribués temps réel.

Bibliographie

[AA00] T. Andrew Au. Performance Issues of HLA Run Time Infrastructure based on CORBA. In *Proceedings of The Simulation Technology and Training Conference*, SimTecT '00, Sidney, Australia, 2000.

[ABD+97] T. Abdelzaher, M. Bjorklund, S. Dawson, W. c. Feng, F. Jahanian, S. Johnson, P. Marron, A. Mehra, T. Mitton, A. Shaikh, K. Shin, Z. Wang, and H. Zou. ARMADA Middleware and Communication Services. *The Journal of Real-Time Systems*, 16 :127–153, 1997.

[ABJ01] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, RTSS 2001, pages 193–202, London, UK, December 2001.

[Abr70] Norman Abramson. The ALOHA System : Another alternative for computer communications. In *Proceedings of the Fall Joint Computer Conference*, AFIPS '70, pages 281–285, Houston, TX, USA, November 1970.

[ABR+04] Martin Adelantado, Jean-Loup Bussenot, Jean-Yves Rousselot, Pierre Siron, and Marc Betoule. HP-CERTI : Towards a high Performance, high Availability Open-Source RTI for Composable Simulations. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 2004. 04F-SIW-014.

[ABRW94] N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings. STRESS : A Simulator for Hard Real-Time Systems. *SP&S*, 24(6) :543–564, October 1994.

[ABW93] Kevin W. Arthur, Kellogg S. Booth, and Colin Ware. Evaluating 3D task performance for fish tank virtual worlds. *ACM Transactions on Information Systems*, 11(3) :239–265, July 1993.

[ACD+93] Ian F. Akyildiz, Liang Chen, Samir Ranjan Das, Richard M. Fujimoto, and Richard F. Serfozo. The effect of memory capacity on Time Warp performance. *Journal of Parallel and Distributed Computing - Special issue on parallel and discrete event simulation*, 18(4) :411–422, August 1993.

[Ade04] Martin Adelantado. Rapid Prototyping of Airport Advanced Operational Systems and Procedures through Distributed Simulation. *Transactions of the Society for Computer Modeling and Simulation International, Special Issue on Air Traffic Simulation*, 80(1) :1–15, 2004.

[Ade10] Adelantado, M. and Mauget, R. and Igarza, J-L. *Guide HLA Final.* ONERA et CapGemini, 2010.

[ADF+97] T. Abdelzaher, S. Dawson, W. Feng, S. Ghosh, F. Jahanian, S. Johnson, A. Mehra, T. Mitton, J. Norton, A. Shaikh, K. Shin, V. Vaidyan, Z. Wang, and H. Zou. ARMADA Middleware Suite. Technical report, Departement of Electrical Engeneering and Computer Science, University of Michigan, 1997.

[AJ00] B. Andersson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling : to partition or not to partition. In *Proceedings of the Seventh International Conference on Real-Time Systems and Applications*, RTCSA '00, pages 337–, Cheju Island, South Korea, December 2000.

[AK00] M. Law Averill and W.David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Education, 2000.

[All03] Alliot, Jean-Marc. *Qu'est ce que le "Middleware"*. Direction des Services de la Navigation Aérienne, white paper edition, Mars 2003.

[And03] Björn Andersson. *Static-priority scheduling on multiprocessors*. PhD thesis, Chalmers University of Technology - Department of Computer Engineering, Göterborg, Sweden, 2003.

[AOC08] Martin Adelantado, Armand Oyzel, and Jean-Baptiste Chaudron. IESTA (Infrastructure for Air Transport System Evaluation) : A platform allowing acoustics impacts evaluation around airports. In *37th International Congress and Exposition on Noise Control Engineering*, 2008.

[AOC09] Martin Adelantado, Armand Oyzel, and Jean-Baptiste Chaudron. Using the HLA, Physical Modeling and Google Earth for Simulating Air Transport Systems Environmental Impact. In *Proceedings of Spring Simulation Interoperability Workshop*, March 2009. 09S-SIW-045.

[AORC08] Martin Adelantado, Armand Oyzel, Thomas Rivière, and Jean-Baptiste Chaudron. Rapid Prototyping of IESTA : A Platform to Evaluate Innovative Air Transport Concepts. In *Proceedings of American Institute of Aeronautics and Astronautics (AIAA) Modeling and Simulation Technologies Conference*, 2008.

[AR92] Rassul Ayani and Hassan Rajaei. Parallel Simulation using conservative time windows. In *Proceedings of the 24th conference on Winter simulation*, WSC '92, pages 709–717, 1992.

[ARGA04] Adriana C. F. Alvim, Celso C. Ribeiro, Fred Glover, and Dario J. Aloise. A Hybrid Improvement Heuristic for the One-Dimensional Bin Packing Problem. *Journal of Heuristics*, 10(2) :205–229, March 2004.

[Aro97] Aronson, Jesse. Dead Reckoning : Latency Hiding for Networked Games. Technical report, Gamasutra Corporation, Septembre 1997.

[AS00] James H. Anderson and A. Srinivasan. Pfair scheduling : beyond periodic task systems. In *Proceedings of the Seventh International Conference on Real-Time Systems and Applications*, RTCSA '00, pages 297–, Cheju Island, South Korea, 2000.

[AS01] Martin Adelantado and Pierre Siron. Multiresolution Modeling and Simulation of an Air-Ground Combat Application. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 2001.

[Aud91] N.C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start time. Technical Report YCS-164, Real-Time Systems Research Group, Department of Computer Science, University of York, Novembre 1991.

[Awe85] Baruch Awerbuch. Complexity of Network Synchronization. *Journal of the Association for Computing Machinery (ACM)*, 32(4), October 1985.

[Bac95] Leïla Baccouche. *Un mécanisme d'ordonnancement distribué de tâches Temps Réel*. PhD thesis, Institut National polytechnique de Grenoble, Grenoble, France, 1995.

[Bak91] T. P. Baker. Stack-Based Scheduling for Real-Time Processes. *The Journal of Real-Time Systems*, 3(1) :67–99, 1991.

[Bak01] Steve Baker. Middleware to Middleware. In *Proceedings of the 3rd International Symposium on Distributed Objects and Applications*, DOA '01, Rome, Italy, September 2001.

[Bak03] Theodore P. Baker. Multiprocessor EDF and Deadline Monotonic Schedulability Analysis. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, RTSS '03, pages 120–129, Cancun, Mexico, 2003.

[BB02] Albert Benveniste and Gérard Berry. The synchronous approach to reactive and real-time systems. In Giovanni De Micheli, Rolf Ernst, and Wayne Wolf, editors, *Readings in hardware/software co-design*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[BCE+03] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1) :64 – 83, jan 2003.

[BCL05] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. New Schedulability Tests for Real-Time Task Sets Scheduled by Deadline Monotonic on Multiprocessors. In *Proceedings of the 9th International Conference on Principles of Distributed Systemps*, pages 306–321, Pisa, Italy, December 2005.

[BCLR04] François Baccelli, Augustin Chaintreau, Zhen Liu, and Anton Riabov. The One-to-Many TCP Overlay : A Scalable and Reliable Multicast Architecture. Research Report RR-5241, INRIA - Institut National de Recherche en Informatique et Automatique, 2004.

[BCPV93] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress : a notion of fairness in resource allocation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 345–354, San Diego, California, USA, 1993.

[BCSZ98] Pierre Bieber, Jacques Cazin, Pierre Siron, and Guy Zanon. Security Extensions to ONERA HLA RTI Prototype. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 1998. 98F-SIW-086.

[BDL04] G. Behrmann, A. David, and K.G. Larsen. A Tutorial on UPPAAL. In *Proceedings of the International School on Formal Methods for the Design of Computer, Communication and Software Systems - Issues on Formal Methods for the Design of Real-Time Systems*, SFM-RT '04, pages 200–237, Bertinoro, Italy, September 2004.

[Bel66] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2) :78–101, June 1966.

[Ber86] Jean-Louis Bergerand. *LUSTRE : un langage déclaratif pour le temps réel*. PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, France, 1986.

[Ber00] Gérard Berry. *The foundations of Esterel*, pages 425–454. MIT Press, Cambridge, MA, USA, 2000.

[Ber08] Marko Bertogna. *Real-Time Scheduling Analysis for Multiprocessor Platforms*. PhD thesis, Scuola Superiore Sant'Anna, Pisa, Italia, May 2008.

[BFMT98] S. Bachinsky, R. Fujimoto, L. Mellon, and G. Tarbox. RTI 2.0 Architecture. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 1998.

[BG92] Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language : design, semantics, implementation. *Science Of Computer Programming*, 19(2) :87–152, November 1992.

[BI81] Bolt Beranek and Newman Inc. A History of the ARPANET : The First Decade. Technical Report 4799, Department Advanced Research Projects Agency, April 1981.

[Bim07] Franck Bimbard. *Dimensionnement temporel de systèmes embarqués : Application à OSEK*. PhD thesis, Conservatoire National des Arts et Métiers, Paris, France, Novembre 2007.

[BKBH08] Jason Baietto, Joe Korty, John Blackwood, and Jim Houston. *Real Time Linux : The RedHawk Approach*. Concurrent Computer Corporation, white paper edition, September 2008.

[BL05] Azzedine Boukerche and Kaiyuan Lu. A Novel Approach to Real-Time RTI Based Distributed Simulation System. In *Proceedings of the 38th Annual Symposium on Simulation*, ANSS '05, pages 267–274, Washington, DC, USA, 2005. IEEE Computer Society.

[Bla76] Jacek Blazewicz. Scheduling Dependent Tasks with Different Arrival Times to Meet Deadlines. In *Proceedings of the International Workshop on Modelling and Performance Evaluation of Computer Systems*, pages 57–65, Ispra, Italy, October 1976.

[Bla98] Blake, S. and Black, D. and Carlson, M. and Davies, E. and Wang, Z. and Weiss, W. *An Architecture for Differentiated Services (RFC-2475)*. Network Working Group, December 1998.

[BLBA07] Aaron Block, Hennadiy Leontyev, Bjorn B. Brandenburg, and James H. Anderson. A Flexible Real-Time Locking Protocol for Multiprocessors. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '07, pages 47–56, Daegu, Korea, August 2007.

[BM91] S. Wayne Bollinger and Scott F. Midkiff. Heuristic Technique for Processor and Link Assignment in Multicomputers. *IEEE Transactions on Computers*, 40(3) :325–333, March 1991.

[BM05] S. Baskiyar and N. Meghanathan. A survey of contemporary real-time operating systems. *Informatica*, 29(2) :233 – 240, 2005.

[BMK88] D. R. Boggs, J. C. Mogul, and C. A. Kent. Measured capacity of an Ethernet : Myths and Reality. *ACM SIGCOMM Computer Communication Review*, 18(4) :222–234, August 1988.

[BMR90] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor. In *Proceedings of the 11th Real-Time Systems Symposium*, RTSS '90, pages 182–190, Lake Buena Vista, FL, USA, December 1990.

[BN84] Andrew D. Birell and Bruce J. Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1) :39–59, 1984.

[BOS91] BOSH - Robert Bosch GmbH. *CAN Specification - Version 2.0*, September 1991.

[BR02] Azzedine Boukerche and Amber Roy. Dynamic grid-based approach to data distribution management. *Parallel and Distributed Computing - Parallel and Distributed Discrete Event Simulation : An Emerging Technology*, 62(3) :366–392, March 2002.

[Bra94] Braden, R. and Clark, D. and Shenker, S. *IntServ Integrated Services in the Internet Architecture : an Overview (RFC-1633)*. Network Working Group, July 1994.

[Bra97] Braden, R. and Zhang, L. and Berson, S. and Herzog, S. and Jamin, S. *RSVP : Ressource ReServation Protocol (RFC-2205)*. Internet Engineering Task Force, September 1997.

[Bro97] Gerald Brose. JacORB - A Java Object Request Broker. Technical Report B-97-02, Freie Universität Berlin, 1997.

[Bry77] R. E. Bryant. Simulation of Packet Communication Architecture Computer Systems. Technical Report MIT/LCS/TR-188, MIT - Massachusetts Institute of Technology, Cambridge, MA, USA, 1977.

[BS02] Benoit Bréholée and Pierre Siron. CERTI : Evolutions of the ONERA RTI Prototype. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 2002. 02F-SIW-018.

[BS03a] Benoit Bréholée and Pierre Siron. Bridges in HLA distributed simulation. In *Proceedings of Industrial Simulation Conference*, ISC '03, Valencia, Spain, June 2003.

[BS03b] Benoit Bréholée and Pierre Siron. Design and Implementation of a HLA Interfederation Bridge. In *Proceedings of the European Simulation Interoperability Workshop*, Stockholm, Sweden, June 2003. 03E-SIW-054.

[BSZ07] Azzedine Boukerche, Ahmad Shadid, and Ming Zhang. Efficient Load Balancing Schemes for Large-Scale Real-Time HLA/RTI Based Distributed Simulations. In *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '07, pages 103–112, Washington, DC, USA, 2007. IEEE Computer Society.

[Bun59] Mario Augusto Bunge. *Causality. The place of the Causal Principle in Modern science.* Cambridge Harvard University Press, 1959.

[Bur85] John C. Burruss. Status of OSI standards. *ACM SIGCOMM Computer Communication Review Newsletter*, 15(5) :77–80, October 1985.

[But30] Theodore Butterworth. On the Theory of Filter Amplifiers. *Experimental, Wireless Engineer Journal*, 7 :536–541, 1930.

[BW98] Alan Burns and Andy Wellings. *Concurrency in Ada (2nd ed.).* Cambridge University Press, New York, NY, USA, 1998.

[BW01] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages.* Addison Wesley, 2001.

[CA95] Sheng-Tzong Cheng and Ashok K. Agrawala. Allocation and scheduling of real-time periodic tasks with relative timing constraints. Technical Report UMIACS-TR-95-6, University of Maryland - Institute for Advanced Computer Studies, College Park, MD, USA, 1995.

[Cal93] Calvin, James M. and Dickens, Alan and Gaines, Bob and Metzger, Paul and Miller, Dale and Owen, Dan. The Simnet Virtual World Architecture. In *VR*, pages 450–455, 1993.

[Cal09] Caleb, Gordon. *Introduction to IEEE 1588 & Transparent Clocks.* TEKRON International, White Paper edition, 2009.

[Can99] Dominique Canazzi. yarti, an ada 95 hla run-time infrastructure. In *Proceedings of the 1999 Ada-Europe International Conference on Reliable Software Technologies*, Ada-Europe '99, pages 187–198, 1999.

[Can01] Pascal Cantot. *Introduction à la Simulation.* École Nationale Supérieure des Ingénieurs et des Études et techniques d'Armement, Septembre 2001.

[Car02] Martin Carlsson. Worst Case Execution Time Analysis, Case Study on Interrupt Latency, For the OSE Real-Time Operating System. Master's thesis, Royal Institute of Technology, Stockholm, Sweden, 2002.

[Car05] Carter, Paul.A. *Langage assembleur PC*, mars 2005. Traduit de l'anglais par Sébastien Le Ray.

[CB01] Suk Kim Chin and R. Braun. A survey of UDP packet loss characteristics. In *Proceedings of the Conference on Signals, Systems and Computers*, pages 200–204, Pacific Grove, CA, USA, November 2001.

[CBB05] Kendall Correll, Nick Barendt, and Michael Branicky. Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol. In *Proceedings of the Conference on IEEE 1588 Standard for Networked Measurement and Control Systems*, 2005.

[CCGZ90] Ching-Tsun Chou, Israel Cidon, Inder S. Gopal, and Shmuel Zaks. Synchronizing asynchronous bounded delay networks. *IEEE Transactions on Communications*, pages 144–147, 1990.

[CCM+97] J. Calvin, C. Chiang, S. McGarry, S. Rak, D. Van Hook, and Marnie Salisbury. Design Implementation and Performance of the STOW RTI Prototype. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 1997.

[CES03] CES - Creative Electronic Systems S.A. *White Paper on AFDX*, DOC 8854/W Version 1.0 edition, November 2003.

[CFWW97] Christopher D. Carothers, Richard M. Fujimoto, Richard M. Weatherly, and Annette L. Wilson. Design and implementation of HLA time management in the RTI version F.0. In *Proceedings of the 29th conference on Winter simulation*, WSC '97, pages 373–380, 1997.

[Cha07] Hussein Charara. *Évaluation des performances temps réel des réseaux embarqués avioniques*. PhD thesis, Institut de Recherche en Informatique de Toulouse, Toulouse, France, Novembre 2007.

[CK74] Vinton G. Cerf and Robert E. Khan. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 22(5) :637–648, may 1974.

[CK97] Danny Cohen and Andreas Kemkes. DDM Scenarios. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 1997. 97F-SIW-057.

[CKP+08] Silviu S. Craciunas, Christoph M. Kirsch, Hannes Payer, Ana Sokolova, Horst Stadler, and Robert Staudinger. A compacting real-time memory management system. In *Proceedings of the USENIX Annual Technical Conference on Annual Technical Conference*, USENIX'08, pages 349–362, Boston, MA, USA, June 2008.

[CL85] K. Mani Chandy and Leslie Lamport. Distributed snapshots : determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1) :63–75, February 1985.

[CL87] W.W. Chu and L.M. Lan. Task Allocation and Precedence Relations for Distributed Real-Time Systems. *IEEE Transactions on Computers*, pages 667–679, June 1987.

[CL90] Min-Ih Chen and Kwei-Jay Lin. Dynamic Priority Ceilings : a Concurrency Control Protocol for Real-Time Systems. *The Journal of Real-Time Systems*, 2(4) :325–346, October 1990.

[CL94] Sandra Cheung and Margaret Loper. Synchronizing simulations in Distributed Interactive Simulation. In *Proceedings of the 26th conference on Winter simulation*, WSC '94, pages 1316–1323, 1994.

[CLG90] William M. Corwin, Douglass C. Locke, and Karen D. Gordon. Overview of the IEEE POSIX P1003.4 realtime extension to POSIX. *IEEE Real-Time Systems Newsletter*, 6(1) :9–18, March 1990.

[CM79] K.M. Chandy and J. Misra. Distributed simulation : A case study in design and verification of distributed programs. *Software Engineering, IEEE Transactions on*, SE-5(5) :440–452, Sept. 1979.

[CM81] K. M. Chandy and J. Misra. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Communications of the ACM Magazine - Special issue on simulation modeling and statistical computing*, 24(4) :198–206, 1981.

[CM97] Duncan Clark and Gerry Magee. UKRTI - Interface Design Requirements and Implementation Architecture. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 1997. 97F-SIW-036.

[CMH83] K. Mani Chandy, Jayadev Misra, and Laura M. Haas. Distributed Deadlock Detection. *ACM Transactions on Computer Systems (TOCS) Journal*, 1(2) :144–156, May 1983.

[CMR05] Alfons Crespo, Miguel Masmano, and Ismael Ripoll. *Description of the TLSF Memory Allocator Version 2.0.* Grupo de informática Industrial, Universidad Politecnica de Valencia, November 2005.

[CMR06] Alfons Crespo, Miguel Masmano, and Ismael Ripoll. Dynamic Memory Management for Embedded Real-Time Systems. In *Proceedings of the Working Conference on Distributed and Parallel Embedded Systems*, 2006.

[Com94] DIS Steering Commitee. The DIS Vision : A Map to future of Distributed Simulation. Technical report, institute for Simulation and Training, Orlando, Florida, 1994.

[Con94] U.S. Congress. Virtual Reality and Technologies for Combat Simulation - Background paper. Technical Report OTA-BP-ISS-136, Office of Technology Assessment, September 1994.

[Con01] Concurrent Computer Corporation. *Real-Time Clock and Interrupt Module (RCIM) User's Guide*, August 2001.

[Con03] Concurrent Computer Corporation. *RedHawk NightStar Tools Tutorial User's Guide*, December 2003.

[Cor05] Angelo Corsaro. CARDAMOM : A Next Generation Mission and Safety Critical Enterprise Middleware. In *Proceedings of the Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, pages 73–74, 2005.

[Cot96] Cottet, F. and Saad-Bouzefrane, S. Synthèse bibliographique sur le réseau de terrain FIP, comparaison avec les autres protocoles déterministes. Technical Report 95.003, École Nationale Supérieure de Mécanique et d'Aéronautique - Laboratoire d'Informatique Scientifique et Industrielle (ENSMA - LISI), Avril 1996.

[CPRS02] A. Colin, I. Puaut, C. Rochange, and P. Sainrat. Calcul de majorants de pire temps d'exécution : État de l'art. Technical Report 1461, IRISA - Institut de Recherche en Informatique et Systèmes Aléatoires, Mai 2002.

[CPW+96] Bradford S. Canova, Ernest H. Page, Richard M. Weatherly, Annette L. Wilson, Anita A. Zabek, and Mary C. Fischer. Advanced Distributed Simulation through the Aggregate Level Simulation Protocol. In *Proceedings of the 29th Hawaii International Conference on System Sciences (Volume 1 : Software Technology and Architecture)*, HICSS '96, pages 407–415, Maui, Hawaii, USA, January 1996.

[Cri89] F. Cristian. A probabilistic approach to distributed clock synchronization. In *Proceedings of the 9th International Conference on Distributed Computing Systems*, ICDCS '89, pages 288–296, Newport Beach, CA, USA, June 1989.

[CRJ06] Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 101–110, Washington, DC, USA, 2006.

[CS92] Reuven Cohen and Adrian Segall. Distributed priority algorithms under one-bit-delay constraint. In *Proceedings of the eleventh annual ACM symposium on Principles of distributed computing*, PODC '92, pages 1–12, Vancouver, British Columbia, Canada, August 1992.

[CSA10] Jean-Baptiste Chaudron, Pierre Siron, and Martin Adelantado. Towards an HLA Run-time Infrastructure with Hard Real-time Capabilities. In *Proceedings of the International Simulation Multi-Conference*, ISMC'10, Ottawa, Canada, July 2010.

[CSAN11] Jean-Baptiste Chaudron, Pierre Siron, Martin Adelantado, and Eric Noulard. HLA high performance and real-time simulation studies with CERTI. In *Proceedings of the European Simulation and Modelling Conference*, Guimaraes, Portugal, October 2011. HPCN_04.

[CSB90] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *The Journal of Real-Time Systems*, 2(3) :181–194, September 1990.

[CSN11] Jean-Baptiste Chaudron, Pierre Siron, and Eric Noulard. Design and modeling techniques for real-time RTI time management. In *Proceedings of the Spring Simulation Multiconference*, SpringSim'11, 2011.

[CSSA11] Jean-Baptiste Chaudron, David Saussie, Pierre Siron, and Martin Adelantado. Real Time Aircraft Simulation using HLA standard - An overview. In *Proceedings of the First Simulation in Aerospace Conference*, Toulouse, France, April 2011.

[CSSA12] Jean-Baptiste Chaudron, David Saussie, Pierre Siron, and Martin Adelantado. Real Time Aircraft Simulation using HLA standard. *Simulation - Special Issue on Modeling and Simulation of Aeronautical Systems*, 2012. *Under submission process.*

[CT95] Wentong Cai and Stephen J. Turner. An algorithm for reducing null-messages of cmb approach in parallel discrete event simulation. Technical report, University of Exeter - Department of computer Science, 1995.

[CTB94] Chia-Mei Chen, Satish K. Tripathi, and Alex Blackmore. A Resource Synchronization Protocol for Multiprocessor Real-Time Systems. In *Proceedings of the International Conference on Parallel Processing*, ICPP '94, pages 159–162, North Carolina State University, NC, USA, August 1994.

[CWP97] Andy Cox, Douglas Wood, and Mikel Petty. Gateway Performance Using RTI 1.0. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 1997. 97F-SIW-072.

[DA49] Hugh L. Dryden and Ira H. Abbott. The Design of Low-Turbulence Wind Tunnels. Technical Report Report 940, National Advisory Committee For Aeronautics, Washington, D.C, USA, 1949.

[DB11] Robert I. Davis and Alan Burns. A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems. *ACM Computing Surveys*, 43(4) :35 :2–35 :44, October 2011.

[DBBL07] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller Area Network (CAN) schedulability analysis : Refuted, Revisited and Revised. *The Journal of Real-Time Systems*, 35(3) :239–272, April 2007.

[dBS02] B. d'Ausbourg, J.-L. Bussenot, and P. Siron. Perfosim : A performance evaluation tool for hla distributed simulations. In *Proceedings of the Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, DS-RT '02, pages 23–30, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[Dec03] David Decotigny. *Une infrastructure de simulation modulaire pour l'évaluation de performances de système temps réel.* PhD thesis, Université de Rennes 1, Avril 2003.

[Der74] Michael L. Dertouzos. Control Robotics : The Procedural Control of Physical Processes. In *Proceedings of the Information Processing Conference*, pages 807–813, Stockholm, Sweden, 1974.

[Des97] Patrick Desseaux. Prototypage d'un système de simulation distribuée compatible avec HLA. Master's thesis, Université Paul Sabatier Toulouse III, Toulouse, France, 1997.

[Dij68] Edsger W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages : NATO Advanced Study Institute*, pages 43–112. Academic Press, 1968.

[DL91] M. Dao and Kwei-Jay Lin. Remote Procedure Call protocols for real-time systems. In *Proceedings of Workshop on Real Time Systems, Euromicro '91.*, pages 216 –223, june 1991.

[DLA06] UmaMaheswari C. Devi, Hennadiy Leontyev, and James H. Anderson. Efficient Synchronization under Global EDF Scheduling on Multiprocessors. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, ECRTS '06, pages 75–84, Dresden, Germany, July 2006.

[DM89] M. L. Dertouzos and A. K. Mok. Multiprocessor On-line Scheduling of Hard-Real-Time Tasks. *IEEE Transactions on Software Engineering*, 15(12) :1497–1506, December 1989.

[DoD95] DoD - Departement of Defence. *Modeling and Simulation (M&S) Master Plan (DoD 50000.59-P).* Under Secretary of Defense for Acquisition and Technology, 1995.

[DoD98a] DoD - Departement of Defence. *High Level Architecture (HLA) - Framework and Rules.* Standard for Modeling and Simulation, 1998.

[DoD98b] DoD - Departement of Defence. *High Level Architecture Interface Specification - Version 1.3.* Standard for Modeling and Simulation, 1998.

[DoD98c] DoD - Departement of Defence. *High Level Architecture Object Model template Specification - Version 1.3.* Standard for Modeling and Simulation, 1998.

[DoD99] DoD - Department of Defence. *High Level Architecture Federation Execution Details (FED) File Specification*, v1.3 edition, 1999.

[DoD00] DoD - Departement of Defence. *RTI 1.3 - Next Generation Programmer's Guide Version 3.2*, September 2000.

[Dow97] Gilles. Dowek. Le langage mathématique et les langages de programmation. In *Voir, Entendre et Calculer*, Cité des sciences et de l'industrie, Paris, France, Juin 1997.

[DS80] E. W. Dijkstra and C. S. Sholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1), 1980.

[DTB93] R.I. Davis, K.W. Tindell, and A. Burns. Scheduling Slack Time in Fixed Priority Preemptive Systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS '93, pages 222 –231, Raleigh-Durham, NC, USA, December 1993.

[Dub93] Lydia P. Dubon. Joining a distributed simulation environment via ALSP. In *Proceedings of the 25th conference on Winter simulation*, WSC '93, pages 1073–1076, Los Angeles, CA, USA, December 1993.

[DWE+01] Lisa Cingiser Dipippo, Victor Fay Wolfe, Levon Esibov, Gregory Cooper Ramachandra Bethmangalkar, Ramachandra Bethmangalkar, Russell Johnston, Bhavani Thuraisingham, and John Mauer. Scheduling and Priority Mapping for Static Real-Time Middleware. *The Journal of Real-Time Systems*, 20(2) :155–182, 2001.

[EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of Publish/Subscribe. *ACM Computing Surveys Journal*, 35(2) :114–131, 2003.

[Ell94a] Ahmed Elleuch. *Migration de Processus dans les Systèmes Massivement Parallèles.* PhD thesis, Institut National Polytechnique de Grenoble, Novembre 1994.

[Ell94b] Stephen R. Ellis. What are virtual environments ? *IEEE Computer Graphics and Applications*, 14(1) :17–22, January 1994.

[Elm86] Ahmed K. Elmagarmid. A survey of distributed deadlock detection algorithms. *ACM SIGMOD Record Newsletter*, 15(3) :37–45, September 1986.

[FB09] Nathan Fisher and Sanjoy Baruah. The feasibility of general task systems with precedence constraints on multiprocessor platforms. *The Journal of Real-Time Systems*, 41(1) :1–26, January 2009.

[FBC09] F. Faggioli, M. Bertogna, and F. Checconi. An EDF scheduling class for the Linux kernel. In *Proceedings of the 11th Real-Time Linux Workshop*, October 2009.

[FBG+10] Julien Forget, Frédéric Boniol, Emmanuel Grolleau, David Lesens, and Claire Pagetti. Scheduling dependent periodic tasks without synchronization mechanisms. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '10, Stockolm, Sweden, April 2010.

[Fed03] Federal Aviation Administration. *Software Approval Guidelines.* U.S. Department of Transportation, June 2003.

[FF99] Steve L. Ferenci and Richard Fujimoto. RTI Performance on Shared Memory and Message Passing Architectures. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 1999.

[FGH06] Peter H. Feiler, David P. Glush, and John J. Hudak. The Architecture Analysis & Design Language (AADL) : An Introduction. Technical Report CMU/SEI-2006-TN-011, Carnegie Mellon University, 2006.

[Fit02] Fitzgibbons, Brad and McLean, Thom and Fujimoto, Richard. RTI Benchmark Studies. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 2002. 02S-SIW-105.

[FK03] Pierre Ficheux and Patrice Kadionik. Temps Réel sous LINUX. *Linux Magazine*, 52, 2003.

[FMPT00] Richard Fujimoto, Thom McLean, Kalyan Perumalla, and Ivan Tacic. Design of High Performance RTI Software. In *Proceedings of the Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, DS-RT '00, page 89, Washington, DC, USA, 2000. IEEE Computer Society.

[FN69] W. D. Farmer and E. E. Newhall. An experimental distributed switching system to handle bursty computer traffic. In *Proceedings of the first ACM Symposium on Problems in the Optimization of Data Communications Systems*, pages 1–3, Pine Mountain, Georgia, United States, 1969.

[FN98] W. D. Farmer and E. E. Newhall. A Comparison Study Between TDMA and FDMA in Digital Wireless Systems. In *Issues of the Spread Spectrum Scene - The Wireless, PCs and Digitals communications*, volume 6, 1998.

[For09] Forget, Julien. *Un langage synchrone pour les systèmes embarqués critiques soumis à des contraintes temps réel multiples.* PhD thesis, Université de Toulouse, Toulouse, France, 2009.

[Fra05] Sephen. J. Franz. *LynxOS RTOS (Real-Time Operating System)*, 2005.

[Fuj90] Richard Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM - Special issue on Simulation*, 33 :30–53, October 1990.

[Fuj97] Richard M. Fujimoto. Zero Lookahead And Repeatability In The High Level Architecture. In *Proceedings of the Spring Simulation Interoperability Workshop*, pages 3–7, Orlando, FL, USA, March 1997. 97S-SIW-046.

[Fuj98] Richard M. Fujimoto. Time Management in the High Level Architecture. *Simulation*, 71, 1998.

[Gaf88] A. Gafni. Rollback mechanisms for optimistic distributed simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 61 –67, Febuary 1988.

[GAGB01] Paolo Gai, Luca Abeni, Massimiliano Giorgi, and Giorgio Buttazzo. A New Kernel Approach for Modular Real-Time Systems Development. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, ECRTS '01, Washington, DC, USA, 2001. IEEE Computer Society.

[Gau05] François Gauthier. Les compilateurs C/C++ pour l'embarqué. *Électronique Mensuel*, 155 :74–80, 2005.

[GB00] James Gosling and Greg Bollella. *The Real-Time Specification for Java.* Addison-Wesley Longman Publishing Co, Inc., Boston, MA, USA, 2000.

[Geo05] Jean-Philippe Georges. *Systèmes contrôlés en réseau : Évaluation de performance d'architecture Ethernet commutés.* PhD thesis, Université Henry Poincaré, Nancy, France, Novembre 2005.

[Ger04] Philippe Gerum. Xenomai, Implementing a RTOS emulation framework on GNU/-LINUX. White Paper, 2004.

[GJSB05] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java(TM) Language Specification, The 3rd Edition.* Addison-Wesley, 2005.

[GLN01] Paolo Gai, Giuseppe Lipari, and Marco Di Natale. Minimizing Memory Utilization of Real-Time Task Sets in Single and Multi-Processor Systems-on-a-Chip. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, RTSS '01, London, UK, December 2001.

[GLS99] Christopher D. Gill, David L. Levine, and Douglas C. Schmidt. The Design and Performance of a Real-Time CORBA Scheduling Service. *The International Journal of Time-Critical Computing Systems - Sepcial Issue on Real-time Middleware*, 20(2), March 1999.

[GNRS03] A.H. Goktogan, E. Nettleton, M. Ridley, and S. Sukkarieh. Real Time Multi-UAV Simulator. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2 of *ICRA '03*, pages 2720 – 2726, Taipei, Taiwan, september 2003.

[GPFF97] Sean.P. Griffin, Ernest H. Page, Zachary Furness, and Mary Fischer. Providing Uninterrupted Training to the Joint Training Confederation (JTC) Audience During Transition to the High Level Architecture (HLA). In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 1997. 97F-SIW-147.

[GRHR07] Tarek Guesmi, Rojdi Rekik, Salem Hasnaoui, and Houria Rezig. Design and Performance of DDS-based Middleware for Real-Time Control Systems. *International Journal on Computer Science and Network Security*, December 2007.

[GS88] J. B. Goodenough and L. Sha. The Priority Ceiling Protocol : A method for minimizing the blocking of high priority Ada tasks. *ACM Ada Letters - Special Edition : International Workshop on Real-Time Ada Issues*, VIII(7) :20–31, June 1988.

[GS96] Aniruddha Gokhale and Douglas C. Schmidt. Measuring the Performance of Communication middleware on High-Speed Networks. *ACM SIGCOMM Computer Communication Review*, 26(4) :306–317, August 1996.

[GS97] A. S. Gokhale and D. C. Schmidt. Evaluating CORBA latency and scalability over high-speed ATM networks. In *Proceedings of the 17th International Conference on Distributed Computing Systems*, ICDCS '97, 1997.

[GS98] Aniruddha S. Gokhale and Douglas C. Schmidt. Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks. *IEEE Transactions on Computer*, 47(4) :391–413, April 1998.

[GSHP96] A. Gokhale, D. C. Schmidt, T. Harrison, and G. Parulkar. Operating System Support for High-performance, Real-time CORBA. In *Proceedings of the 5th International Workshop on Object Orientation in Operating Systems*, IWOOOS '96, 1996.

[GSMT97] O. Gonzalez, C. Shen, I. Mizunuma, and M. Takegaki. Implementation and Performance of MidART. In *Proceedings of IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, San Francisco, CA, USA, 1997. IEEE.

[Gui05] P.Y. Guidotti. Entrées CNES sur les besoins de démonstration HLA. Technical Report DVF-CR-BV-AI-NS-144-CN, Centre National d'Etudes Spatiales (CNES), 2005.

[HA08] Hassen Jawhar Hadj-Amor. *Contribution au prototypage virtuel de systèmes mécatroniques basé sur une architecture distribué HLA. Expérimentation sous les environnements OpenModélica-OpenMASK.* PhD thesis, Université du Sud-Toulon-Var, Toulon, France, 2008.

[Hai83] R. Hainich. Backoff Strategies for CSMA/CD with Real-Time Applications. In *Kommunikation in Verteilten Systemen - Anwendungen und Betrieb, GI/ITG-Fachtagung*, volume 60, pages 671–691, Berlin, Germany, January 1983.

[Har93] Michael Gonzalez Harbour. Real-Time POSIX : An Overview. Technical report, Departamento de Electrónica, Universidad de Cantabria, 1993.

[HAR94] E. S. H. Hou, N. Ansari, and H. Ren. A Genetic Algorithm for Multiprocessor Scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2) :113–120, Febuary 1994.

[HC05] Yusuf Hasan and Morris Chang. A study of Best-Fit memory allocators. *Computer Languages, Systems and Structures*, 31(1) :35 – 48, 2005.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9) :1305–1320, September 1991.

[HFP+00] Ding He, Ding He Fuhu, Dave Pape, Greg Dawe, and Dan S. Video-Based Measurement of System Latency. In *International Immersive Projection Technology Workshop*, 2000.

[HGCD00] Wolfgang A. Halang, Roman Gumzej, Matjaz Colnaric, and Marjan Druzovec. Measuring the Performance of Real-Time Systems. *The International Journal of Time-Critical Computing Systems*, 18(1) :59–68, January 2000.

[HHK03] T.A. Henzinger, B. Horowitz, and C.M. Kirsch. GIOTTO : a time-triggered language for embedded programming. In *Proceedings of the First International Workshop on Embedded Software*, volume 91, pages 84–99, January 2003.

[Hil92] Dan Hildebrand. An Architectural Overview of QNX. In *Proceedings of the Workshop on Micro-kernels and Other Kernel Architectures*, pages 113–126, Berkeley, CA, USA, 1992.

[HKK07] Jeong Hee Hong, Jae Hyun Kim, and Tag Gon Kim. Design and implementation of Time Management service for IEEE 1516 HLA/RTI. In *Proceedings of the Summer Computer Simulation Conference*, SCSC '07, pages 379–385, 2007.

[HLS97] Timothy H. Harrison, David L. Levine, and Douglas C. Schmidt. The design and performance of a real-time CORBA event service. *ACM SIGPLAN Notification*, 32(10) :184–200, 1997.

[HMPM01] R. Herzog, J. Mulder, K. Pixius, and H.-P. Menzler. HLA on top of CORBA Common Object Services. In *Proceedings of European Simulation Interoperability Workshop*, June 2001.

[HNB97] F. Hodum, J. Noseworthy, and S. Bachinsky. Implementation of the Next Generation RTI. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 1997.

[Hol75] J. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

[Hor94] Horowitz, Benjamin. *Giotto : A Time-Triggered Language for Embedded Programming.* PhD thesis, University of California, Bekerley, CA, USA, 1994.

[Hou90] Catherine Houstis. Module Allocation of Real-Time Applications to Distributed Systems. *IEEE Transactions on Software Engineering*, 16(7), July 1990.

[HRX08] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC : a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review - Special Issue on research and developments in the Linux kernel*, 42(5) :64–74, July 2008.

[HS94] CH.J. Hou and K.G. Shin. Replication and Allocation of Task Modules in Distributes Real-Time Systems. In *Proceedings of the 24th Symposium on Fault-Tolerant Computing*, Austin, Texas, USA, June 1994.

[HS08] Jamal Habibullah and Kiran Sultan. Performance Analysis of TCP Congestion Control Algorithms. *International Journal of Computers and Communications*, 2(1), 2008.

[HSG09] Joe Hoffert, Douglas C. Schmidt, and Aniruddha Gokhale. Evaluating Transport Protocols for Real-Time Event Stream Processing Middleware and Applications. In *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE on the Move to Meaningful Internet Systems*, OTM '09, pages 614–633, 2009.

[HTHL05] Jiung-yao Huang, Ming-Chih Tung, Lin Hui, and Ming-Che Lee. An Approach for the Unified Time Management Mechanism for HLA. *Simulation Journal*, 81 :45–56, January 2005.

[HU01] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 2001.

[Hug05] Jérôme Hugues. *Architecture et Services des Intergiciels Temps Réel.* PhD thesis, École Doctorale d'Informatique, Télécommunications et Électronique de Paris, Septembre 2005.

[Hui03] Huitema, C. *Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP) (RFC-3605).* Network Working Group, October 2003.

[HWTC01] Bill Helfinstine, Deborah Wilbert, Marc Torpey, and Wayne Civinskas. Experiences with Data Distribution Management in Large-Scale Federations. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, Fl, USA, September 2001. 01F-SIW-032.

[Hä08] Johan Hägg. Scheduling Aperiodic Tasks Using Slack Stealing. Technical report, School of Innovation, Design and Engineering, Västeras, Sweden, December 2008.

[IEC00] IEC - The International Engineering Consortium. *Time Division Multiple Access (TDMA)*, web proforum tutorials edition, 2000.

[IEE93a] IEEE - The Institute of Electrical and Electronics Engineers. *1278-1993 : IEEE Standard for Information technology - Protocols for Distributed Interactive Simulation Applications*, March 1993.

[IEE93b] IEEE - The Institute of Electrical and Electronics Engineers (IEEE) Computer Society. *POSIX, Real-time extensions*, 1003.1b-1993 edition, 1993.

[IEE95a] IEEE - The Institute of Electrical and Electronics Engineers. *1278.1-1993 : IEEE Standard for Distributed Interactive Simulations - Application Protocols*, September 1995.

[IEE95b] IEEE - The Institute of Electrical and Electronics Engineers. *1278.2-1993 : IEEE Standard for Distributed Interactive Simulations - Communication Services and Profiles*, September 1995.

[IEE95c] IEEE - The Institute of Electrical and Electronics Engineers (IEEE) Computer Society. *POSIX, Threads extensions*, 1003.1c-1995 edition, 1995.

[IEE98] IEEE - The Institute of Electrical and Electronics Engineers. *Part 5 : Token ring access method and physical layer specifications*, ANSI/IEEE Std 802.5 - 1998 edition, May 1998.

[IEE00a] IEEE - The Institute of Electrical and Electronics Engineers. *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*, ISO/IEC 8802-3 - IEEE 802.3 edition, December 2000.

[IEE00b] IEEE - The Institute of Electrical and Electronics Engineers Computer Society. *Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification.* Simulation Interoperability Standards Committee, September 2000.

[IEE00c] IEEE - The Institute of Electrical and Electronics Engineers Computer Society. *Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules.* Simulation Interoperability Standards Committee, September 2000.

[IEE00d] IEEE - The Institute of Electrical and Electronics Engineers Computer Society. *Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification.* Simulation Interoperability Standards Committee, September 2000.

[IEE02] IEEE - The Institute of Electrical and Electronics Engineers. *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std 1588 - 2002 edition, November 2002.

[IEE03] IEEE - The Institute of Electrical and Electronics Engineers Computer Society. *IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP)*, April 2003.

[IEE07] IEEE - The Institute of Electrical and Electronics Engineers Computer Society. *IEEE Recommended Practice for Verification, Validation, and Accreditation of a Federation - An Overlay to the High Level Architecture Federation Development and Execution Process* , September 2007.

[Inc95] Sun Microsystems Inc. *RPC : Remote Procecure Call - Protocol Specification - Version 2 (RFC-1831).* Network Working Groupe, August 1995.

[Inf81] Information Sciences Institute University of Southern California. *Internet Protocol - Darpa Internet Program Protocol Specification (RFC - 791).* Defense Advanced Research Projects Agency Information Processing Techniques Office, September 1981.

[Inn02] Real-Time Innovations. Real-Time Publish-Subscribe (RTPS) Wire Protocol Specification - Protocol Version 1.0. Technical report, Real-Time Innovations, Inc., Sunnyvale, CA 94089, 2002.

[IRA88] Philippe Ingels, Michel Raynal, and Michel Adam. Algorithmes Distribués synchrones et systèmes répartis asynchrones : Concepts, mises en oeuvre et expérimentations. Technical Report RR-0862, INRIA - Institut National de Recherche en Informatique et Automatique, Juillet 1988.

[IS01] Jean-Louis Igarza and Christian Sautereau. Distribution, Use and Reuse : Questioning the cost effectiveness of re-using simulations with and without HLA. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 2001. 01F-SIW-002.

[ISO89] ISO - International Organization for Standardization, Genève, Switzerland. *Information processing systems — Fibre Distributed Data Interface (FDDI) — Part 2 : Token Ring Media Access Control (MAC)*, iso 9314-2 edition, 1989.

[ISO99] ISO - International Standard Organization. *Road vehicles – Controller area network (CAN) – Part 1 : Data link layer and physical signalling*, ISO/CD 11989 edition, 1999.

[Jac55] Jackson, James.R. Scheduling a production line to minimize maximum tardiness. Technical Report 43, Management Sciences Research Project, University of California, Los Angeles, July 1955.

[Jac88] V. Jacobson. Congestion Avoidance and Control. *ACM SIGCOMM Computer Communication Review*, 18(4) :314–329, August 1988.

[Jan98] J.W. Janneck. Generalizing lookahead-behavioral prediction in distributed simulation. In *Proceedings of 12' Workshop on Parallel and Distributed Simulation, 1998. PADS 98.*, pages 12 –19, may. 1998.

[Jan04] Jansen, Roger and Huiskamp, Wim and Boomgaardt, Jan-Jell and Brassé, Marco. Real-Time Scheduling of HLA Simulator Components. In *Proceedings of the Spring Simulation Interoperability Workshop*, Arlington, VA, USA, April 2004. 04S-SIW-30.

[JBW+87] D. Jefferson, B. Beckman, F. Wieland, Blu L., and M. Diloreto. Time Warp Operating System. *SIGOPS Opererating System Review*, 21(5) :77–93, November 1987.

[JC09] Rajive Joshi and Gerardo-Pardo Castellote. *A Comparison and Mapping of Data Distribution Service and High Level Architecture.* Real-Time Innovations, Inc. (RTI), White Paper edition, 2009.

[Jef85] David Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7 :404–425, 1985.

[JH05] Kim Jae-Hyun. *Proposal of High Level Architecture Extension and Run-Time Infrastructure Implementation.* PhD thesis, Korea Advanced Institute of Science and

Technology, Departement of Electrical Engineering and Computer Science, Daejon, Korea, 2005. Director-Tag Gon, Kim.

[JP86] M. Joseph and P. Pandya. Finding response times in a real-time system. In *BCS Computer Journal*, pages 390–395, 1986.

[JSM91] K. Jeffay, D.F. Stanat, and C.U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 129 –139, San Antonio, TX, USA, December 1991.

[JW98] Mark S. Johnstone and Paul R. Wilson. The memory fragmentation problem : solved ? In *Proceedings of the 1st International Symposium on Memory Management*, ISMM '98, pages 26–36, Vancouver, British Columbia, Canada, 1998.

[Kar98] Karlsson, Mikael and Lof, Stafan and Loftrand, Bjorn. Experiences from Implementing a RTI in Java. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 1998. 98S-SIW-062.

[KCL01] Pamela Knight, Aaron Corder, and Ron Liedel. Independent Throughput and Latency Benchmarking for the Evaluation of RTI Implementations. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 2001. 01F-SIW-033.

[KCS06] A. Konak, D.W. Coit, and A.E. Smith. Multi-objective optimization using genetic algorithms : A tutorial. *Reliability Engineering & System Safety*, 91(9) :992–1007, January 2006.

[Ker88] Kernighan, B.W and Ritchie, D.M. *The C programming Language*. Prentice-Hall, London, UK, 1988.

[KG94] Hermann Kopetz and Günter Grünsteidl. TTP-A Protocol for Fault-Tolerant Real-Time Systems. *Computer*, 27(1) :14–23, January 1994.

[Kir83] Kirpatrick, S. and Gelatt, C.D. and Vecchi, M.P. Optimization by Simulated Annealing. *Science Journal*, 220(4598) :671–680, May 1983.

[KK04] Jae-hyun Kim and Tag Gon Kim. Proposal of High Level Architecture Extension. In *Artificial Intelligence and Simulation*, pages 128–137, 2004.

[KKS02] Raymond Klefstad, Arvind S. Krishna, and Douglas C. Schmidt. Design and Performance of a Modular Portable Object Adapter for Distributed, Real-Time and Embedded CORBA Applications. In *Proceedings of Proceedings of the Distributed Objects and Applications*, DOA '02, Irvine, CA, USA, November 2002.

[KN80] K. H. Kim and Mahmoud Naghibzadeh. Prevention of task overruns in real-time non-preemptive multiprogramming systems. *ACM SIGMETRICS Performance Evaluation Review*, 9(2) :267–276, May 1980.

[KO87] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers - Special Issue on Real-Time Systems*, 36(8) :933–940, August 1987.

[KO01] Mikael Karlsson and Lennart Olsson. pRTI 1516 – Rationale and Design. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 2001. 01F-SIW-038.

[KO06] Mikael Karlsson and Lennart Olsson. EODiSP - An Open and Distributed Simulation Platform. In *Proceedings of the 9th International Workshop on Simulation for European Space Programmes*, SESP 2006, Noordwijk, the Netherlands, November 2006.

[Kol06] Samad S. Kolahi. Comparison of FDMA. TDMA and CDMA system capacities. In *Proceedings of the 10th WSEAS International Conference on Communications*, ICCOM'06, pages 333–336, Athens, Greece, 2006.

[Kop11] Hermann Kopetz. *Real-Time Systems : Design Principles for Distributed Embedded Applications*. Springer, second edition, 2011.

[KRK03] Arvind S. Krishna, Douglas C. Schmidt Krishna Raman, and Raymond Klefstad. Optimizing the ORB core to enhance Real-time CORBA predictability and performance. In *Proceedings of Distributed Objects and Applications Workshop*, DOA '03, Catania, Sicily (Italy), November 2003.

[KRSS98] O. Gonzáles K. Ramamritha, C. Shen, S. Sen, and S. Shirgurkar. Using Windows NT for Real-Time Applications : Experimental Observations and Recommendations. In *Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium*, pages 102–, Washington, DC, USA, 1998. IEEE Computer Society.

[KSY84] James F. Kurose, Mischa Schwartz, and Yechiam Yemini. Multiple-Access Protocols and Time-Constrained Communication. *ACM Computing Surveys*, 16(1) :43–70, 1984.

[KTS08] A.S Karuppan, B.M. Tresa, and K. Shanthakumari. A New Robust Distributed Real Time Scheduling Services for RT-CORBA Applications. In *Proceedings of the International Conference on Computing, Communication and Networking*, ICCCN '08, 2008.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7) :558–565, 1978.

[Lat05] Jerôme Latour. *DARE-HLA (Design And Runtime Environment for HLA) : un intergiciel pour la conception et la réutilisation de fédérés et fédérations HLA*. PhD thesis, École Nationale Supérieure de l'Aéronautique et de l'Espace (SUPAERO-ISAE) , Toulouse, France, 2005.

[Law73] E.L. Lawler. Optimal Sequencing of a Single Machine Subject to Precedence Constraints. *Management Science*, 19(5) :544–546, January 1973.

[LBT11] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus : a theory of deterministic queuing systems for the internet*. Springer-Verlag, Berlin, Heidelberg, August 2011.

[LDG04] J. M. López, J. L. Díaz, and D. F. García. Utilization bounds for edf scheduling on real-time multiprocessor systems. *The Journal of Real-Time Systems*, 28(1) :39–68, October 2004.

[Le 93] Le Lann, Gérard and Rivierre, Nicolas. Real-time communications over broadcast networks : The CSMA-DCR and the DOD-CSMA-CD protocols. Technical Report RR-1863, Institut National de Recherche en Informatique et Automatique, Mars 1993.

[Lee94] Suk Kyoon Lee. On-line multiprocessor scheduling algorithms for real-time tasks. In *IEEE Region 10's 9th Annual International Conference on Frontiers of Computer Technology*, TENCON '94, pages 607 –611, August 1994.

[Len09] Ricardo Lent. Improving Federation Executions with Migrating HLA/RTI Central Runtime Components. In *Proceedings of IEEE 14th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks*, CAMAD '09, pages 1–5, Pisa, Italy, June 2009.

[LFGGS99] David L. Levine, Sergio Flores-Gaitan, Christopher D. Gill, and Douglas C. Schmidt. Measuring OS Support for Real-time CORBA ORBs. In *IEEE International Workshop on Object-oriented Real-time Dependable Systems*, WORDS'99, January 1999.

[LFGS99] David L. Levine, Sergio Flores-Gaitan, and Douglas C. Schmidt. An Empirical Evaluation of OS Endsystem Support for Real-time CORBA Object Request Brokers. *Real-Time Technology and Applications Symposium*, June 1999.

[LGDG00] J. M. López, M. García, J. L. Díaz, and D. F. García. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proceedings of the 12th Euromicro conference on Real-time systems*, Euromicro-RTS'00, pages 25–33, Stockholm, Sweden, 2000.

[LHM+97] Rodger Lea, Yasuaki Honda, Kouichi Matsuda, Olof Hagsand, and Marten Stenius. Issues in the design of a scalable shared virtual environment for the Internet. In *Proceedings of the 30th Hawaii International Conference on System Sciences*, HICSS '97, page 653, 1997.

[Liu00] Jane W.S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[LJB03] Stefan Lankes, Andreas Jabs, and Thomas Bemmerl. Integration of a CAN-Based Connection-Oriented Communication Model into Real-Time CORBA. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, Washington, DC, USA, 2003. IEEE Computer Society.

[LJR02] S. Lankes, A. Jabs, and M. Reke. A time-triggered Ethernet protocol for Real-Time CORBA. In *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, ISORC '02, pages 215 –222, Washington, DC, USA, May 2002.

[LK99] Mohammed Ali Liviani and Jörg Kaiser. Evaluation of a Hybrid Real-Time Bus Scheduling Mechanism for CAN. In *Proceedings of the 11 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, IPPS/SPDP'99, pages 425–429, San Juan, Puerto Rico, April 1999.

[LKV02] Arjan Lemmers, Paul Kuiper, and René Verhage. Performance of a component-based flight-simulator architecture using the HLA paradigm. Technical Report NLR-TP-2002-470, NLR - Nationaal Lucht en Ruimtevaartlaboratorium, September 2002.

[LL73] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environnment. *Journal of the Association for Computing Machinery*, 20(1), 1973.

[LM80] J. Y. T. Leung and J. Merrill. A Note on Preemptive Scheduling of Periodic, Real-Time Tasks. *Information Processing Letters*, 11 :115–118, 1980.

[LPZ+06] Ning Li, Xiao-Yuan Peng, Min-Hu Zhang, Mei Wang, and Guang-Hong Gong. Multimedia communication over HLA/RTI. *Simulation Modelling Practice and Theory*, 14(2) :161–176, 2006.

[LR11] Jose M. Lopez-Rodriguez. How to Develop true distributed real-time simulation ? Mixing IEEE HLA and OMG DDS standards. In *Proceedings of the Spring Simulation Interoperability Workshop*, April 2011. 11S-SIW-072.

[LRCJ04] Peng Li, Binoy Ravindran, Hyeonjoong Cho, and E. Douglas Jensen. Scheduling Distributable Real-Time Threads in Tempus Middleware. In *Proceedings of the Parallel and Distributed Systems, Tenth International Conference*, ICPADS '04, Newport Beach, CA, USA, July 2004.

[LRD+93] Jane W.S. Liu, Juan Redondo, Zhong Deng, Too Tia, Riccardo Bettati, Ami Silberman, Matthew Storch, Rhan Ha, and Wei Shih. PERTS : A prototyping environment for Real-Time Systems. Technical report, University of Illinois, Champaign, IL, USA, May 1993.

[LRT92] J.P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS '92, pages 110 –123, Phoenix, Arizona, USA, December 1992.

[LSD89] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm : exact characterization and average case behavior. In *Proceedings of 10th Real-Time Systems Symposium*, RTSS '89, pages 166–171, Santa Monica, CA, USA, December 1989.

[LSS87] John P. Lehoczky, Lui Sha, and Jay K. Strosnider. Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS '87, pages 261–270, San Jose, CA, USA, December 1987.

[LSST91] J.P. Lehoczky, L. Sha, J.K. Strosnider, and H. Tokuda. Fixed priority scheduling theory for hard real-time systems. In *Foundations of Real-Time Computing : Scheduling and resource management*, pages 1–30, 1991.

[LT88] K. B. Lakshmanan and K. Thulasiraman. On the Use of Synchronizers for Asynchronous Communication Networks. In *Proceedings of the 2nd International Workshop on Distributed Algorithms*, pages 257–277. Springer-Verlag, 1988.

[LT95] John P. Lehoczky and Sandra R. Thuel. Scheduling periodic and aperiodic tasks using the slack stealing algorithm. In Sang H. Son, editor, *Advances in Real-Time Systems*, chapter 8, pages 175–198. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[Lut97] Robert Lutz. A Comparison Of HLA Object Modeling Principles With Traditional Object-Oriented Modeling Concepts. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 1997. 97F-SIW-025.

[LW82] J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation Journal*, 2(4) :237–250, 1982.

[LY87] Ten H. Lai and Tao H. Yang. On distributed snapshots. *Information Processing Letters*, 25(3) :153–158, May 1987.

[LYTW06] Buquan Liu, Yiping Yao, Jing Tao, and Huaimin Wang. Development of a runtime infrastructure for large-scale distributed simulations. In *Proceedings of the 38t Winter Simulation Conference*, WSC '06, pages 1036–1043, Monterey, California, 2006.

[Mai96] Luc Maillet. *Spécification et Validation d'une architecture de système distribué pour le contrôle d'exécution d'applications temps réel complexes.* PhD thesis, École Nationale Supérieur de l'Aéronautique et de l'Espace, Toulouse, France, Juillet 1996.

[MAJ+98] D. Miller, S. Adelson, A. Janett, J. Eskew, S. Haes, B. Gibson, J. Huntley, C. Rayment, B. Stroud, T. Usher, and S. Kasputis. Dynamic Terrain and Objects in the STOW 97 Advanced Concept Technology Demonstration Using the High Level Architecture. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 1998.

[Mat89] Friedemann Mattern. Virtual Time and Global States of Distributed Systems. In *Parallel and Distributed Algorithms*, pages 215–226. North-Holland, 1989.

[Mat93] Friedemann Mattern. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel and Distributed Computing - Special issue on parallel and discrete event simulation*, 18(4) :423–434, August 1993.

[MB98] S. Moore and J. Bell. A Simulation Execution Environment for STOW 97. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 1998.

[MBB00] Y. Meylan, A. Bajpai, and R. Bettati. ProtEx : a toolkit for the analysis of distributed real-time systems. In *Proceedings of the Seventh International Conference on Real-Time Systems and Applications*, RTCSA '00, page 183, Cheju Island, South Korea, December 2000.

[MBR+07] Audrey Marchand, Patricia Balbastre, Ismael Ripoll, Miguel Masmano, and Alfons Crespo. Memory Resource Management for Real-Time Systems. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, ECRTS '07, pages 201–210, Pisa, Italy, July 2007.

[MBRC08] A. Marchand, P. Balbastre, I. Ripoll, and A. Crespo. Providing Memory QoS Guarantees for Real-Time Applications. In *Proceedings of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '08, pages 312–317, Kaohisung, Taiwan, August 2008.

[MC82] Jayadev Misra and K. M. Chandy. Termination Detection of Diffusing Computations in Communicating Sequential Processes. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(1) :37–43, January 1982.

[McF76] J. H. McFadyen. Systems Network Architecture : An overview. *IBM Systems Journal*, 15(1) :4–23, March 1976.

[McL01] Thom McLean. Hard Real-Time Simulations using HLA. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 2001.

[Mcl02] Thom Mclean. *Real-Time Distributed Simulation Analysis : An application of Temporal Database and Simulation Systems Research.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2002. Director-Mark, Leo and Director-Fujimoto, Richard.

[MCS91] John M. Mellor-Crummey and Michael L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 9(1) :21–65, February 1991.

[Meu09] Meumeu Yomsi, Patrick. *La prise en compte du coût exact de la préemption dans l'ordonnacement temprs réel monoprocesseur avec des contraintes multiples.* PhD thesis, Faculté des sciences d'Orsay et Université Paris-Sud 11, Paris, France, Avril 2009.

[MF00] Thom McLean and Richard Fujimoto. Repeatability in Real-Time Distributed Simulation Executions. In *Proceedings of the fourteenth workshop on Parallel and distributed simulation*, PADS '00, pages 23–32, Bologna, Italy, May 2000.

[MF01] Thom McLean and Richard Fujimoto. The Federated-Simulation Development Kit (FDK) : A "Source-Available" HLA RTI. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 2001. 01S-SIW-095.

[MF03] Thom McLean and Richard Fujimoto. Predictable Time Management for Real-Time Distributed Simulation. In *Proceedings of the 17th Workshop on PArallel and Distributed Simulation*, PADS '03, page 89, San Diego, CA, USA, June 2003.

[MFF04] Thom McLean, Richard Fujimoto, and Brad Fitzgibbons. Middleware for real-time distributed simulations. *Concurrency and Computation Journal : Practice and Experience*, 16(15) :1483 –1501, 2004.

[MH07] Mohamed.A. Mastouri and Salem Hasnaoui. Performance of a Publish/Subscribe Middleware for the Real- Time Distributed Control systems. *International Journal on Computer Science and Network Security*, 7(1), January 2007.

[Mil85] Mills, D.L. *Network Time Protocol (NTP).* Internet Engineering Task Force (RFC-958), September 1985.

[Mil90] David L. Mills. On the Accuracy and Stability of Clocks Synchronized by the Network Time Protocol in the Internet System. *ACM Computer Communication Review*, 20 :65–75, 1990.

[Mil92] Mills, D.L. *Network Time Protocol Specification, Implementation and Analysis - Version 3 (RFC-5905).* Internet Engineering Task Force, March 1992.

[Mil94] David L. Mills. Precision synchronization of computer network clocks. *ACM SIGCOMM Computer Communication Review Newsletter*, 24(2) :28–43, April 1994.

[Mil10] Mills, D. and Delaware, U. and Martin, J. and Burbank, J. and Kasch, W. *Network Time Protocol Version 4 : Protocol and Algorithms Specification (RFC-5905).* Internet Engineering Task Force, June 2010.

[Mis86] Jayadev Misra. Distributed Discrete-Event Simulation. *ACM Computing Surveys (CSUR) Journal*, 18 :39–65, 1986.

[MlR09] L. Malinga and H. le Roux. HLA RTI Performance Evaluation. In *Proceedings of the European Simulation Interoperability Workshop*, Istanbul, Turkey, June 2009. 09E-SIW-005.

[MMMM01] G. Manimaran, Shashidhar Merugu, Anand Manikutty, and C. Siva Ram Murthy. Integrated scheduling of tasks and messages in distributed real-time systems. In *Engineering of distributed control systems*, pages 99–112. Nova Science Publishers, Inc., Commack, NY, USA, 2001.

[MMTS96] H. Mori, Y. Mano, H. Takada, and K. Sakamura. μITRON bus : a real-time control LAN for open network environment. In *Proceedings of the Third International Workshop on Real-Time Computing Systems Application*, RTCSA '96, pages 227 –234, Seoul, Korea, October 1996.

[Mok83] Aloysius Ka-Lau Mok. *Fundamental design problems for the hard real-time environments.* PhD thesis, MIT - Massachusetts Institute of Technology, Boston, MA, USA, 1983.

[MRR+08] M. Masmano, I. Ripoll, J. Real, A. Crespo, and A. J. Wellings. Implementation of a constant-time dynamic storage allocator. *Software - Practice & Experience*, 38(10) :995–1026, August 2008.

[MS97] Katherine L. Morse and Jeffrey S. Steinman. Data Distribution Management in the HLA - Multidimensional Regions and Physically Correct Filtering. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 1997.

[MS98] Michael D. Myjak and Sean.T Sharp. Java Real-Time RTI. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, March 1998. 98F-SIW-244.

[MT07] Miguel Angel Masmano Tello. *Gestion de la Memoria Dinamica en Sistemas de Tiempo Real.* PhD thesis, Universad Politecnica de Valencia, Valencia, Spain, Mayo 2007.

[MWM88] Vijay Madisetti, Jean Walrand, and David Messerschmitt. Wolf : a rollback algorithm for optimistic distributed simulation systems. In *Proceedings of the Winter Simulation Conference*, pages 296 –305, December 1988.

[MZ95] Nicholas Malcolm and Wei Zhao. Hard real-time communication in multiple-access networks. *The Journal of Real-Time Systems*, 8(1) :35–77, 1995.

[MZ96] Gordon Miller and A. A. Zabek. The Joint Training Confederation and the Aggregate Level Simulation Protocol. *Phalanx*, pages 24–27, June 1996.

[Mö05] Möller, Born and Karlsson, Mikael. Developing Well-Balanced Federations using the HLA Evolved Smart Update Rate Reduction. In *Proceddings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 2005. 05F-SIW-087.

[Mö08] Möller,B. and Morse, KL and Lightner, M. and Little, R. and Lutz, R. . HLA Evolved – A Summary of Major Technical Improvements. In *Proceddings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 2008. 08F-SIW-064.

[Mö09] Möller,B. and Morse, KL and Lightner, M. and Little, R. and Lutz, R. . Early Experiences from Migrating to the HLA Evolved C++ and Java APIs. In *Proceddings of the Spring Simulation Interoperability Workshop*, San Diego, CA, USA, April 2009. 09S-SIW-083.

[Nat76] National Oceanic and Atmospheric Administration, National Aeronautics and Space Administration, US Air Force. *U.S. Standard Atmosphere*, 1976.

[Nav99] Nicolas Navet. *Évaluations de performances temporelles et optimisation de l'ordonnancement de tâches et messages.* PhD thesis, Institut National Polytechnique de Lorraine, Nancy, France, Novembre 1999.

[NdS08] Eric Noulard, Bruno d'Ausbourg, and Pierre Siron. Running Real Time Distributed Simulations under Linux and CERTI. In *Proceedings of European Simulation Interoperability Workshop*, June 2008. 08E-SIW-061.

[Nel98] R.C. Nelson. *Flight Stability and Automatic Control.* MA : WCB/Mc GrawHill, Boston, MA, USA, second edition, 1998.

[Nil04] Anders Nilsson. Compiling java for real-time systems. Licentiate thesis, Department of Computer Science, Lund Institute of Technology, Sweden, 2004.

[NRS09] Eric Noulard, Jean-Yves Rousselot, and Pierre Siron. CERTI : An open Source RTI, Why and How. In *Proceedings of the Spring Simulation Interoperability Workshop*, March 2009. 09S-SIW-015.

[Obe82] Ron Obermarck. Distributed deadlock detection algorithm. *ACM Transactions on Database Systems (TODS)*, 7(2) :187–208, June 1982.

[Oga94] K. Ogata. *Discrete-Time Control Systems (2nd Edition).* Hoboken, NJ : Wiley. Prentice Hall,, 1994.

[Oga95] T. Ogasawara. An algorithm with constant execution time for dynamic storage allocation. In *Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications*, RTCSA '95, pages 21–25, Tokyo, Japan, October 1995.

[OKS+00] Carlos O'Ryan, Fred Kuhns, Douglas C. Schmidt, Ossama Othman, and Jeff Parsons. The design and performance of a pluggable protocols framework for real-time distributed object computing middleware. In *IFIP/ACM International Conference on Distributed systems platforms*, Middleware '00, pages 372–395, Secaucus, NJ, USA, 2000. Springer-Verlag New York, Inc.

[OMG98a] OMG - Object Management Group. *Corba Messaging Specifications*, OMG Document orbos/98-05-05 edition, May 1998.

[OMG98b] OMG - Object Management Group. *Minimum CORBA - Joint Revised*, OMG Document orbos/98-08-04 edition, August 1998.

[OMG99a] OMG - Object Management Group. *Real-Time CORBA - Joint Revised Submission*, orbos/99-02-12 edition, Febuary 1999.

[OMG99b] OMG - Object Management Group. *Real-Time CORBA Spécifications 1.0*, orbos/99-06-02 edition, June 1999.

[OMG02] OMG - Object Management Group. *Distributed Simulation Systems Specification - Version 2.0*, formal/02-11-11 edition, September 2002.

[OMG04] OMG - Object Management Group. *Data Distribution Service for Real-time Systems Specification version 1.0*, OMG Document formal/04-12-02 edition, December December, 2004.

[OMG05] OMG - Object Management Group. *Real-Time CORBA Spécifications 1.2*, orbos/05-01-04 edition, January 2005.

[OMG07] OMG - Object Management Group. *Data Distribution Service for Real-time Systems Specification version 1.2*, OMG Document formal/07-01-01 edition, January 2007.

[OMG08] OMG - Object Management Group. *Enhanced View of Time Service Specification - Version 2.0*, formal/2008-08-01 edition, August 2008.

[OMG09] OMG - Object Management Group. *The Real-time Publish-Subscribe Wire Protocol - DDS Interoperability Wire Protocol Specification - Version 2.1*, formal/2009-01-05 edition, January 2009.

[Ore87] Ti Oren. *Taxonomy of Simulation Model Processing.* Pergamon Press, 1987.

[OS93] Yingfeng Oh and Sang H. Son. Tight Performance Bounds of Heuristics for a Real-Time Scheduling Problem. Technical Report CS-93-24, University of Virginia, Charlottesville, VA, USA, May 1993.

[OS95] Yingfeng Oh and Sang H. Son. Fixed-Priority Scheduling of Periodic Tasks on Multiprocessor Systems. Technical Report CS-95-16, University of Virginia, Charlottesville, VA, USA, March 1995.

[OSE04] OSEK Group. *OSEK/VDX Operating Binding Specification 1.4.2*, July 2004.

[OSE05] OSEK Group. *OSEK/VDX Operating System Specification 2.2.3*, Febuary 2005.

[OY98] S.-H. Oh and S.-M. Yang. A Modified Least-Laxity-First Scheduling Algorithm for Real-Time Tasks. In *Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications*, RTCSA '98, Hiroshima, Japan, October 1998.

[Pal07] Palma, N. and Krakowiak, S. and Mossière, J. *Intergiciel et Construction d'Applications Réparties.* Creative Commons Licence, Janvier 2007.

[Pau02] Laurent Pautet. Intergiciels schizophrènes : une solution fortement interopérable fondée sur les composants adaptables. In *Journées Systèmes à composants adaptables et extensibles*, 2002.

[Pay07] Hannes E. Payer. A compacting real-time memory management system. Master's thesis, University of Salzburg, Salzburg, Austria, 2007.

[PB97] Ernest H. Page and William E. Babineau. The ALSP Joint Training Confederation : A Case Study of Federation Testing. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 1997. 97F-SIW-061.

[PC03] Gerardo Pardo-Castellote. OMG Data-Distribution Service : Architectural Overview. In *Proceedings of the 2003 IEEE conference on Military communications - Volume I*, MILCOM'03, pages 242–247, 2003.

[Pea03] Trevor W. Pearce. Simulation-driven architecture in the engineering of real-time embedded systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS '03, Cancun, Mexico, December 2003.

[PGGH97] J.C. Palencia, J.J. Garcia, and M. Gonzalez Harbour. On the schedulability analysis for distributed hard real-time systems. In *Proceedings of the IEEE Euromicro Conference on Real-Time Systems*, Toledo, Spain, June 1997.

[Phi05] Phillips, Robert G. and Crues, Edwin Z. Time Management Issues and Approaches for Real-Time HLA Based Simulation. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 2005. 05F-SIW-058.

[PK89] P. Puschner and Ch. Koza. Calculating the maximum, execution time of real-time programs. *The Journal of Real-Time Systems*, 1(2) :159–176, September 1989.

[Pos80a] Postel, Jon. *Transmission Control Protocol (RFC-761)*. Internet Engineering Task Force, August 1980.

[Pos80b] Postel, Jon. *User Datagram Protocol (RFC-769)*. Internet Engineering Task Force, August 1980.

[Pos81] Postel, Jon. *Service Mappings (RFC-795)*. Network Working Group, September 1981.

[Pre90] Preiss, Bruno R. and Loucks, Wayne M. The Impact of Lookahead on the Performance of Conservative Distributed Simulation. In *1990 European Multiconference-Simulation Methodologies, Languages and Architectures*, pages 204–209, Nuremberg, FRG, June 1990. SCS.

[PS89] D.T. Peng and K.G. Shin. Static Allocation of Periodic Tasks with Precedence Constraints in Distributed Real-Time Systems. In *Proceedings of the 10th International Conference on Distributed Computing*, Newport Beach, CA, USA, June 1989.

[PS97] Peter Puschner and Anton Schedl. Computing Maximum Task Execution Times - A Graph-Based Approach. *The Journal of Real-Time Systems*, 13 :67–91, 1997.

[PSCS00] Irfan Pyarali, Marina Spivak, Ron Cytron, and Douglas C. Schmidt. Evaluating and Optimizing Thread Pool Strategies for Real-Time CORBA. In *Proceedings of the ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Embedded Systems*, pages 214–222, 2000.

[Pua02] Isabelle Puaut. Real-time performance of dynamic memory allocation algorithms. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, ECRTS '02, pages 41–51, Vienna, Austria, June 2002.

[Pul99] Pullen, M. and Myjak, M. and Bouwens, C. *Limitations of Internet Protocol Suite for Distributed Simulation in the Large Multicast Environment (RFC-2502)*. Internet Engineering Task Force - Large-Scale Multicast Applications (LSMA) working group , Febuary 1999.

[Qui03] Thomas Quinot. *Conception et réalisation d'un intergiciel schizofrène pour la mise en œuvre de systèmes répartis interopérables.* PhD thesis, Université Paris VI, Pierre-et-Marie Curie, Paris, France, Mars 2003.

[Ram90] K. Ramamritham. Allocation and Scheduling of Complex Periodic Tasks. In *Proceedings of the International Conference on Distributed Comping Systems*, Austin, Texas, USA, May 1990.

[Ray92] Michel Raynal. *Synchronisation et état global dans les systèmes répartis (Tome 2 d'une introduction aux principes des systèmes répartis).* Editions Eyrolles, 1992.

[RCK01] Pascal Richard, Francis Cottet, and Claude Kaiser. Précédences généralisées et ordonnançabilité des tâches de suivi temps réel d'un laminoir. *Journal Européen des Systèmes Automatisés*, 35(9) :1055–1071, 2001.

[Rey92] Franklin Reynolds. An Architectural Overview of Alpha : A Real-Time Distributed Kernel. In *Proceedings of the Workshop on Micro-kernels and Other Kernel Architectures*, pages 127–146, 1992.

[RH02] Mario Aldea Rivas and Michael González Harbour. POSIX-Compatible Application-Defined Scheduling in MaRTE OS. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, Washington, DC, USA, 2002. IEEE Computer Society.

[Riv98] Nicolas Rivierre. *Ordonnancement temps réel centralisé, les cas préemptifs et non préemptifs.* PhD thesis, Université de Versailles - Saint Quentin en Yvelines, Versailles, France, Février 1998.

[RML03] J. Rodriguez, Z. Mammeri, and P. Lorenz. Using CORBA notification service and RSVP to provide end-to-end QoS guarantees. In *Proceedings of the 10th International Conference on Telecommunications*, volume 2 of *ICT'03*, pages 1243–1250, march 2003.

[Rob77] J.M Robson. Worst Case fragmentation of First-fit and Best-fit storage allocation strategies. *The Computer Journal*, 20(3) :242–244, January 1977.

[RPA02] I. Ripoll, P. Pisa, and L. Abeni. Rtos state of the art analysis : Deliverable d1.1 – rtos analysis. Technical report, WP1 OCERA Project, 2002.

[RRC01] M. Richard, P. Richard, and F. Cottet. Méthode d'affectation optimale des priorités des tâches et des messages dans les systèmes distribués temps-réel basée sur l'analyse holistique. Technical report, Laboratoire d'Informatique Scientifique et Industrielle - École Nationale Supérieure de Mécanique et d'Aéro-technique, April 2001.

[RRC03a] Michael Richard, Pascal Richard, and Francis Cottet. Placement et Validation dans les Systèmes Temps Réel Distribués. In Teknea, editor, *RTS'03*, pages 387–410, 2003.

[RRC03b] Pascal Richard, Michaël Richard, and Francis Cottet. *Analyse holistique des systèmes temps réel distribués : principes et algorithmes*, chapter 7. Hermès, 2003.

[RSL88] R. Rajkumar, L. Sha, and J.P. Lehoczky. Real-time Synchronization Protocols for Multiprocessors. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, RTSS '88, pages 259 –269, December 1988.

[RSM97] Steven J. Rak, Marnie Salisbury, and Robert S. Macdonald. HLA/RTI data distribution management in the synthetic theater of war. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 1997. 97F-SIW-119.

[RSP07] A.S. Rhoades, G. Schrader, and P. Poulin. Benchmarking Publish/Subscribe Middleware for Radar Applications. In *Proceedings of the High Performance and Embedded Computing*, September 2007.

[RW89] R. Righter and J.C. Walrand. Distributed simulation of discrete event systems. *Proceedings of the IEEE*, 77(1) :99–113, jan 1989.

[SADLCJ01] Witawas Srisa-An, Chia-Tien Dan Lo, and Morris Chang J. A Performance Analysis of the Active Memory System. In *Proceedings of the International Conference on Computer Design : VLSI in Computers & Processors*, ICCD '01, pages 493–496, Austin, TX, USA, September 2001.

[Sai91] Kazunori Saitoh. ITRON standards. In *Proceedings of the TRON Project Symposium*, pages 16–24, November 1991.

[Sak99] Ken Sakamura. *μITRON 4.0 Specification*. ITRON Committee, TRON association, June 1999.

[Sam85] Behrokh Samadi. *Distributed Simulation, Algorithms and Performance Analysis*. PhD thesis, University of California, Los Angeles, CA, USA, 1985.

[Sau10] David Alexandre Saussie. *Contrôle du vol longitudinal d'un avion civil avec satisfaction de qualités de manoeuvrabilité*. PhD thesis, Thèse de Doctorat, École Polytechnique de Montréal, Département de génie électrique, Montréal, Canada, 2010.

[SB94] M. Spuri and G.C. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings of the 15th IEEE Real-Time Systems Symposium*, RTSS '94, pages 2 –11, San Juan, Puerto Rico, December 1994.

[SB98] Samia Saad-Bouzefrane. *Étude temporelle des Applications Temps Réel Distribuées à Contraintes Strictes basée sur une Analyse d'Ordonnançabilité*. PhD thesis, Université de Poitiers, Poitiers, France, Janvier 1998.

[Sch98] Douglas C. Schmidt. Evaluating architectures for multithreaded object request brokers. *Communincation ACM*, 41(10) :54–60, October 1998.

[Sch03] Schulzrinne, H. and Casner, S. and Frederick, R. and Jacobson, V. *RTP : A Transport Protocol for Real-Time Applications (RFC-3550).* Network Working Group, July 2003.

[SCS11] David Saussié, J.B Chaudron, and Pierre Siron. Simulation Distribuée du Vol d'un Avion selon le Standard HLA. Technical report, ISAE - Institut Supérieur de l'Aéronautique et de l'Espace, 2011.

[SD01] Victor.J Skowronski and Matthew Dorsh. Modifying the RTI for Active Networks. In *Proceedings of Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 2001. 01S-SIW-007.

[Ser71] Omri Serlin. Scheduling of time critical processes. In *Proceedings of the Fall Joint Computer Conference*, AFIPS '71, pages 925–932, Las Vegas, Nevada, November 1971.

[SG83] James E. Smith and James R. Goodman. A study of instruction cache organizations and replacement policies. *ACM SIGARCH Computer Architecture News*, 11(3) :132–137, June 1983.

[She97] Shenker, S. and Partridge, C. and Guerin, R. *Specification of Guaranteed Quality of Service (RFC-2212).* Network Working Group, September 1997.

[Sho75] John E. Shore. On the external storage fragmentation produced by First-fit and Best-fit allocation strategies. *Communications of the ACM*, 18(8) :433–440, August 1975.

[Sin07] Singhoff, F. . Cheddar Release 2.0 user's guide. Technical Report singhoff-01-07, Laboratoire LISYC, Université de Brest, February 2007.

[Sir98] Pierre Siron. Design and Implementation of a HLA RTI Prototype at ONERA. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, 1998. 98F-SIW-036.

[SJ87] K. C. Sevcik and M. J. Johnson. Cycle Time Properties Of The FDDI Token Ring Protocol. *IEEE Transactions on Software Engineering*, 13(3), March 1987.

[SK86] Madalene Spezialetti and Phil Kearns. Efficient distributed snapshots. In *Proceedings of the 6th International Conference on Distributed Computing Systems*, ICDCS'86, pages 382–388, Cambridge, Massachusetts, USA, May 1986.

[SK00] Douglas C. Schmidt and Fred Kuhns. An Overview of Real-time CORBA Specification. *Computer - Special issue on Object-Orented Real-time Distributed Computing*, 33(6) :56 –63, June 2000.

[SL96] J. Sun and J. Liu. Bounding the end-to-end response times of tasks in a distributed real-time systems using direct synchronisation protocol. Technical Report R-96-1949, Deptartment of Computer Science, University Urbana-Champaign, Urbana, IL, USA, 1996.

[SL03] B.L. Stevens and F.L. Lewis. *Aircraft Control and Simulation* . Hoboken, NJ : Wiley, second edition, 2003.

[SLNM04] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar : a flexible real time scheduling framework. In *Proceedings of the 2004 annual ACM SIGAda international conference on Ada : The engineering of correct and reliable software for real-time & distributed systems using Ada and related technologies*, SIGAda '04, pages 1–8, Atlanta, Georgia, USA, 2004.

[SLS88] B. Sprunt, J. Lehoczky, and L. Sha. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS '88, pages 251 –258, Huntsville, Alabama, USA, December 1988.

[SLS95] Jay K. Strosnider, John P. Lehoczky, and Lui Sha. The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. *IEEE Transactions on Computers*, 44(1) :73–91, January 1995.

[SMFGG98] Douglas C. Schmidt, Sumedh Mungee, Sergio Flores-Gaitan, and Aniruddha Gokhale. Alleviating Priority Inversion and Non-determinism in Real-time CORBA ORB Core Architectures. In *Proceedings of the IEEE Real-Time Technology And Applications Symposium*, RTAS '98, Denver, CL, USA, June 1998.

[Smi97] Steven W. Smith. *The scientist and engineer's guide to digital signal processing.* California Technical Publishing, San Diego, CA, USA, 1997.

[SNT+06] Vahid Salmani, Mahmoud Naghibzadeh, Amirhossein Taherinia, Malihe Bahekmat, and Sedigheh Khajouie Nejad. Performance evaluation of deadline-based and laxity-based scheduling algorithms in real-time multiprocessor environments. In *Proceedings of the 6th WSEAS International Conference on Systems Theory & Scientific Computation*, pages 121–126, Elounda, Crete Island, Greece, 2006.

[SO03] Douglas C. Schmidt and Carlos O'Ryan. Patterns and performance of distributed real-time and embedded publisher/subscriber architectures. *Jounal of System Softwares*, 66 :213–223, June 2003.

[Sor74] Paul Gordon Sorenson. *A methodology for real-time system development.* PhD thesis, University of Toronto, Toronto, Canada, 1974.

[SPB02] Etienne Schneider, Florentin Picioroaga, and A. Béchina. Distributed Real-Time Computing for Microcontrollers - The OSA+ Approach. In *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, ISORC '02, Washington, DC, USA, May 2002.

[Spr90] Brinkley Sprunt. *Aperiodic task scheduling for real-time systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, August 1990.

[Spu96] Marco Spuri. Analysis of deadline scheduled real-time systems. Research Report RR-2772, INRIA - Institut National de Recherche en Informatique et Automatique, January 1996.

[Spu00] Charles E. Spurgeon. *Ethernet : The Definitive Guide.* O'REILLY, 2000.

[SRL90] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols : An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9) :1175–1185, September 1990.

[SS88] Wen-King Su and Charles Seitz. Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm. Technical Report CS-TR-88-22, Departement of Computer Science, California Institute of Technology, 1988.

[SSL89] Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for Hard-Real-Time systems. *The Journal of Real-Time Systems*, 1(1) :27–60, 1989.

[Sta88] John A. Stankovic. Misconceptions About Real-Time Computing : A Serious Problem for Next-Generation Systems. *Computer Journal*, 21(10) :10–19, October 1988.

[Sta92] John A. Stankovic. Distributed real-time computing : the Next Generation. Technical report, University of Massachussets Amherst, 1992.

[Str97] Bjarne Stroustrup. *The C++ Programming Language, Third Edition.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1997.

[Sun97] Jun Sun. *Fixed-Priority End-To-End Scheduling in Distributed Real-Time Systems.* PhD thesis, University of Illinois, Champaign, IL, USA, 1997.

[SUN00] SUN Microsystems, Inc. *Scalable Real-Time Computing in the Solaris Operating Environment*, technical white paper edition, 2000.

[Sym99] Symington, S. and Wood, D. and Pullen, M. *Modeling and Simulation Requirements for IPng (RFC-1667).* Internet Engineering Task Force - Large-Scale Multicast Applications (LSMA) working group , Febuary 1999.

[Tan06] Tan, Kun and Song, Jingmin and Zhang, Qian and Sridharan, Murari. Compound TCP : A Scalable and TCP-Friendly Congestion Control for High-speed Networks. In *Proceedings of the 4th International Workshop on Protocols for Fast Long-Distance Networks*, PFLDN'06, Nara, Japan, May 2006.

[TBW92] K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks : an NP-hard problem made easy. *The Journal of Real-Time Systems*, 4(2) :145–165, May 1992.

[TBW95] K. Tindell, A. Burns, and A. J. Wellings. Analysis of hard real-time communications. *The Journal of Real-Time Systems*, 9(2) :147–171, September 1995.

[TC94] Ken Tindell and John Clark. Holistic Schedulability Analysis for Distributed Hard real-time systems. *Microprocessing and Microprogramming - Parallel Orocessing in Embedded Real-Time Systems*, 40(2-3) :117–134, April 1994.

[Ten02] Ten Gigabit Ethernet Alliance. *Ethernet – The Next Generation WAN Transport Technology*, white paper edition, May 2002.

[TF98] Ivan Tacic and R. M. Fujimoto. Synchronized Data Distribution Management in Distributed Simulations. *ACM SIGSIM Simulation Digest Newsletter*, 28(1) :108–115, 1998.

[THW94] K.W. Tindell, H. Hansson, and A.J. Wellings. Analysing real-time communications : Controller Area Network (CAN). In *Proceedings of Real-Time Systems Symposium*, pages 259 –263, San Juan, Puerto Rico, December 1994.

[Tie96] T.R. Tiernan. Synthetic Theater of War (STOW) Demonstration-1 (ED-1) Final Report. Technical Report 1721, Defence Advanced Research Project Agency, June 1996.

[Tie03] Tietje, Herbert. Benchmarking of RTIs for Real-Time Applications. In *Proceedings of the European Simulation Interoperability Workshop*, Stockholm, Sweden, June 2003. 03E-SIW-026.

[Tin94] Ken Tindell. *Fixed priority scheduling of hard real time systems.* PhD thesis, University of York, York, England, 1994.

[TLS96] Too-Seng Tia, Jane W.-S. Liu, and Mallikarjun Shankar. Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems. *The Journal of Real-Time Systems*, 10(1) :23–43, January 1996.

[TLSH94] Too-Seng. Tia, Jane W.-S. Liu, Jun. Sun, and Rhan Ha. *A Linear-Time Optimal Acceptance Test for Scheduling of Hard Real-Time Tasks.* Urbana, IL, USA, November 1994. (Submitted to IEEE Transactions on Software Engineering).

[TPF+05] Yarong Tang, Kalyan S. Perumalla, Richard M. Fujimoto, Homa Karimabadi, Jonathan Driscoll, and Yuri Omelchenko. Optimistic Parallel Discrete Event Simulations

of Physical Systems Using Reverse Computation. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, PADS '05, pages 26–35, Monterey, CA USA, 2005.

[VdC96] Francisco Vasques de Carvalho. *Sur l'intégration de mécanismes d'ordonnancement et de communication dans la sous-couche MAC de réseaux locaux temps réel.* PhD thesis, Université Paul Sabatier de Toulouse, Toulouse, France, 1996.

[VK48] Theodore Von Kármán. Progress in the Statistical Theory of Turbulence. *National Academy of Science Journal*, 34 :530–539, november 1948.

[VM00] Fernando Vardânega and Carlos Maziero. A Generic Rollback Manager for Optimistic HLA Simulations. In *Proceedings of the Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, DS-RT '00, San Francisco, CA, USA, August 2000.

[WBTK95] Victor Fay Wolfe, John K Black, Bhavani Thuraisingham, and Peter Krupp. Real-time Method Invocations in Distributed Environments. In *Proceedings of the International Conference on High Performance Computing*, HiPC'95, New Delhi, India, December 1995.

[WDG+99] Victor Fay Wolfe, Lisa Cingiser Dipippo, Roman Ginis, Michael Squadrito, Steven Wohlever, Igor Zykh, and Russell Johnston. Expressing and Enforcing Timing Constraints in a Dynamic Real-Time CORBA System. *The Journal of Real-Time Systems*, 16(2/3) :253–280, May 1999.

[WEE+08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3) :36 :1–36 :53, May 2008.

[Wei09] Thomas Weise. *Global Optimization Algorithms - Theory and Application - (2nd Edition).* GNU Free Documentation License, 2009.

[WG01] Douglas.D. Wood and Len Granowetter. Rationale and Design of the MÄK Real-Time RTI. In *Proceedings of the Spring Simulation Interoperability Workshop*, Orlando, FL, USA, March 2001. 01S-SIW-104.

[Wil97] P.A. Wilsey. Feedback Control in Time Warp Synchronized Parallel Simulators. In *Pronceedings of the First International Workshop on Distributed Interactive Simulation and Real Time Applications*, DS-RT '97, pages 31 –38, Eilat, Israel, January 1997.

[Win08] Wind River Systems, Inc. *VxWorks Application Programmer's Guide (version 6.7)*, November 2008.

[Wir71] Wirth, Niklaus . The Programming Language Pascal. In *Acta Informatica*, volume 1, pages 35–63, 1971.

[WJNB95] Paul R. Wilson, Mark S. Johnstone, Michael Neely, and David Boles. Dynamic Storage Allocation : A Survey and Critical Review. In *Proceedings of the International Workshop on Memory Management*, IWMM '95, pages 1–78, Kinross, UK, September 1995.

[Wlo11] Tomasz Wlostowski. Precise time and frequency transfer in a White Rabbit network. Master's thesis, Warsaw University of Technology, Warszawa (Varsovie), Politechniki (Pologne), 2011.

[WO00] Jiann-Rong Wu and Ming Ouhyoung. On latency compensation and its effects on head-motion trajectories in virtual environments. *The Visual Computer Journal*, 16(2) :79–90, 2000.

[Wro97a] Wroclawski, J. *Specification of the Controlled-Load Network Element Service (RFC-2211)*. Network Working Group, September 1997.

[Wro97b] Wroclawski, J. *The Use of RSVP with IETF Integrated Services (RFC-2210)*. Network Working Group, September 1997.

[WT94] Kenneth R. Wood and Stephen J. Turner. A generalized carrier-null method for conservative parallel simulation. In *Proceedings of the eighth workshop on Parallel and distributed simulation*, PADS '94, pages 50–57, Edinburgh, Scotland, United Kingdom, 1994.

[WTY+04] Xiaoguang Wang, Stephen John Turner, Malcolm Yoke, Hean Low, and Boon Ping Gan. Optimistic synchronization in HLA based distributed simulation. In *Proceedings of the eighteenth workshop on Parallel and distributed simulation*, PADS '04, pages 123–130, Kufstein, Austria, May 2004.

[WW94] A.L. Wilson and R.M. Weatherly. The Aggregate Level Simulation Protocol : An Evolving System. In *Proceedings of the 26th conference on Winter simulation*, WSC '94, pages 781–787, Lake Buena Vista, FL, USA, December 1994.

[Yag02] Karim Yaghmour. ADEOS : Adaptative Domain Environment for Operating Systems. *White Paper*, 2002.

[Yea98] Jessie C. Yeager. Implementation and testing of turbulence models for the F18-HARV simulation. Technical Report NASA/CR-1998-206937, NASA Langley Research Center, Hampton, Virginia, USA, March 1998.

[Zea97] Sherali Zeadally. An evaluation of the real-time performances of SVR 4.0 and SVR 4.2. *ACM SIGOPS Operating Systems Review*, 31(1) :78–87, January 1997.

[ZG01a] Hui Zhao and Nicolas D. Georganas. HLA Real-Time Extension. In *Proceedings of the Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, DS-RT '01, August 2001.

[ZG01b] Hui Zhao and Nicolas.D Georganas. Architecture proposal for Real-Time RTI. In *Proceedings of the Fall Simulation Interoperability Workshop*, Orlando, FL, USA, September 2001.

[Zha01] Hui Zhao. *HLA Streaming and Real-Time Extension*. PhD thesis, School of Information Technology Engineering, University of Ottawa, Ottawa, Canada, 2001.

[Zor93] Benjamin Zorn. The measured cost of conservative garbage collection. *Software Practice & Experiments*, 23(7) :733–756, July 1993.

[ZPGK00] Bernard P. Zeigler, Herber Praehofer, and Tag Gon Kim. *Theory of Modeling and Simulation*. Academic Press, second edition, 2000.

[ZR87] W. Zhao and K. Ramamritham. Virtual Time CSMA Protocols for Hard Real-Time Communication. *IEEE Transactions on Software Engineering*, 13(8) :938–952, 1987.

[ZS94] Qin Zheng and K.G. Shin. On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks. *IEEE Transactions on Communications*, 42(234) :1096–1105, March 1994.

[ZS95] Qin Zheng and Kang G. Shin. Synchronous Bandwidth Allocation in FDDI Networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(12) :1332–1338, December 1995.

Netographie

[1] ACE :
. http://www.cs.wustl.edu/~schmidt/ACE.html.

[2] ADA HOME :
. http://www.adahome.com/.

[3] ADA Inc. :
. http://www.adaic.org/.

[4] ADEOS :
. http://home.gna.org/adeos/.

[5] ARTISST :
. http://www.irisa.fr/aces/software/artisst.

[6] Bibliographie ordonnacement (D.Decotigny) :
. http://david.decotigny.free.fr/rt/intro-ordo/intro-ordo-index.html.

[7] C++ :
. http://www.cplusplus.com/.

[8] CARDANOM :
. http://cardamom.ow2.org/.

[9] CERTI :
. http://savannah.nongnu.org/projects/certi/.

[10] CHEDDAR :
. http://beru.univ-brest.fr/~singhoff/cheddar/index-fr.html.

[11] Chronos RTI :
. http://www.magnetargames.com/Products/Chronos/.

[12] CONCURRENT :
. http://real-time.ccur.com/.

[13] CORE DX :
. http://www.twinoakscomputing.com/coredx.

[14] DC NET :
. http://www.eecis.udel.edu/~mills/dcnet.html.

[15] DL MALLOC :
. http://g.oswego.edu/dl/html/malloc.html.

[16] DRYDEN WIND TURBULENCE :
. http://www.mathworks.com/help/toolbox/aeroblks/drydenwindturbulencemodelcontinuous.html.

[17] ENEA :
. http://www.enea.com/.

[18] EODiSP HLA :
. http://www.pnp-software.com/eodisp.

[19] ESTEREL Technologies (SCADE) :
. http://www.esterel-technologies.com/products/scade-suite/.

[20] FDK :
. http://www.cc.gatech.edu/computing/pads/fdk/.

[21] FLIGHT GEAR :
. http://www.flightgear.org/.

[22] FLIGHT SIMULATOR :
. http://www.microsoft.com/games/flightsimulatorx.

[23] GCC :
. http://gcc.gnu.org/.

[24] GNAT :
. http://www.gnu.org/software/gnat/gnat.html.

[25] GPL :
. http://www.gnu.org/licenses/quick-guide-gplv3.html.

[26] gRMA :
. http://www.tregar.com/gRMA/.

[27] Hard and Soft Real-Time Systems :
. http://www.real-time.org/real-time/hard-and-soft-real-time.html.

[28] IEEE CSMA/CD :
. http://standards.ieee.org/getieee802/802.3.html.

[29] InterCOM DDS :
. http://www.gallium.com/products/intercom.html.

[30] JacORB :
. http://www.jacorb.org/.

[31] JAVA :
. http://www.java.com/fr/.

[32] JAVAC :
. http://openjdk.java.net/groups/compiler/.

[33] JBED :
. http://www.myriadgroup.com/Device-Manufacturers/Java.aspx.

[34] JBED (Dr Doobs Journal) :
. http://drdobbs.com/java/228801031.

[35] LGPL :
. http://www.gnu.org/copyleft/lesser.html.

[36] LINUX EMBARQUE :
. http://uuu.enseirb.fr/~kadionik/embedded/embeddedlinux.html.

[37] LINUX PREEMPTION :
. http://lwn.net/Articles/146861/.

[38] Linux Works :
. http://www.linuxworks.com.

[39] MARTE :
. http://marte.unican.es/.

[40] MICO :
. http://www.mico.org/.

[41] MilSOFT DDS :
. http://dds.milsoft.com.tr/en/dds-home.php.

[42] MÄK RTI :
. http://www.mak.com/products/rti.php.

[43] NTP :
. http://www.ntp.org/.

[44] Omni ORB :
. http://omniorb.sourceforge.net/.

[45] OPEN SLICE :
. http://www.prismtech.com/opensplice.

[46] OSEK-VDX :
. http://www.osek-vdx.org/.

[47] Pitch RTI :
. http://www.pitch.se/products/prti.

[48] PoRTIco :
. http://porticoproject.org/index.php?title=Main_Page.

[49] POSIX 1.2008 :
. http://pubs.opengroup.org/onlinepubs/9699919799/.

[50] PRELUDE COMPILER :
. http://www.lifl.fr/~forget/prelude.html.

[51] PROFIBUS :
. http://www.profibus.com/.

[52] PT MALLOC :
. http://www.malloc.de/en/.

[53] PTPd :
. http://ptpd.sourceforge.net/.

[54] QNX Software :
. http://www.qnx.com.

[55] QT :
. http://qt.nokia.com/products/.

[56] Rapid RMA :
. http://www.tripac.com/html/prod-fact-rrm.html.

[57] RED-HAT SWAP SPACE :
. http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Deployment_Guide/ch-swapspace.html.

[58] ROBIN :
. http://fermitools.fnal.gov/abstracts/robin/abstract.html.

[59] RT Linux :
. http://www.rtlinuxfree.com/.

[60] RT LINUX FOUNDATION :
. http://www.realtimelinuxfoundation.org/.

[61] RTI DDS :
. http://www.rti.com/products/dds/index.html.

[62] RTI DDS :
. http://www.opendds.org/.

[63] RTI Kit :
. http://www.cc.gatech.edu/computing/pads/tech-highperf-rti.html.

[64] RTI-NG 1.3v6 :
. http://fivedots.coe.psu.ac.th/Software.coe/HLA.

[65] RTSJ :
. http://www.rtsj.org/.

[66] SHARK :
. http://shark.sssup.it/.

[67] SIMULINK :
. http://www.mathworks.fr/products/simulink/index.html.

[68] STL ALLOCATORS :
. http://www.tantalon.com/pete.htm.

[69] SymTA/S :
. http://www.symtavision.com/products.html.

[70] TAO 1 :
. http://www.theaceorb.com.

[71] TAO 2 :
. http://www.cse.wustl.edu/~schmidt/TAO.html.

[72] TCC :
. http://bellard.org/tcc/.

[73] TimeWiz :
. http://www.embeddedtechnology.com/product.mvc/TimeWiz-0001.

[74] TLSF :
. http://rtportal.upv.es/rtmalloc/.

[75] TRON :
. http://tronweb.super-nova.co.jp/trondef.html.

[76] UPPAAL :
. http://www.uppaal.com/.

[77] VIRTUAL AIR :
. http://sourceforge.net/projects/virtualair/.

[78] VisiBroker :
. http://www.borland.com/fr/products/visibroker/index.html.

[79] VON KARMAN WIND TURBULENCE :
. http://www.mathworks.com/help/toolbox/aeroblks/
vonkarmanwindturbulencemodelcontinuous.html.

[80] WHITE RABBIT :
. http://www.ohwr.org/projects/white-rabbit.

[81] WIKI RTOS :
. http://en.wikipedia.org/wiki/List_of_real-time_operating_systems.

[82] Wind River :
. http://www.windriver.com.

[83] WINDOWS PRIORITIES :
. http://msdn.microsoft.com/en-us/library/ms685100.aspx.

[84] WINDOWS VIRTUAL MEMORY MANAGEMENT :
. http://msdn.microsoft.com/en-us/library/ms810627.aspx.

[85] WinNT :
. http://technet.microsoft.com/fr-fr/library/cc750820.aspx.

[86] X-PLANE :
. http://www.flightgear.org/.

[87] XENOMAI :
. https://www.xenomai.org.

[88] yaRTI :
. http://pagesperso-orange.fr/dominique.canazzi/dominique.htm.

Architecture de simulation distribuée temps-réel

Ce travail de thèse s'inscrit dans le projet plus global PRISE (Plate-forme de Recherche pour l'Ingénierie des Systèmes Embarqués) dont l'objectif principal est le développement d'une plate-forme d'exécution pour les logiciels embarqués. De tels logiciels sont dits critiques et ils sont, par conséquent, soumis à des règles de conception spécifiques. Notamment, ces logiciels doivent répondre à des contraintes de temps réel et ainsi garantir des comportements temporels prédictifs afin de toujours donner des résultats justes avec le respect d'échéances temporelles.
L'objectif de cette thèse est d'évaluer l'utilisation des techniques de la simulation distribuée (et particulièrement de la norme HLA) pour répondre aux besoins de simulation hybride et temps réel de la plate-forme. Afin de respecter ces contraintes et garantir la prédictibilité temporelle d'une simulation distribuée, il faut avoir une vision complète de l'ensemble du problème et notamment des différents niveaux d'actions : applicatif, intergiciel, logiciel, matériel et aussi formel pour la validation du comportement temporel.
Cette thèse se base sur la RTI (Run Time Infrastructure, intergiciel HLA) de l'ONERA : le CERTI et propose une démarche méthodologique adaptée à ces différents niveaux d'actions. Des cas d'étude, notamment un simulateur du vol d'un avion, ont été spécifiés, implémentés et expérimentés sur la plate-forme PRISE.

Mots clés : PRISE, simulation distribuée, temps réel, HLA, CERTI.

Real-time distributed simulation architecture

This work takes place in the global project PRISE (Plate-forme de Recherche pour l'Ingénierie des Systèmes Embarqués) in which the focus is to develop an execution platform for embedded software. Embedded software are said criticals and, therefore, are subject to specific design rules. Particularly, these software must meet real time constraints and thus ensure a temporal predictive behaviour in order to always give accurate results with respect to corresponding timing deadlines.
The main objective of this thesis is to study the use of distributed simulation techniques (and specifically the HLA standard) to meet the real-time and hybrid simulation needs of the PRISE platform. To comply with these real-time constraints and ensure the predictability of a distributed simulation, we must have a complete view of the whole problem and in particular the different levels of action: application, middleware, software, hardware and also a formal level for validation of the timing behaviour.
This work is based on the RTI (Run Time Infrastructure, HLA middleware) from ONERA laboratory called : the CERTI and proposes a methodological approach adapted to take into account these different levels of action. Some case studies, including a flight simulator of an aircraft, have been specified, implemented and tested on the PRISE platform.

Keywords: PRISE, distributed simulation, real time, HLA, CERTI.