Titre: Title:	Intégration d'un simulateur de partitionnement spatial et temporel à un flot de conception basé sur les modèles
Auteur: Author:	Julien Savard
Date:	2012
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Savard, J. (2012). Intégration d'un simulateur de partitionnement spatial et temporel à un flot de conception basé sur les modèles [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/990/

UNIVERSITÉ DE MONTRÉAL

INTÉGRATION D'UN SIMULATEUR DE PARTITIONNEMENT SPATIAL ET TEMPOREL À UN FLOT DE CONCEPTION BASÉ SUR LES MODÈLES

JULIEN SAVARD

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE EN SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

DÉCEMBRE 2012

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

INTÉGRATION D'UN SIMULATEUR DE PARTITIONNEMENT SPATIAL ET TEMPOREL À UN FLOT DE CONCEPTION BASÉ SUR LES MODÈLES

présenté par : SAVARD Julien

en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquée

a été dûment accepté par le jury d'examen constitué de :

Mme NICOLESCU Gabriela, Doct., présidente

M. BOIS Guy, Ph.D., membre et directeur de recherche

M. BOLAND Jean-François, Ph.D., membre et codirecteur de recherche

M. ZHU Guchuan, Doct., membre

REMERCIEMENTS

Dans un premier temps, j'aimerais remercier mon directeur Guy Bois, ainsi que mon codirecteur Jean-François Boland, pour leur support tout au long de mes travaux. La latitude dont j'ai bénéficié a été très appréciée, et leurs conseils ont grandement contribués aux résultats de ce projet. Je leur suis également reconnaissant pour leur disponibilité durant les moments cruciaux.

Ensuite, je tiens à remercier l'École Polytechnique de Montréal, le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), le Consortium de recherche et d'innovation en aérospatiale du Québec (CRIAQ), le programme de stage du réseau de chercheurs en Mathématiques des technologies de l'information et des systèmes complexes (MITACS), ainsi que les partenaires industriels du projet AVIO 509, en particulier CMC Électroniques, pour leur contribution à la fois financière, matérielle et humaine dans le cours de mes travaux. Celle-ci a été essentielle tout au long de mes activités de recherches.

Je veux également remercier tous mes collègues de l'équipe AREXIMAS et de Polytechnique, MM. Oudet, Legault, Bao, Leblanc, Carmel-Veilleux, Beaulieu, McKinnon et Rogers-Vallée, ainsi que Mlle Benyoussef. Leurs questions, commentaires, collaborations et amitiés ont rendu tous ces défis plus motivants et agréables.

Finalement, je veux remercier tout particulièrement ma Catherine chérie, qui m'a aimé, motivé et encouragé durant mes études; ma famille, pour son amour et soutient indéfectible depuis toujours; et mes sages amis David et Philippe qui m'ont sensibilisé aux défis et aux réalités des études supérieurs du début à la fin.

RÉSUMÉ

L'architecture avionique modulaire intégrée (IMA) représente une préoccupation cruciale pour l'industrie aérospatiale dans le développement de systèmes de plus en plus complexes, afin de réduire les coûts, ainsi que les temps, de développement, de certification et de production.

D'un point de vue logiciel, cet objectif pousse les développeurs à développer ou migrer une multitude d'applications vers des systèmes d'exploitation temps réel (RTOS) conformes à la norme ARINC 653. Cette norme propose un partitionnement dans l'espace et dans le temps sécuritaire pour les systèmes critiques, un élément crucial aux IMA.

Toutefois, le prix des licences pour les principaux environnements de développement commerciaux peut être très élevé. Il devient donc intéressant de s'attarder aux alternatives moins dispendieuses qui pourraient très bien être utilisées en début de développement aux fins de simulations, préalablement au déploiement sur la plateforme cible. D'un autre côté, plusieurs alternatives offrent peu de documentation ou de support, et sont souvent limitées quant aux approches de développement basé sur les modèles, ou quant à la conformité à la norme ARINC 653.

Pour répondre à cette problématique, ce projet se concentre sur ces environnements de développement peu coûteux ou libres de licences, et propose un flot de conception novateur incluant à la fois un environnement de modélisation efficace pour l'analyse basée sur les modèles, ainsi qu'un environnement de simulation.

Le flot proposé utilise le langage « Architecture Analysis and Design Language (AADL) » pour modéliser le système, et le simulateur commercial de système IMA (SIMA) développé par l'entreprise GMV pour exécuter les applications. Ce simulateur est conforme à la norme ARINC 653, et s'exécute sur un ordinateur de bureau par dessus un système d'exploitation Linux comportant un noyau temps-réel. Pour faire le pont entre les deux environnements, le générateur de code libre OCARINA, qui prend en entrée du AADL, a été étendu pour réaliser la génération de fichiers de configurations, et de codes sources, vers la cible SIMA. Le code source généré vise la gestion des appels aux services de l'interface de programmation ARINC 653.

Une application avionique a été développée en tant qu'étude de cas pour expérimenter ce flot. Elle consiste en une unité de contrôle et d'affichage multiusage (de l'anglais : « Multi-purpose Control and Display Unit » ou MCDU) communiquant avec un système de gestion de vol simulé fourni par CMC Électronique. Durant l'expérimentation, le simulateur s'est démontré utile en permettant l'identification et la correction d'erreurs de conception dans la configuration et dans l'implémentation du MCDU, ce qui a réduit considérablement les erreurs de transmission de pages. Il a aussi été démontré qu'il pouvait être déployé sur une plateforme dotée d'une distribution Linux embarquée.

Concernant le générateur étendu, les résultats furent concluants. La version actuelle de l'outil réduit considérablement le temps alloué à la configuration du système, et à la migration d'application vers l'environnement ARINC 653. De plus, le simulateur SIMA a dorénavant accès à une approche de développement basé sur les modèles. Par contre, des limitations fondamentales ont été identifiées quant à la génération de code source. Néanmoins, nous considérons que le flot que nous proposons est un point de départ satisfaisant qui pourra être étendu à d'autres technologies dans le cadre de futurs travaux.

ABSTRACT

The Integrated Modular Avionics (IMA) architecture has been a crucial concern for the aerospace industry in developing more complex systems, while seeking to reduce cost as well as development, certification and production time.

From a software perspective, that objective pushes developers to develop or migrate most applications toward real-time operating systems (RTOS) compliant to the ARINC 653 standard which offers a safety critical space and time partitioning central to IMA.

However, due to very high license costs, mainstream commercial development environments can be restrictive. That situation is even more striking considering low-cost alternatives could instead be used in early simulation, before deployment on target platform. On the other hand, many alternatives offer little documentation or support, and are often limited when it comes to either model-based engineering (MBE) approach, or compliance to the ARINC 653 standard.

To answer that problematic, this project reviewed existing low-cost and open-source development environments, and proposes a novel flow including both a modeling environment effective for model-based analysis and a simulation level.

The proposed flow uses the Architecture and Analysis Design Language (AADL) to model the system, and the commercial Simulated IMA (SIMA) simulator developed by GMV to execute its applications. That simulator is ARINC 653 compliant, and runs on a desktop computer over a Linux distribution with a real-time kernel. To bridge the two environments, the open-source OCARINA generator, which takes AADL inputs, was extended to achieve source code and configuration generation toward the SIMA target. The generated source code aims to manage calls to the ARINC 653 programming interface services.

An avionic application was developed as a case study to experiment the latter flow. It consisted of a Multi-purpose Control and Display Unit (MCDU) communicating with an external Flight Management System (FMS) simulation provided by CMC Electronics. During the experiment, the simulator proved useful in leading to the identification and correction of design flaws in the MCDU system configuration, which considerably reduced page transmission failures. It also demonstrated that it could be deployed on a platform with an embedded Linux distribution.

Concerning the extended generator, results have proved successful so far. The current version of the tool greatly reduces the time required to configure the system or migrate applications to an ARINC 653 environment. It also enhances the simulator with a MBE approach. However, fundamental limitations were identified as far as source code generation is concerned. Nevertheless, we consider our proposed flow to be a satisfying starting point, which could be extended to other technologies in future work.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AADL	Architecture Analysis and Design Language
AIR	ARINC 653 Interface in RTOS (or RTEMS)
APEX	ARINC 653 Application Executive
API	Application Programming Interface
ASSERT	Automated proof-based System and Software Engineering for Real-Time systems
BSD	Berkeley Software Distribution
BSP	Board Support Package
EDF	Earliest Deadline First
ESA	European Space Agency
FIFO	First In, First Out
FMS	Flight Management System
GNAT	GNU NYU Ada Translator *(no longer applies)*
GNU	GNU's Not Unix
GPL	General Public Licence
GUI	Graphic User Interface
HAL	Hardware Abstraction Layer
HM	Health Monitoring
IMA	Integrated Modular Avionics
IMADE	Integrated Modular Avionics Development Environment
LLF	Least Slack Time
MARTE	Modeling and Analysis of Real-Time and Embedded Systems
MASIW	Modular Avionics System Integrator Workplace
MCDU	Multipurpose Control and Display Unit

MBE	Model-Based Engineering
MMU	Memory Management Unit
MOS	Module Operating System
OS	Operating System
OSATE	Open Source AADL Tool Environment
PAL	POS Adaptation Layer
PCS	Plan Configuration de Système
PMK	Partition Management Kernel
POK	PolyORB Kernel
POS	Partition Operating System
PPC	PowerPC
REAL	Requirement Enforcement Analysis Language
ROM	Read-Only Memory
RTEMS	Real-Time Executive for Multiprocessor Systems
RTOS	Real-Time Operating System
SIMA	Simulated Integrated Modular Avionics
TASTE	The ASSERT Set of Tools for Engineering
UML	Unified Modeling Language
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High Speed Integrated Circuits
XML	Extensible Markup Language

LISTE DES ANNEXES

INTRODUCTION

L'architecture des systèmes avioniques dits modulaires intégrés (de l'anglais : « Integrated Modular Avionics » ou IMA) [1, 2] a été conçue en majeure partie pour réduire les coûts inhérents à l'architecture des systèmes dits fédérés, c'est-à-dire le temps de développement, de certification et de production. Ils représentent donc une préoccupation importante pour l'industrie aérospatiale dans un contexte où les systèmes sont de plus en plus complexes.

Sous l'architecture fédérée, chaque fonction avionique est montée et intégrée de façon autonome sur des LRU ou LRM distincts (de l'anglais: « Line-Replaceable Unit » ou « Line-Replaceable Module »). Bref, chaque fonctionnalité ou application dispose de sa propre plateforme matérielle.

L'architecture IMA propose plutôt un environnement partitionné à haute intégrité qui est l'hôte de plusieurs fonctions avioniques ayant différents niveaux de criticité, et ce sur une plateforme matérielle partagée. Plusieurs applications peuvent donc s'exécuter sur cette même plateforme, dans la mesure où le partage des ressources est prédéfini et déterministe, et où l'isolation temporelle et spatiale entre celles-ci est assurée. Cette isolation implique que les applications ne peuvent d'aucune façon, et ce même en cas de fautes, affecter l'espace mémoire utilisé par les autres, ni retarder leur période d'exécution. Pour ce faire, des mécanismes de partitionnement logiciel robustes, comme ceux définis dans la norme ARINC 653 sont inhérents à cette architecture.

La norme ARINC 653 [3-9] spécifie les fonctionnalités qu'un environnement d'exécution, ou un système d'exploitation (de l'anglais : « Operating System » ou OS), doit supporter pour garantir un partitionnement spatial et temporel robuste dans les systèmes de bord (en anglais : « onboard systems ») IMA, c'est-à-dire pour assurer la protection et la séparation fonctionnelle entre les applications logicielles. Ce partitionnement est très utile pour la contention des fautes, la facilité de vérification, la validation et la certification.

Malheureusement, les systèmes d'exploitation commerciaux supportant cette norme, comme

VxWorks 653 [10] de Wind River, PikeOS [11] de Sysgo, ou Integrity-178B [12] de Greenhill, sont très dispendieux, alors que des environnements alternatifs moins coûteux, voir libres de licence, pourraient être tout à fait acceptables pour des simulations début de développement. Cette approche permettrait entre autres de vérifier l'équivalence entre le code implémenté et la spécification initiale [13], ou encore, de prototyper, développer et tester des applications avant d'avoir accès à la plateforme cible [5].

En effet, plusieurs environnements d'exécution ARINC 653 alternatifs existent déjà, ou sont en cours de développement. On peut penser aux projets AIR-II, XtratuM, et POK qui respectent assez fidèlement la norme ARINC 653, tant du point de vue du partitionnement spatial et temporel que des services logiciels offerts, et qui sont tous libres de licence, à l'exception de AIR-II qui ne l'est que partiellement.

Parallèlement, ces environnements visent de plus en plus à s'intégrer à des flots de conceptions utilisant une approche de développement basée sur les modèles (de l'anglais : « Model-Based Engineering» ou MBE). L'utilisation d'outils supportant ce type de développement facilite grandement la détection d'erreurs en début de développement, tant par la possibilité d'effectuer des analyses de faisabilité directement sur les modèles, que par la réduction des erreurs humaines en simplifiant la configuration des systèmes et la création de code pour l'intégration d'applications à l'environnement ARINC 653.

Malheureusement, il est fréquent que les alternatives aux environnements d'exécution commerciaux soient insatisfaisantes soit pour des raisons techniques, ou pour des raisons d'accessibilité et de support.

L'absence d'un flot de développement de systèmes IMA qui supporterait le MBE et qui intègrerait un environnement d'exécution ARINC 653 permettant une simulation ou une exécution des systèmes partitionnés en début de développement, et cela à faibles coûts, est donc une lacune importante.

Ainsi, le présent projet vise à résoudre cette problématique en s'attardant à ces environnements

peu dispendieux ou libres de licence, et en proposant un flot de conception novateur. Celui-ci inclut un environnement de modélisation permettant l'analyse basée sur les modèles, ainsi qu'un environnement de simulation.

Le flot proposé permet de modéliser le système à l'aide du langage « Architecture Analysis and Design Language (AADL) » [14], et d'exécuter celui-ci sur le simulateur commercial de système IMA (SIMA) [5, 15] développé par l'entreprise GMV [16]. Ce simulateur est conforme à la norme ARINC 653, et s'exécute sur un ordinateur de bureau par dessus un système d'exploitation Linux doté d'un noyau temps-réel. Puisque SIMA ne dispose pas d'un environnement de modélisation AADL, le générateur de code libre OCARINA [17], qui utilise ce langage, a été étendu pour réaliser la génération de fichiers de configurations et de codes sources vers la cible SIMA. Le code source généré vise la gestion des appels aux services de l'interface de programmation ARINC 653.

Finalement, une étude de cas plus complexe est réalisée pour expérimenter à la fois le simulateur et le flot de conception proposé. Celle-ci consiste à développer une unité de contrôle et d'affichage multiusage (de l'anglais : « Multi-purpose Control and Display Unit » ou MCDU) [18] s'exécutant sur SIMA et communiquant via protocole TCP/IP avec un système de gestion de vol (de l'anglais : « Flight Management System » ou FMS) externe, fournit par CMC Électronique [19].

Le corps de ce mémoire est structuré comme suit:

- Le chapitre 1 présente le contexte du projet et la problématique de recherche
- Le chapitre 2 présente la revue de littérature pertinente au problème
- Le chapitre 3 présente le flot de conception proposé
- Le chapitre 4 présente les détails de l'étude de cas
- Le chapitre 5 présente les résultats obtenus

En définitive, nous concluons avec une synthèse de nos travaux et des suggestions pour des avenues de recherches futures.

Ce projet a été réalisé dans le cadre du projet CRIAQ AVIO-509 visant l'exploration architecturale des systèmes IMA (de l'anglais : « ARchitectural EXploration in Integrated Modular Avionics Systems » ou AREXIMAS) [20].

CHAPITRE 1 CONTEXTE ET PROBLÉMATIQUE

1.1 Contexte du projet

1.1.1 Les systèmes avioniques modulaires intégrés

Tel qu'indiqué précédemment, l'architecture IMA propose un environnement partitionné à haute intégrité qui est l'hôte de plusieurs fonctions avioniques caractérisées par différents niveaux de criticité, et ce sur une plateforme matérielle commune.

Les IMA comportent plusieurs avantages. En consolidant le matériel, le filage, et les dispositifs de refroidissement, ce type d'architecture permet de réduire le poids et la puissance dissipée de l'ensemble des systèmes.

De plus, en proposant un bassin de ressources de calculs, d'entrée/sortie et de mémoire qui est partagé entre les fonctionnalités avioniques, ces ressources peuvent être utilisées plus efficacement, et de façon plus optimale. Cette optimisation est faite via des tables de configuration élaborées par l'intégrateur du système. L'allocation de ressource doit nécessairement être statique puisqu'elle affecte la certification des fonctionnalités hébergées. Si elle change suite à une mise à jour, le système doit être certifié de nouveau.

Ensuite, puisque les développeurs investissent moins de temps à développer les ressources de traitement, on assiste également à une consolidation des efforts de développement, ce qui économise temps, argent, et réduit la durée du cycle de conception. Cela facilite également le travail des développeurs au niveau de la certification et du développement.

Finalement, puisque plusieurs éléments de l'architecture IMA ont été normalisés à travers une série de normes (i.e. la série ARINC) qui sont non-propriétaires, ceux-ci sont davantage portables. Cela permet dorénavant aux petites compagnies de pouvoir intégrer le marché des équipements ou logiciels avioniques, ce qui améliore ainsi la compétitivité et l'efficacité de développement.

Pour davantage de détails sur les IMA, les travaux de Carmel-Veilleux [21], membre du projet CRIAQ AVIO-509, peuvent être consultés.

1.1.2 La norme ARINC 653

La norme ARINC 653 vise à assurer la protection et la séparation fonctionnelle entre les applications logicielles, en spécifiant notamment les fonctionnalités qu'un environnement d'exécution doit supporter. Cette section en présentera les concepts les plus importants.

1.1.2.1 Structure de la norme

Pour fins de référence, précisons que la norme ARINC 653 est divisée en quatre parties:

- **Partie 1 – Les services requis** : Présentation de l'APEX, des services de partitionnements essentiels et des mécanismes de configuration [6].
- **Partie 2 – Les services additionnelles *(en anglais : « Extended services »)*** : Présentation des services additionnels, notamment de l'ordonnancement de module multiple, et des journaux de bord [7].
- **Partie 3 – La spécification des tests de conformités** : Présentation de la spécification des processus de test permettant de démontrer que le comportement de l'interface d'un OS est conforme à la norme ARINC 653 [8].
- **Partie 4 – Sous-ensemble de services**: Présentation d'un sous-ensemble réduit des services de la partie 1, où des restrictions sont proposées sur ceux-ci. Définition des données échangées via services ou via configuration [9].

1.1.2.2 Système d'exploitation du module (MOS)

Sous ARINC 653, les applications avioniques d'un système, qu'il y en ait une ou plusieurs, sont hébergées, et exécutées de façon indépendante, par une unité de traitement appelé un module.

C'est l'OS de ce module (en anglais : « Module Operating System » ou MOS) qui réalise le partitionnement spatial et temporel des applications via un ordonnancement fixe, prédéfini, et cyclique, contenu dans un cadre temporel principal (en anglais : « Major time frame ») d'une durée fixe qui est répété périodiquement lors de l'exécution. Chaque partition se voit assigner un ensemble de fenêtres d'exécution où son application obtient le monopole du processeur.

Parmi les services additionnels rajoutés dans la partie 2 de la norme, on retrouve également la possibilité d'ordonnancer plusieurs modules différents selon une stratégie appelée « Multiple Module Scheduling ». Il est ainsi possible de modifier cycliquement l'ordonnancement des partitions durant l'exécution du système, selon un patron prédéfini et statique.

1.1.2.3 Système d'exploitation de partition (POS)

À l'intérieur d'une partition, l'application qui s'exécute a généralement besoin d'un environnement en mesure de lui fournir un second niveau d'ordonnancement pour gérer les différents processus qui la composent, ainsi que de lui fournir d'autres services habituellement offerts par un OS, comme la communication intra-partition, l'allocation dynamique de mémoire, et la gestion des interruptions.

Ainsi, chaque partition comporte généralement un système d'exploitation de partition (en anglais: « Partition Operating System » ou POS) qui ne s'exécute pas sur le matériel, mais par dessus le MOS. Le POS doit par exemple ordonnancer les processus d'une application selon une politique préemptive basée sur la priorité avec une file d'attente de type premier arrivé, premier servi (en anglais: « First in, first out » ou FIFO) pour les processus avec une même priorité.

Ce double niveau d'ordonnancement est parfois associé à la technologie de virtualisation, ce qui explique que certains systèmes d'exploitation ou simulateurs d'environnement ARINC 653 procèdent de cette façon. En ce sens, il est considéré que l'architecture IMA remplace les

éléments matériels par du logiciel, puisque les LRU ou LRM de l'architecture fédérée sont désormais remplacés par des machines virtuelles.

Selon la norme, les modes d'opération d'une partition sont:

- Normal : La partition s'exécute normalement.
- Démarrage à froid: La partition et l'application associée s'initialisent. Ce mode est utilisé pour effectuer un démarrage « lent ». Il peut effectuer des tests requis pour des analyses de sécurité.
- Démarrage à chaud: La partition et l'application associée s'initialisent. L'environnement matériel où démarre la partition peut toutefois être différent du démarrage à froid puisqu'il vise un démarrage « rapide », souvent requis en vol (ex.: pas besoin de copier le code de la mémoire non volatile vers la RAM).
- Au repos (en anglais: « Idle »): Aucune application ne s'exécute. La partition n'est pas initialisée, mais préserve ses fenêtres d'exécution.

1.1.2.4 Interface exécutante de l'application (APEX)

ARINC 653 offre également une interface de programmation normalisée pour les applications avioniques d'un système. Cette interface exécutante de l'application (en anglais: « Application Executive » ou APEX) est fournie aux développeurs d'applications logicielles est composée des requêtes de services définies sous la norme.

Ces services sont regroupés sous les catégories suivantes:

- Gestion de partition
- Gestion de processus
- Gestion du temps
- Communication inter et intra-partition
- Monitorage de la santé du système

Les services de communication inter-partition comprennent la création de ports de file d'attente (en anglais: « Queuing port »), et de ports d'échantillonnages (en anglais: « Sampling port »), tandis que les services de communication intra-partition comprennent notamment des moyens de synchronisation comme des évènements, des tampons de messages, des tableaux noirs (en anglais : « blackboard »), et des sémaphores.

APEX offre également les services nécessaires à la gestion des processus qui peuvent d'ailleurs se trouver dans quatre états:

- Dormant (Inéligible à l'ordonnancement)
- En attente (Attend un évènement ou une ressource pour être prêt à s'exécuter)
- Prêt s'exécuter (Attend le monopole du processeur)
- En cours d'exécution

Un système de journal de bord (en anglais : « logbook ») utilisé pour écrire des messages sur l'état du système fait également parti des services. Il fut rajouté dans la partie 2 de la norme.

C'est généralement la responsabilité du POS de supporter l'APEX et de l'offrir aux développeurs. Celui-ci doit obligatoirement la supporter pour affirmer être conforme à la norme ARINC 653. L'APEX vise la portabilité, la réutilisabilité, et la modularité des applications logicielles, ainsi que l'intégration de logiciels de niveaux de criticité multiples.

Cela dit, même si le comportement et l'interface de l'APEX sont définis dans la norme, l'implémentation est laissée à la discrétion des développeurs d'OS.

1.1.2.5 Monitorage de la santé

Le monitorage de la santé (en anglais : « Health Monitoring » ou HM) du système consiste en un ensemble de mécanismes pour surveiller les ressources et les applications du système.

Le HM contribue à l'isolation des fautes et à prévenir la propagation des erreurs et des pannes, puisqu'il vise à fournir de solides mécanismes de récupération suite à une erreur pour assurer qu'un échec ne produira pas de pertes permanentes. Dans le cadre de la norme ARINC 653, les fonctions de HM sont définies au niveau système, au niveau partition, et au niveau des processus.

La configuration et les détails des réactions du HM sont définis dans les tables, ou fichiers, de configuration du système pour les erreurs gérées au niveau du système (i.e. du module) et au niveau de la partition, tandis qu'ils sont définis dans des processus de traite-erreurs (en anglais: « error handlers ») pour les erreurs gérées au niveau de l'application (i.e. des processus).

1.1.2.6 Fichiers de configuration

La section 5.0 de la partie 1 de la norme ARINC 653, définit un format pour les tables, ou fichiers, de configuration d'un OS ARINC 653. La structure et les types des données nécessaires à la configuration sont explicités. Le format de fichier retenu est le XML.

Les structures, propriétés, et autres éléments existants dans ces fichiers ne peuvent pas être altérés par l'OS pour conserver la conformité à ARINC 653. Par contre, il est possible de les étendre en rajoutant des éléments de configuration additionnels. Pour des fins de portabilité, il est toutefois conseillé d'éviter ce rajout d'éléments personnalisés autant que possible.

1.1.3 L'ingénierie basée sur les modèles

Les approches de développement basé sur les modèles (de l'anglais : « Model-Based Engineering» ou MBE) [22] sont dorénavant considérées très utiles pour faciliter le

développement de systèmes critiques. Celles-ci proposent un cadre conceptuel permettant la capture, la validation et l'implémentation des systèmes via des modèles. Elles visent à accroître l'efficacité et la productivité du processus de développement système et logiciel [23, 24].

Dans le cadre du développement de systèmes, ou de MBE, les modèles ont depuis longtemps joué un rôle important dans la représentation des notions issues du monde réel, ainsi que des notions plus abstraites. À la base, ils sont très utiles aux fins de documentation en structurant l'information caractérisant un système en cours de conception. Ils peuvent également être utilisés à des fins génératives, c'est-à-dire dans le but de générer des fichiers de configuration, du code source, de la documentation, des cas de tests, etc. Finalement, ils peuvent être utilisés pour faire de la validation tôt dans le processus de développement avant même l'implémentation, notamment via la vérification de la faisabilité de l'ordonnancement, ou de l'allocation des ressources [25, 26]. Cette approche permet donc d'analyser et de détecter les erreurs tôt dans le cycle de développement, ce qui est crucial puisqu'au moins 70% des erreurs seraient introduites durant la spécification, avant même l'implémentation [24]. Ces validations peuvent habituellement tenir compte autant des aspects fonctionnels que non-fonctionnels qui ont été modélisés et visent à réduire les coûts et les risques.

Les systèmes ARINC 653, qui utilisent fortement des tables de configuration, peuvent également bénéficier de ces techniques puisque la génération de code ou de configuration réduit la possibilité d'erreurs humaines [23]. Celles-ci peuvent provenir d'une mauvaise compréhension de la spécification, ou encore d'une erreur de code, et peuvent se reproduire à chaque modification à la spécification [13] suite à la détection d’un problème.

Certains outils de génération de code peuvent toucher des aspects liés au concept de temps-réels, ainsi qu’aux communications, qui sont importantes lors d’un processus d’intégration. Dans certains cas, il n'est plus nécessaire d’implémenter le formatage ou le décodage de message, la création de tâches, la configuration du système d’exploitation, le partage des ressources, et la synchronisation. Actuellement, l’intégration est encore faite manuellement dans la plupart des cas, ce qui peut devenir couteux, par exemple lors de modifications aux interfaces [27].

1.2 Problématique

Notre problématique de recherche s'articule donc autour de la question suivante: « Est-il possible d'obtenir un flot de conception de système IMA faisant usage d'une approche de MBE, et qui comprendrait un environnement d'exécution ARINC 653 à faible coût pour réaliser des simulations en début de développement? »

Tel que précisé plus tôt, le problème est motivé par le coût élevé des systèmes d'exploitation commerciaux qui sont conformes à la norme ARINC 653, et par l'intérêt grandissant pour les stratégies de développement basées sur les modèles.

Dans le cadre du projet CRIAQ AVIO-509, les partenaires industriels du domaine de l'avionique ont démontré leur intérêt pour un tel flot de développement qui pourrait faciliter l'intégration d'applications vers les environnements partitionnés, et qui investiguerait les alternatives aux systèmes commerciaux. Cette méthode faciliterait également l'exploration architecturale du logiciel via simulation.

Ce problème a été identifié au cours des trois dernières années lors de réunions entre les chercheurs de notre équipe de projet.

1.3 Objectifs

La résolution de cette problématique repose sur trois objectifs.

1) Le premier objectif est de développer les connaissances et le savoir-faire de notre groupe de recherche sur les outils, les concepts, et les environnements de développement relatifs aux systèmes IMA.

Ce dernier est important considérant que le savoir-faire autour des systèmes d'exploitation conformes à la norme ARINC 653 est peu développé dans le milieu académique canadien, et que ces connaissances sont nécessaires à la résolution de la problématique.

2) Le second objectif vise à proposer la composition d'un flot de conception pour les systèmes IMA qui est peu dispendieux, et doté d'environnements de modélisation et de simulation. Il vise également à détailler et réaliser les travaux d'implémentation, et d'intégration nécessaires à son obtention.

La proposition de ce flot sera envisageable sachant que nous aurons procédé à une synthèse des connaissances et identifié les lacunes, les avantages, et la complémentarité éventuelle entre les outils et les technologies susceptibles d'être utilisés.

3) Finalement, le troisième objectif vise à explorer sommairement les limitations et le potentiel des environnements de modélisation et de simulation du flot de conception proposé via une étude de cas constituée d'un système et d'une application avionique.

Cet objectif permettra d'illustrer si la solution amenée dans le cadre de ce projet offre des possibilités intéressantes et mérite d'être reprise par d'autres projets. Cela identifiera aussi des éléments plus limitatifs qui devront plutôt être perfectionnés. Puisque notre environnement académique ne dispose pas d'une application avionique typique pouvant servir d'étude de cas, précisons que celle-ci devra être implémentée dans le cadre de ce projet.

1.4 Méthodologie

La réalisation du premier objectif se fondera avant tout sur une revue de littérature exhaustive, ainsi que sur des expérimentations, s'il y a lieu, avec les technologies rencontrées.

La réalisation du second objectif sera un travail de réflexion et d'analyse, afin d'identifier ce que serait un flot de conception idéal et conforme aux objectifs visés par le présent projet. Certains travaux d'implémentation seront possiblement entrepris si nécessaire, pour l'obtention d'un tel flot.

Finalement, pour la réalisation du troisième objectif, l'implémentation de l'application avionique se fera en partie avec l'appui des normes ARINC concernées, ainsi que du support personnel et technique de nos entreprises partenaires. Elle sera ensuite utilisée comme étude de cas afin d'illustrer, entre autres, le fonctionnement du flot proposé.

1.5 Contribution

De la réalisation de nos travaux de recherche, et de l'atteinte des objectifs mentionnés, résulteront les contributions suivantes:

- ✓ Une revue de littérature détaillée des différents environnements existant et des projets en cours. Ceux-ci seront mis en perspective, ce qui comblera le manque de connaissances et de savoir-faire autour de ce domaine dans notre groupe de recherche.

- ✓ Une proposition de flot de conception novateur qui mitigera les désavantages habituellement répertoriés chez les environnements libre de licence ou peu coûteux. Celui-ci intéressera grandement nos partenaires industriels.

- ✓ L'implémentation d'une application avionique dans un environnement conforme à ARINC 653. Ce système prototype pourra être réutilisé pour des projets ultérieurs.

- ✓ Les problèmes rencontrés lors de la réalisation de l'étude de cas et du port de l'application vers l'environnement ARINC 653 seront répertoriés et détaillés pour de futurs projets de recherches qui souhaiteraient perfectionner le flot proposé.

- ✓ La présentation d'un poster d'équipe récipiendaire de la 1ère place pour les projets en cours au gala CRIAQ 2012.

- ✓ La présentation d'un poster récipiendaire de la 3e place au « Forum des Étudiants en Aérospatiale 2012 ».

CHAPITRE 2 REVUE DE LITTÉRATURE

Ce chapitre présente un survol des environnements d'exécution alternatifs, c'est-à-dire peu coûteux ou libres de licence, qui sont conformes ou près de la norme ARINC 653. Nous passerons également en revue quelques technologies et projets touchant les flots de modélisation. Nous analyserons finalement les limites des environnements étudiés.

2.1 Environnement d'exécution ARINC 653 alternatifs

2.1.1 Architecture de système AIR-II

2.1.1.1 Origine et objectifs

Initialement, le projet AIR (« ARINC 653 Interface in RTEMS ») [3] visait à faire un premier pas vers l'usage d'un RTOS de série, gratuit, et de code source ouvert pour l'exécution de systèmes ARINC 653. Ainsi, ce projet a adapté RTEMS pour qu'il supporte l'interface APEX. Ce sont ces modifications qui permettront d'ailleurs l'usage de RTEMS sous d'autres projets, comme XtratuM, présenté à la section 2.1.2. RTEMS fut sélectionné car il était qualifié pour les logiciels de bord des programmes spatiaux [28]. Des modules pour la gestion du partitionnement spatial et temporel furent également développés. Ce projet demeure avant tout une preuve de concept.

Le projet AIR-II [29, 30] (« ARINC 653 Interface in RTOS »), qui est un consortium entre l'agence spatiale européenne (ESA), la Faculté des Sciences de l'Université de Lisbonne (FCUL), Thales, et GMV, visait donc à faire évoluer cette preuve de concept technologique vers un produit industriel. L'objectif principal était d'obtenir un environnement d'exécution ARINC 653 en permettant l'usage des fonctionnalités de la norme dans des RTOS de série tout en conservant l'indépendance entre le matériel et le OS.

2.1.1.2 Description

AIR comporte quatre composants :

1) Le noyau de gestion des partitions (en anglais : « AIR Partition Management Kernel (PMK) ») faisant office de MOS
2) Un noyau de système d'exploitation temps-réel par partition (i.e. un POS)
3) Une interface APEX pour les services ARINC 653
4) Un moniteur de santé

2.1.1.2.1 MOS

Le PMK est un micronoyau simple qui gère efficacement l'ordonnancement et la répartition des partitions, ainsi que les communications inter-partition. Il fait office de MOS ARINC 653. L'ordonnanceur de partitions propre à PMK s'occupe de l'isolation temporelle et de la préemption des partitions. Le répartiteur de partitions de PMK s'occupe de son côté d'enregistrer et de restaurer les contextes et d'informer les partitions du temps écoulé depuis leur dernière préemption. Considérant que chaque partition possède sa propre page mémoire, il s'agit surtout ici de changer le pointeur et son espace d'adressage [31]. Précisons aussi que AIR-II supporte la Partie 2 de ARINC 653, soit l'ordonnancement de modules multiples, c'est-à-dire la capacité de changer entre plusieurs configurations d'ordonnancement pour les partitions. La section 4.2.1 du mémoire de Craveiro contient beaucoup de détails à ce sujet [30]. Le PMK est le seul composant qui s'exécute en mode privilégié. Ce dernier peut s'exécuter sur les plateformes à architecture SPARC, c'est-à-dire la famille de processeurs LEON. Bien qu'un port x86 existe, celui-ci n'est pas aussi avancé que la version SPARC. Un port vers une architecture PowerPC est aussi dans les plans[1].

La configuration d'un système sous AIR-II se fait via un fichier XML selon le format défini sous la norme ARINC 653. Ce fichier pourra cependant être généré via un outil de configuration appelé CONFIGUIMA (voir section 2.1.1.5).

---

[1] C. Silva, GMV, Discussion électronique via courriel, 15 mai 2012

2.1.1.2.2 POS

Le POS peut être n'importe quel RTOS supportant POSIX qui est préalablement porté vers l'environnement AIR. L'intégration d'un nouvel RTOS se fait sans changements fondamentaux. Seul le processus d'initialisation de l'OS et le gestionnaire de l'horloge du système doivent être adaptés. Pour s'assurer que le PMK n'a pas à être modifié à chaque fois, ce qui amènerait un travail de validation ou de certification additionnel, les modifications ne se font que sur une couche d'adaptation pour POS (en anglais : « POS Adaptation Layer » ou PAL). Le PAL représente la couche faisant le pont entre le POS et l'interface APEX, ainsi qu'entre le POS et le PMK. Le but du PAL est de rendre le RTOS indépendant du PMK. En date du 15 mai 2012, les RTOS supportés étaient les versions 4.8, et 4.1x de RTEMS, ainsi que sa version qualifiée pour l'espace, soit RTEMS-impr[2]. RTEMS est présenté en détails en 2.1.2.2.2.1. Il est également possible d'avoir une partition sans POS, où le code usager n'utilise qu'un petit sous-ensemble des appels système de PMK, ce qui peut-être utile pour du code très critique. Une version Linux Embarqué [32] est également dans les plans. Précisons qu'il est aussi possible d'avoir des partitions dites « systèmes » avec davantage de privilèges qui permettent l'implémentation de modules d'entrées/sorties [31].

2.1.1.2.3 APEX

Tel que mentionné en 1.1.2.4, l'interface APEX implémente les services définis dans ARINC 653. Sous AIR-II, elle consiste en deux composants :

1) La couche de l'interface APEX qui fournit aux applications les services APEX.
2) La couche du noyau APEX, qui implémente une version portable de l'interface APEX [4] à l'aide la bibliothèque POSIX. Cette couche est également appelée AIRPEX. POSIX a été sélectionnée en raison de sa disponibilité sur une vaste gamme d'OS modernes. Cette couche noyau fait le lien entre l'interface APEX et le PAL. Elle peut aussi faire

[2] C. Silva, GMV, Discussion électronique via courriel, 15 mai 2012

directement le lien entre l'interface APEX et le PMK, sans passer par le POS, pour certaines fonctionnalités. En effet, pour la communication inter-partition, cette couche du noyau copie le message à envoyer dans un tampon mémoire local représentant le port source. Celui-ci est ensuite transféré par le PMK vers le tampon local du port de destination de la partition concernée.

2.1.1.2.4 Monitorage de la santé

Finalement, le moniteur de santé de AIR-II assure, autant que possible, la non-propagation des erreurs entre les partitions. Celui-ci gère les échéances ratées, notamment sur des contraintes temps-réel dures, ainsi que les violations sur la mémoire et les problèmes matériels. Il peut aussi remplacer des traite-erreurs du RTOS (POS) par les siens, ou encore de rajouter une étape de traitement avant ou après le traite-erreurs du RTOS.

2.1.1.3 Licence

Les composants du projet AIR-II ne sont pas tous couverts par la même licence[3]. La version portable de l'interface APEX, soit AIRPEX, n'est pas un logiciel libre et est un produit propriétaire à GMV. Tel que mentionné, elle supporte complètement la norme ARINC 653. C'est notamment cette même interface, utilisée avec un logiciel de bas niveau supportant une cible matérielle (en anglais : « board support package » ou BSP) différente, qui est utilisée sur leur simulateur SIMA présenté à la section 2.1.4.

Par contre, le projet AIR-II a également développé une autre interface appelée IMASPEX, destinée aux applications spatiales, qui comprend un sous-ensemble de l'interface APEX, ainsi que quelques services spécifiques au domaine spatial. Bien qu'elle ne soit pas complètement conforme à la norme ARINC 653, cette interface sera prochainement offerte en tant que logiciel libre selon la même licence que RTEMS, soit la GPLv3, avec pour exception que le code lié aux

[3] C. Silva, GMV, Discussion électronique via courriel, 15 mai 2012

binaires n'aura pas besoin d'être ouvert. Il en sera de même pour le PMK, ses outils de configuration, ainsi que le module d'entrée/sortie pour les partitions supportant Ethernet, Spacewire, MIL-STD-1553B et RS-232.

2.1.1.4 Déploiement

Tel que mentionné plus haut, AIR peut s'exécuter sur les plateformes à architecture SPARC, ainsi que celles à architecture x86, bien que cette dernière version soit moins avancée.

2.1.1.5 Environnement de modélisation

Le projet AIR-II offre un environnement de configuration appelé CONFIGUIMA[4]. Il s'agit d'un greffon d'Eclipse visant une configuration des systèmes ARINC 653 via une interface graphique, afin d'éviter de manipuler manuellement les fichiers de configuration XML. Cet outil permet une vérification automatique des erreurs du système configuré, et offre la possibilité pour plusieurs personnes de travailler simultanément à la configuration en raison d'une interaction avec une base de données MySQL. À la fin du processus, l'outil génère des fichiers de configuration XML compatibles avec le AIR-PMK, mais également avec le simulateur de GMV présenté au chapitre 3.

En date du 16 mai 2012, CONFIGUIMA était toujours sous développement, et était considéré être au niveau « alpha », avec une sortie « bêta » prévue au début de juin 2012.

SIMA, AIR et CONFIGUIMA feront partie de l'environnement de développement de IMA (IMADE) de GMV, où les composants visent à être complètement intégrés et à couvrir complètement le cycle de vie d'une application, soit la configuration, la validation, le développement, la simulation et le déploiement.

---

[4] C. Silva, GMV, Discussion électronique via courriel, 16 mai 2012

Rien n'indique toutefois que cet environnement offrira les mêmes perspectives d'analyses préliminaires ou de génération de code que celles offertes par une approche de MBE. La perspective de portabilité vers d'autres plateformes est aussi inconnue.

2.1.1.6 Feuilles de routes

Tel que mentionné plus tôt, le support du PowerPC, l'intégration d'un nouveau POS, soit une version embarquée de Linux, et le développement de IMASPEX, un sous-ensemble de l'APEX libre de licence, figurent parmi les projets à venir. Faire du PMK un logiciel libre, et la complétion du CONFIGUIMA sont également à l'agenda.

2.1.2 XtratuM

2.1.2.1 Origine et objectifs

Les premières versions de XtratuM avaient pour objectif de permettre à plusieurs instances de RTOS de s'exécuter simultanément sur la même machine. Une forme de partitionnement spatial et temporel était ainsi possible pour les OS comme RTLinux/GPL et MarteOS. Ce n'est que pour les versions ultérieures que l'objectif principal du projet a évolué en la réalisation d'un environnement d'exécution pour les applications spatiales avec partitionnement robuste, similaire à celui que fournissent les OS conformes à la norme ARINC 653 [21].

La version actuelle de XtratuM est un moniteur de machines virtuelles, ou plus précisément un hyperviseur de type 1. Ce dernier peut réaliser un partitionnement spatial et temporel relativement fidèle à ce que nécessite ARINC 653. Cependant, malgré le fait que son interface de programmation pour les applications (en anglais : « Application programming interface » ou API) et ses opérations internes aient été adaptées pour ressembler à la norme ARINC 653, il faut noter que XtratuM ne vise pas à être conforme avec ARINC 653 [33].

Pour consultation, les travaux de Carmel-Veilleux [21] présentent XtratuM avec un haut niveau de détails.

2.1.2.2 Description

2.1.2.2.1 MOS

En tant qu'hyperviseur, XtratuM est en soit similaire à un MOS. Il assure une isolation spatiale et temporelle robuste : les partitions ne peuvent pas prendre le contrôle d'une ressource critique du système. C'est le noyau qui permet d'avoir accès aux ressources matérielles, c'est-à-dire aux périphériques.

Sous cet hyperviseur, les partitions sont réalisées via des machines virtuelles autonomes. Chacune d'entre elles est associée à une tâche s'exécutant en mode usager, qui est ensuite préemptée par l'hyperviseur au moment indiqué selon l'ordonnancement défini. Cette méthode est contraire à d'autres OS conformes à ARINC 653 qui réalisent plutôt les partitions en isolant des groupes de tâches à l'aide de restrictions dans le noyau. XtratuM ne gère pas directement de tâches [21, 33].

La configuration du système sous XtratuM se fait via un fichier de type XML appelé le « Plan de configuration de système (PCS) », qui suit de très près le modèle de la section 5.0 de la norme ARINC 653. Outre les éléments configurables de base (ex. : définition de l'ordonnancement), ce fichier détaille la gestion des erreurs, l'assignation de différentes interruptions matérielles aux partitions, et les paramètres de configuration pour les pilotes. Un outil appelé « xmcparser » permet de valider la configuration avant la compilation.

Avant qu'il ne supporte le processeur LEON 3, l'hyperviseur ne disposait d'aucune unité de gestion de la mémoire (en anglais « Memory Management Unit » ou MMU). Tel que mentionné dans Carmel-Veilleux [21], cela générait certaines lacunes :

- La possibilité pour les partitions d'avoir accès en lecture à la mémoire d'une autre partition, ce qui rendait possible l'espionnage industriel si deux partis différents étaient responsables de deux partitions d'un même système.
- L'utilisation d'un mécanisme d'hyperappel (une forme de routine d'interruption non-préemptible) pour réaliser la communication inter-partition et le monitorage de la santé du système. Cette méthode était très coûteuse en temps et pouvait causer des problèmes lors des changements de partition si un de ces hyperappels était en cours.
- L'absence de protection en lecture en dehors des zones mémoire implémentées, ce qui pouvait générer des erreurs d'accès aux données dont il était très difficile d'identifier la source.
- L'impossibilité d'utiliser de la mémoire virtuelle.

Toutefois, ces limitations ont disparues avec l'utilisation de ce processeur. L'isolement spatiale entre les partitions est maintenant complet, autant en lecture qu'en écriture. XtratuM supporte dorénavant des sections de mémoires partagées entre les partitions, ce qui permet d'implémenter des mécanismes de communication inter-partition plus efficaces au lieu de s'en remettre aux hyperappels. L'hyperviseur est en mesure de gérer de la mémoire virtuelle. Aussi, il sera dorénavant possible de porter vers XtratuM des POS plus complexes (ex. : Linux) qui nécessitent un MMU pour s'exécuter.

2.1.2.2.2 POS

XtratuM ne prend pas en charge le deuxième niveau d'ordonnancement propre aux partitions. Pour obtenir un environnement d'exécution avec ordonnancement hiérarchique similaire à la norme, XtratuM doit être utilisé en combinaison avec un POS étranger. Ainsi, XtratuM se compare au PMK du projet AIR-II. À l'instar de quelques équivalents commerciaux, précisons qu'un même système XtratuM peut exécuter simultanément des POS différents sur ses différentes partitions. Les POS conformes à ARINC 653 pouvant s'exécuter sur celui-ci sont :

1) RTEMS
2) LithOS

2.1.2.2.2.1 RTEMS

Le « Real-Time Executive for Multiprocessor Systems (RTEMS) » [34] est un RTOS complet supportant une grande variété d'interfaces de programmation. Suite aux travaux du projet AIR [3] présentés en 2.1.1, il est possible de doter ce dernier de l'interface portable ARINC 653 développée par Santos et coll [4]. Cependant, cette interface demeure un produit commercial de GMV. Des travaux sont toutefois en cours pour implémenter une interface ARINC 653 sur RTEMS dont le code source serait ouvert [35].

La version distribuée de RTEMS est une collection d'outils comportant une variété de licences gratuites et ouvertes [36]. La licence couvrant la majorité du code RTEMS est la GNU General Public License telle que publiée par la « Free Software Foundation ». Les versions 2.0 et plus s'appliquent. Il y a également des licences distinctes pour quelques modules, notamment la pile TCP/IP de RTEMS, le support du service réseau RPC/XDR, et le serveur web.

Les grandes décisions sur RTEMS sont prises par un comité directeur guidé par l'objectif de fournir un RTOS déterministe, gratuit, et destiné aux systèmes fortement embarqués. RTEMS entre donc en compétition avec plusieurs produits commerciaux. Il encourage aussi l'usage et le support d'API standardisées afin de promouvoir la portabilité de son environnement.

Tous les usagers peuvent améliorer RTEMS. La version 4.10.2 [37] du RTOS supportait notamment les architectures ARM, Intel i386, MIPS, et SPARC.

Toutefois, dans le contexte où il est utilisé comme une couche au-dessus XtratuM, seul les architectures SPARC ou i386 sont possibles.

2.1.2.2.2.2 LithOS

LithOS [38, 39] est un système d'exploitation s'exécutant sur une machine virtuelle gérée par XtratuM qui est conforme à ARINC 653 et qui supporte toutes les fonctionnalités de la

bibliothèque APEX. Il vise à combler les lacunes de XtratuM en fournissant les primitives nécessaires à la création des ressources du système (structure de tableau noir, tampon mémoire, évènement, sémaphore, etc.), ainsi que les mécanismes pour créer les processus, les compteurs, et l'ordonnanceur de tâches. LithOS utilise les services fournis sous XtratuM pour implémenter ces mécanismes : il est donc indépendant de la plateforme matérielle. Ainsi, LithOS pourrait être perçu comme une simple extension de XtratuM, puisqu'il ne pourrait s'exécuter ni sur un autre hyperviseur, ni en mode autonome. Il se distingue donc de RTEMS sur ce point.

Lorsqu'utilisé en tant que POS, LithOS peut ordonnancer ses processus selon une politique basée sur des priorités fixes où la granularité de son horloge est de 1 nanoseconde. Il implémente aussi l'ordonnancement à modules multiples de la partie 2 de ARINC 653; le changement de la configuration d'ordonnancement se fait via une partition « système » de XtratuM.

LithOS permet aussi l'installation d'un processus de traite-erreurs. Ainsi, les exceptions issues des applications peuvent être définies, levées et gérées en utilisant les services de LithOS. C'est le moniteur de santé inspiré de ARINC 653 défini sous XtratuM qui propage les erreurs vers les partitions fautives.

LithOS implémente aussi des fonctionnalités additionnelles qui ne font pas partie de l'interface APEX, mais qui peuvent être pertinentes pour des systèmes partitionnés. La conformité de LithOS à ARINC 653 a été validée à la suite de tests.

Lors de la compilation, la mémoire et les ressources nécessaires à une partition sont allouées selon le contenu du fichier de configuration. LithOS utilise une empreinte mémoire plus faible que RTEMS lorsqu'utilisé sous XtratuM.

Aucune information n'a été trouvée quant à la licence sous laquelle LithOS est distribué.

2.1.2.2.3 Monitorage de la santé

XtratuM définit un moniteur de santé inspiré de ARINC 653. Celui-ci peut également propager les erreurs vers les partitions fautives. Les POS concernés sont ensuite responsables d'en assurer le traitement.

2.1.2.3 Licence

XtratuM est distribué sous la licence GPL [40], de version 2 et plus, avec une distinction explicitant que du code faisant usage des services de XtratuM n'entre pas dans la catégorie des travaux dérivés. Tout comme pour la licence du système d'exploitation Linux, il est possible de développer des applications propriétaires en utilisant la licence qui convient.

2.1.2.4 Déploiement

XtratuM supporte les processeurs LEON 2 et 3, ainsi que l'architecture i386. En date du 12 avril 2012, le support des architectures ARM et PowerPC est planifié, mais non implémenté.

Pour son exécution, un système sous XtratuM est compilé en une image déployable, où l'on retrouve le chargeur de programmes, le noyau, la table de configuration PCS, et les partitions. Cette image peut ensuite être téléversée sur la ROM de la carte cible. Notons qu'il est possible, sous XtratuM, de faire du traçage pour les partitions et le noyau.

2.1.2.5 Environnement de modélisation

Depuis le 27 juillet 2011, cet hyperviseur est une des cibles du générateur de code OCARINA et de la chaîne d'outils TASTE, qui seront présentés respectivement aux sections 2.2.1.1.1 et 2.2.2.2. Cela accroît son potentiel d'attraction pour les développeurs, et en fait un environnement fertile aux travaux de recherche.

2.1.2.6 Performances

Les performances de XtratuM sont intéressantes, puisque sa résolution temporelle est de 1 microseconde. Cette résolution est d'ailleurs utilisée pour toutes les spécifications temporelles.

2.1.2.7 Feuilles de route

Il est attendu que la version 4.0 de XtratuM supporte le multicœur. Le support des architectures ARM et PowerPC est également envisagé.

2.1.3 POK

2.1.3.1 Origine et objectifs

Le « PolyORB Kernel (POK) » [41, 42] est un projet de recherche qui avait initialement pour objectif d'expérimenter les architectures partitionnées et de construire des systèmes embarqués sécuritaires. Il a été initié durant la thèse de doctorat de Julien Delange à l'École nationale supérieure des télécommunications de Paris, également connue sous le nom de « Télécom ParisTech » [43], avec la collaboration des laboratoires d'informatique de Paris 6 (LIP6) [44].

Plusieurs autres étudiants de l'École pour l'informatique et les techniques avancées (EPITA) [45], en France, ont ensuite joint le projet, ainsi que bien d'autres par la suite, le code source du projet étant ouvert.

POK est un noyau dédié aux systèmes embarqués temps réels dont le but est d'être conforme à plusieurs normes de l'industrie. D'un point de vue avionique, il supporte des fonctionnalités de partitionnement, ainsi que l'API APEX dans les langages C et ADA. Il est ainsi conforme à la norme ARINC 653, ou du moins en partie (voir section 2.1.3.2.3). De l'aveu des développeurs, POK reste pour l'instant un prototype [46].

2.1.3.2 Description

2.1.3.2.1 MOS

POK est composé d'une couche d'abstraction du matériel (en anglais : « Hardware Abstraction Layer (HAL) »), d'un gestionnaire de mémoire, d'un gestionnaire de temps, d'un manipulateur d'erreurs, d'un ordonnanceur assurant l'isolation temporelle et spatiale, ainsi que d'un module de communication inter-partition. Il remplit ainsi toutes les fonctionnalités attendues d'un MOS.

POK se fie à un MMU pour réaliser l'isolation mémoire entre les partitions. Les partitions sont contenues dans des segments mémoires distincts, et ne peuvent accéder à la mémoire des autres partitions. Les mécanismes de protection différents sont utilisés selon l'architecture de la plateforme cible.

Les messages pour communications inter-partitions sont stockés durant l'exécution du cadre temporel principal, et sont disponibles pour lectures lors du cadre temporel suivant. Bref, lorsque toutes les partitions ont eu l'occasion de s'exécuter au moins une fois.

Pour éviter les problèmes de sécurité lors des communications extérieures, l'accès à chaque périphérique est associé à une seule partition lors de la configuration du système. En effet, l'exécution des pilotes de périphériques ne se fait pas dans le noyau, mais sur une partition système, notamment parce que cela limite l'impact d'une panne éventuelle. Si plusieurs partitions souhaitent avoir accès au périphérique, elles doivent communiquer avec celle qui en a le monopole.

La configuration d'un système sous POK se fait via une série de fichiers de définitions qui initialisent les variables importantes du système qui seront utilisées par le noyau. Ces fichiers peuvent cependant être totalement générés par un outil lorsque l'usager fait usage de l'environnement de modélisation de l'OS, présenté en 2.1.3.5, ou en partie, lorsque l'usager utilise un fichier XML de configuration selon le format défini sous la norme ARINC 653.

2.1.3.2.2 POS

Pour l'instant, aucun RTOS étranger ne peut être utilisé comme POS sous POK. Son noyau s'occupe de gérer autant l'exécution des applications de chaque partition, que l'ordonnancement des partitions elles-mêmes. Il remplit ainsi à la fois les rôles du MOS et des POS.

Parmi les politiques d'ordonnancement supportées à l'intérieur d'une partition, on retrouve un algorithme FIFO, et d'autres plus avancés comme l'échéance la plus proche en premier (en anglais : « Earliest Deadline First » ou EDF) et le « Least Slack Time » (LLF).

Pour le développement des applications, POK supporte un sous-ensemble très réduit de la bibliothèque C, à l'exception d'une impressionnante collection de fonctions de calculs.

2.1.3.2.3 APEX

Sous POK, toutes les fonctionnalités de APEX ne sont pas implémentées. Cela s'explique du fait que cet environnement est avant tout utilisé dans le cadre d'un flot de conception incluant des étapes de modélisation et de génération de code, et que ce générateur est en mesure de contourner les fonctionnalités manquantes par différents moyens (ex. : usage de variables globales ou de fonctionnalités alternatives). Pour les applications développées sans utiliser cet environnement, ces limitations peuvent cependant devenir contraignantes, comme il sera présenté à la section 2.3.

2.1.3.2.4 Monitorage de la santé

POK attrape et redirige les erreurs et les exceptions aux traite-erreurs appropriés selon la configuration du système, c'est-à-dire soit au niveau noyau, au niveau partition ou au niveau tâche.

2.1.3.3 Licence

Il est disponible en tant que logiciel libre sous une licence BSD [47] et est maintenu par une communauté académique et industrielle.

2.1.3.4 Déploiement

Les architectures supportées par POK sont PowerPC (PPC) et LEON3. Le processeur x86 simulé sous QEMU est aussi une plateforme disponible. Le déploiement devrait donc pouvoir se faire sur toute carte comportant un de ces processeurs. Aucune garantie n'est toutefois fournie par les développeurs quant à la cible x86, puisque celle-ci n'a pas été testée en dehors du simulateur QEMU faute d'intérêt [48].

Il est possible de développer pour POK autant dans les environnements Linux, Windows, que Mac OS X. Un environnement Linux de distribution Ubuntu 11.10 fut utilisé pour les expérimentations de ce projet.

Un système ARINC 653 complété sous POK est compilé simultanément avec le code source du noyau, ce qui génère un unique fichier binaire d'amorce (« bootable binary ») qui peut être déployé sur une carte (ou un simulateur de processeur) pour exécuter le système. Précisons qu'il est possible de ne pas inclure dans l'image de déploiement les fonctionnalités du noyau qui sont inutilisées par l'application développée. Cela permet de réduire l'empreinte mémoire, ainsi que la couverture de code nécessaire lors des tests.

2.1.3.5 Environnement de modélisation

La configuration des systèmes IMA qui sont développés sous POK, ainsi qu'une partie importante du code de ses applications, peuvent être automatiquement générées à partir d'un modèle défini dans le « Architecture Analysis and Design Language (AADL) » [14]. Son intégration avec cet

environnement de modélisation est sa caractéristique la plus importante, et sera présentée en détail dans la section 2.2.1.1.1.

Dans une moindre mesure, il est également possible pour POK de générer une partie de sa configuration à partir d'un fichier XML de configuration de système ARINC 653 sans passer par la modélisation. Notons toutefois que l'outil responsable de cette transformation n'est pour l'instant qu'une preuve de concept, et n'a pas été validé rigoureusement.

2.1.3.6 Feuilles de route

POK étant un logiciel libre, en date du 24 février 2012 [49], il n'existait pas de feuille de route formelle ou informelle pour le projet. Il existe différentes initiatives autour du projet, mais il est difficile de savoir si celles-ci tentent de coordonner leurs efforts. L'implantation d'un processus de coordination reste à faire.

2.1.4 SIMA

2.1.4.1 Origine et objectifs

Le simulateur SIMA est issu du projet AMOBA [4] (de l'anglais : « ARINC 653 Simulator for Modular Space Based Applications ») initié par l'entreprise portugaise Skysoft, une filiale de GMV (de l'espagnol : « Grupo Mecánica del Vuelo » pour « Groupe de mécanique de vol »), qui a depuis abandonné son nom distinctif [50]. Le projet AMOBA visait à créer un simulateur ARINC653 multiplateforme. C'est lors de celui-ci que les développeurs ont eu l'idée d'implémenter une version standardisée d'APEX via POSIX pour davantage de portabilité. Les objectifs de ce simulateur étaient :

- De permettre d'exécuter et de vérifier des applications ARINC 653
- D'offrir une solution à faible coût
- De ne pas nécessiter un accès à la plateforme cible finale.
- De ne pas exiger l'acquisition d'un RTOS ARINC653.

- D'être portable entre les OS supportant POSIX, et ainsi non-dépendant du noyau de l'OS hôte.

Les résultats du projet ont démontré qu'un tel outil de simulation et de développement était réalisable, et qu'il comblait des besoins préalablement identifiés par des développeurs de systèmes aéronautiques [5]. SIMA est le résultat de l'extension et de l'amélioration de AMOBA.

2.1.4.2 Description

SIMA permet de simuler un système IMA comportant des applications conformes à la norme ARINC 653 sur tout ordinateur de bureau doté d'un système d'exploitation Linux.

À l'instar de POK, et contrairement à la plupart des OS commerciaux, l'ordonnancement hiérarchique sous SIMA n'est pas réalisé par deux couches de système d'exploitation, soit une couche de MOS (ex. : PikeOS, VxWorks 653, etc.) ou d'hyperviseur (ex. : XtratuM), et une couche de POS (ex. : Linux, RTEMS, etc.).

En effet, le système IMA étant simulé, ces deux niveaux de système d'exploitation sont remplacés par des outils et des bibliothèques s'exécutant sur Linux qui imitent leur comportement. Par exemple, les concepts de partitions et de tâches sont modélisés à l'aide de la « Native POSIX Thread library » (NPTL). La section 3.1.1.2.1 présente ces éléments en détail.

SIMA devrait supporter la plupart des distributions Linux, du moment que celles-ci offrent les bibliothèques « librt » et « libpthreads » et qu'elles soient dotées d'une version de noyau supérieure à 2.6. Jusqu'à maintenant, le simulateur a été testé sur les distributions Debian 4.x et 5.x, Fedora 6 à 10, Suse 10 et 11, ainsi que sur Ubuntu 8 à 10. Afin d'obtenir des performances et un comportement plus près de la réalité et de réduire les latences causées par les sections protégées du noyau Linux, il est recommandé d'utiliser une distribution supportant le temps réel dur. Autrement, les sections 2.3 et 3.3. du manuel de SIMA [15] proposent d'appliquer un

correctif logiciel (de l'anglais : « patch ») au noyau de celle-ci afin qu'elle devienne préemptible et puisse simuler un contexte temps réel. Le correctif PREEMPT-RT développé par Ingo Mólnar [51] fut utilisé pour les expérimentations de ce projet.

Les activités de configuration et d'exécution doivent être réalisées manuellement par l'usager en modifiant des fichiers de script console, de configuration et de compilation. Ainsi, SIMA ressemble davantage à une suite de composants et d'outils qu'à un environnement de développement. Ses principaux constituants sont un outil faisant office de MOS, une bibliothèque fournissant l'interface APEX aux applications, et un outil appelé « SIMOUT » permettant l'affichage à la console des sorties des différentes partitions durant l'exécution.

SIMA implémente la partie 1 et la majorité de la partie 2 de la norme ARINC 653. Cette conformité a été démontrée en respectant les spécifications de tests de la section 3 [52]. Plus précisément, SIMA supporte les services suivants de la partie 1:

- Gestion de partition
- Gestion de processus
- Gestion du temps
- Communication Intra-Partition
- Communication Inter-Partition
- Monitorage de la santé

Et les services suivants de la partie 2:

- Ordonnancement de modules multiples
- Système de journaux de bord (de l'anglais « logbooks »)

Les applications peuvent être développées dans les langages C ou ADA.

2.1.4.2.1 MOS

Sous SIMA, le MOS est représenté par un outil du même nom qui s'occupe, comme il se doit, de l'ordonnancement des partitions, de la communication entre celles-ci, ainsi que de la gestion du moniteur de la santé du système.

En réalité, chaque partition est représentée par une application que le MOS lie de façon transparente à un processus de type POSIX. Ainsi, chaque partition dispose de sa propre mémoire de façon statique et prédéfinie. Ce sont ces processus qui sont ordonnancés par le MOS selon le schéma défini dans la configuration du système. L'ordonnancement de modules multiples est supporté depuis la version 1.0 de SIMA.

De la même manière, les tâches créées à l'intérieur d'une partition sont associées à des processus légers (de l'anglais : « threads ») appartenant au processus parent qu'est la partition. Ceux-ci sont ordonnancés selon une politique préemptive basée sur des priorités, où une sélection à la FIFO est utilisée pour les processus légers avec la même priorité. Cette politique d'ordonnancement est conforme à ce qui est spécifié en section 2.3.5.9 de la norme ARINC 653.

Précisons qu'en raison de cette façon de concevoir les partitions et les tâches, la bibliothèque POSIX ne peut pas être utilisée par une application sous SIMA sans nuire à l'ordonnancement du système par le MOS. Il s'agit d'une limitation de SIMA.

L'outil MOS reçoit et distribue également toutes les erreurs internes provenant du POS, c'est-à-dire les contraintes de temps manquées, les erreurs de l'application, et les signaux provenant du noyau Linux.

En ce qui concerne les communications, le transport des messages inter-partitions ou extra-partitions est également géré par le MOS. Cependant, puisque ces opérations contiennent des séquences protégées, dans le but d'assurer le respect des échéances, il est possible et recommandé

de configurer le système pour que les messages ne soient échangés qu'au début et à la fin des fenêtres d'exécution des partitions, dans une tranche de temps prédéfinie.

Pour les communications extra-partition, seuls les ports UDP sont pour l'instant supportés par le simulateur. Il est toutefois possible de faire usage de port TCP/IP à l'intérieur d'une partition, mais leur création et la transmission de leurs messages ne sont pas gérées par le MOS. Ceux-ci peuvent donc causer le non-respect de certaines échéances.

Finalement, les compteurs de temps sont pour l'instant dépendants du système d'exploitation Linux[5]; il n'y a pas de virtualisation de temps. Cela implique qu'une simulation ne peut être mise en pause pour être relancée plus tard, puisque le temps du système n'aura jamais cessé de s'incrémenter. Cette situation amènerait nécessairement des dépassements d'échéances. La feuille de route du projet prévoit cependant implémenter cette fonctionnalité dans le courant de l'année 2012.

2.1.4.2.2 POS et APEX

Les partitions simulées par SIMA ne comportent pas de POS à proprement parler. Une partition est une application qui a été compilée avec une bibliothèque comprenant l'interface APEX portable, adaptée pour Linux, présentée en 2.1.1.2.3, et qui est également utilisée par AIR (i.e. AIRPEX). Ainsi, l'application du développeur peut faire usage de l'entièreté des fonctionnalités et des appels systèmes de l'APEX par-dessus Linux. Ces partitions sous SIMA disposent donc des mêmes services qui seraient offerts via un POS, ou du moins, ceux qui sont couverts par la norme ARINC 653.

[5] T. Schoofs, GMV, Discussion électronique via courriel, 22 février 2012

2.1.4.2.3 Monitorage de la santé

Le monitorage de santé est assuré par l'outil MOS, qui assure la distribution de la gestion des erreurs selon le cas. Le MOS s'occupe de la gestion des erreurs au niveau du module et des partitions, tandis que des traites-erreurs d'applications peuvent gérer les erreurs au niveau « processus ».

Évidemment, bien que la non-propagation des erreurs entre partitions soit visée, SIMA demeure non sécuritaire pour les applications critiques (de l'anglais : « safety-critical ») puisqu'il s'exécute par-dessus un système d'exploitation Linux qui ne l'est pas. Ainsi, il est possible qu'une erreur causée par une application de partition puisse faire échouer le simulateur et le système Linux comme n'importe quelle application de bureau. Cette situation est détaillée dans la section 2.2 du manuel de SIMA [15]. Dans le cas d'une simulation sur un ordinateur de bureau, cela peut impliquer un redémarrage de l'ordinateur.

2.1.4.3 Licence

SIMA est un produit commercial propriétaire de l'entreprise GMV, qui se vend à un coût relativement faible. Une licence coûte environ 1500$ par usager, et n'a pas de date d'expiration. Les mises à jour sont incluses jusqu'à ce qu'une nouvelle version majeure voit le jour. Des frais de support de 500$ par année doivent également être déboursés.[6]

Pour le milieu académique, l'entreprise peut également offrir une licence académique gratuite aux universités.

[6] T. Schoofs, GMV, Discussion électronique via courriel, 22 février 2012

2.1.4.4 Déploiement

SIMA est conçu pour s'exécuter sur un ordinateur de bureau doté d'une architecture x86 (IA-32 ou IA-64) à des fins de simulations. Le déploiement sur une plateforme matérielle n'est pas visé ici.

2.1.4.5 Environnement de modélisation

Préalablement à la simulation d'un système sous SIMA, tant le simulateur que le système doivent être configurés.

Le système à simuler est configuré via un fichier XML conforme à celui spécifié en section 5.0 de la partie 1 de la norme ARINC 653. Ainsi, la configuration d'un système sous SIMA est facilement portable vers tout système également conforme à cet aspect de la norme. La configuration du simulateur est différente. Elle se fait en modifiant une série d'éléments, soit un fichier XML spécifique au simulateur, des fichiers de code script pour lancer les applications qui représenteront les partitions, et les fichiers de compilation.

2.1.4.6 Feuille de route

Il est attendu que SIMA soit intégré à l'environnement de développement IMADE [5] qui devrait couvrir plusieurs étapes du cycle de développement, c'est-à-dire la configuration, la validation, le développement, la simulation et le déploiement (via AIR-II). L'outil CONFIGUIMA de 2.1.1.5 serait utilisé pour la configuration.

Parmi les autres innovations attendues, on retrouve :

- La virtualisation du temps
- Des outils de profilage
- Amélioration des capacités de déverminage
- Une interface d'entrée/sortie pour supporter notamment TCP/IP et ARINC 429.

- Le support du supplément 3 de la partie 1 de ARINC 653
- Le support du supplément 1 de la partie 2 de ARINC 653
- Le support de la partie 4 de ARINC 653

2.2 Flots de modélisation

Dans plusieurs cas, des environnements d'exécution ARINC 653 ont été intégrés à des flots de modélisation afin de bénéficier des avantages liés aux concepts de MBE qui ont été mentionnés en 1.1.3. Cette section présentera tout d'abord un survol de quelques langages de modélisation existants, et ensuite, détaillera quelques projets de flot de modélisation qui ont fait l'objet d'articles dans la littérature.

2.2.1 Langages

L'exploration des différents langages de modélisation n'étant pas l'objet de ce mémoire, les plus importants ne seront ici que sommairement présentés. Seul le langage AADL fera l'objet d'une présentation plus approfondie, considérant qu'il est fréquemment référencé dans ce mémoire.

2.2.1.1 Architecture Analysis and Design Language (AADL)

Le langage AADL [14, 53-55] a été développé en 2004 par SAE International, une communauté d'ingénieurs et de techniciens experts dans les domaines de l'aérospatiale, de l'automobile et de l'industrie des véhicules commerciaux, qui sont impliqués dans l'élaboration et la révision de normes.

AADL permet de modéliser l'architecture d'un système en offrant la possibilité de définir les éléments matériels et logiciels qui le composent. Ceux-ci sont modélisés dans une perspective « boîte noire »; les détails de leur implémentation ne peuvent être explicités par ce langage. C'est plutôt la hiérarchie et la correspondance entre ces composants, leurs interfaces, leurs interconnexions, et leurs propriétés générales qui sont modélisées. Il est également possible de

schématiser les échanges de données, la synchronisation de processus, ainsi que le partage et l'accès à des ressources. C'est un langage de modélisation utilisé avant tout pour concevoir et analyser l'architecture logicielle et matérielle de systèmes temps-réel critiques.

Néanmoins, même s'il est impossible de modéliser l'implémentation d'un composant avec ce langage, des mécanismes sont fournis pour lier ceux-ci à des implémentations extérieures (ex. : programme en langage C, modèle Simulink, etc.).

Plusieurs outils existent pour procéder à des analyses préliminaires d'un système à partir de modèles AADL, ce qui rend ce langage très utile. Mentionnons le langage d'analyse du respect des requis (en anglais : « Requirement Enforcement Analysis Language » ou REAL) [56], qui permet de valider l'intégrité du modèle et le respect de règles définies par le développeur, ainsi que le projet CHEDDAR [57] permettant une analyse d'ordonnancement, tout comme le projet de Sokolsky et coll. [58].

Depuis 2011, le AADL a été étendu pour supporter la modélisation de systèmes ARINC 653. Ce langage est d'ailleurs utilisé pour ce type de modélisation dans les projets MASIW et TASTE qui seront présentés en section 2.2.2, et dans le projet POK, présenté en 2.1.3.

Dans un contexte de génération de code ou de configuration, ce langage permet donc l'intégration automatique de l'implémentation logicielle d'un programmeur à une plateforme d'exécution conforme à ARINC 653.

En effet, à partir du modèle AADL d'un système, le générateur de code OCARINA, présenté en 2.2.1.1.1, peut générer autant le code C effectuant les appels aux fonctionnalités APEX pour l'initialisation du système et la création de composants (ports, évènements, tâches, etc.), que les fichiers de configuration XML du système, dont les systèmes d'exploitation ont besoin.

D'un point de vue industriel[7], cette procédure devient très pertinente lorsque des applications qui n'ont initialement pas été développées conformément à la norme ARINC 653 doivent migrer vers cet environnement. Il n'est ainsi plus essentiel pour le développeur de connaître tous les détails de l'interface de programmation APEX, mais plutôt de maîtriser les concepts pour modéliser adéquatement le système.

La figure 2-1 présente les composants ARINC 653 qui peuvent être modélisés sous AADL [55]. Néanmoins, pour éviter d'alourdir le graphique, seuls les éléments qui influencent la génération de la configuration ou du code C sont présentés; les éléments modélisables qui touchent davantage les éléments matériels (ex. : temps de lecture ou d'écriture sur la mémoire) et qui visent à soumettre le système à des analyses préliminaires via le modèle sont ici ignorés. De plus, afin de présenter clairement tous les éléments ARINC 653 qui peuvent être modélisés, ceux-ci sont présentés comme des composants AADL propres, indépendamment de la façon dont ils sont implémentés dans le langage. Ce schéma fait donc abstraction du fait que certains de ces composants (ex. : blackboard, buffer, event) n'existent pas comme tels dans le langage, mais peuvent être représentés tout de même à partir d'autres composants existants sous le AADL.

On remarque que l'implémentation de fonctionnalités développées en C peut être intégrée aux composants appelés « sous-programme » (de l'anglais : « subprogram ») qui sont ensuite associés à une ou plusieurs tâches, réparties sur une ou plusieurs partitions. Des implémentations peuvent également être liées à des éléments faisant office de périphérique (de l'anglais « device ») ou pilote matériel afin de supporter l'intégration d'interfaces de communications particulières avec des composants ou systèmes extérieurs.

Notons que le comportement dynamique du système et de ses partitions, c'est-à-dire ses modes d'opérations et ses transitions, peut être explicité par le modèle, et être utilisé pour décrire une gestion appropriée des erreurs selon le cas.

---

[7] R. Pontbriand, Esterline – CMC Électronique, Discussion dans les locaux de CMC Électronique, 13 juin 2012

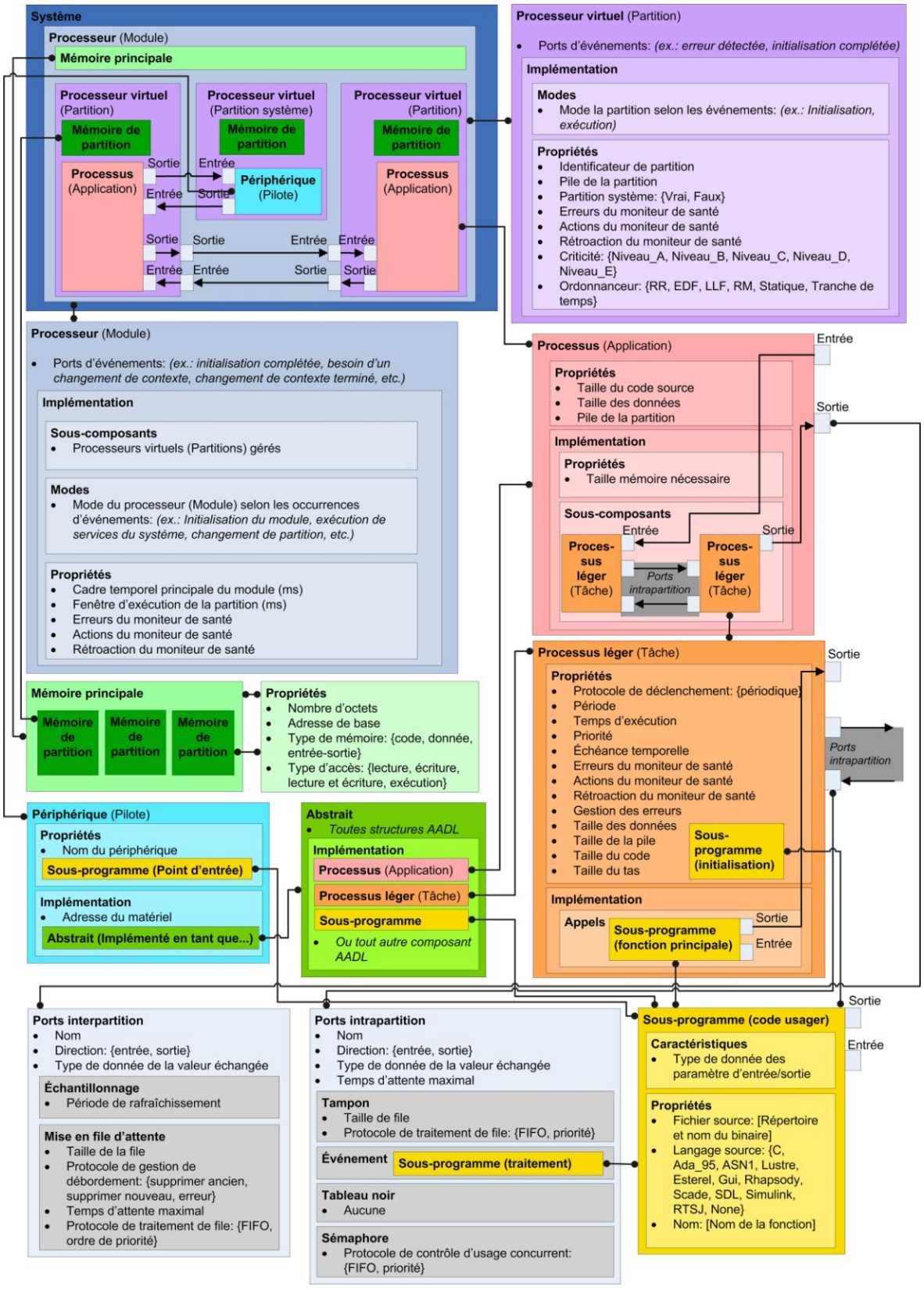

Figure 2-1: Survol de la modélisation de système ARINC 653 via AADL

Le langage a été conçu pour être extensible afin de permettre les analyses sur des architectures que le langage ne supporte pas complètement. Les extensions peuvent consister en de nouvelles propriétés, ou encore, en des notations spécifiques pour l'analyse qui peuvent être associées à des composants. Les utilisateurs ou les développeurs de produits peuvent définir ces extensions, et les proposer ensuite en tant que document d'annexe afin qu'elles puissent éventuellement être incluses dans la norme AADL [53].

De façon générale, AADL peut être utilisé avec plusieurs outils dans le but d'automatiser en partie les étapes de la modélisation, de l'analyse, de l'implémentation, de l'intégration, de la vérification et de la certification.

Un modèle AADL peut être défini autant de façon textuelle dans un éditeur de texte, que dans un environnement avec une interface graphique. Parmi les environnements graphiques disponibles, on retrouve le projet TOPCASED (pour « Toolkit in OPen-source for Critical Applications & SystEms »)[59], un ensemble d'outils et de greffons intégrés à Eclipse, qui supporte l'édition graphique d'architecture de système basée sur AADL à l'aide de l'extension ADELE[60].

2.2.1.1.1 OCARINA

Le projet OCARINA [17, 61] fut initié par l'École nationale supérieure des télécommunications de Paris (Télécom ParisTech). Le projet entrant dans la catégorie des logiciels libres, son développement est dorénavant maintenu par une communauté plus large.

Un peu à l'instar d'un compilateur, OCARINA est composé d'une partie frontale (en anglais : « frontend ») et d'une partie arrière (en anglais : « backend »). Il effectue les analyses lexicales, syntaxiques et sémantiques sur les modèles AADL pour pouvoir générer le code dans le langage ciblé. En plus du langage C, le code peut également être généré dans le langage de programmation ADA.

OCARINA peut être utilisé en combinaison avec d'autres outils pour lui rajouter des fonctionnalités supplémentaires. Parmi ceux-ci, on retrouve :

- CHEDDAR, pour réaliser une analyse d'ordonnancement
- Les réseaux de Pétri, pour faire de la vérification de modèle (en anglais : « model checking »)
- Bound-T de Tidorum Ltd., pour calculer les pires temps d'exécution
- REAL, pour évaluer l'intégrité et le respect des requis du système, définis sous forme de règles, en se basant sur les propriétés et les métriques du modèle

OCARINA fut implémenté dans le langage ADA, notamment en raison du fait que beaucoup de théories de compilation existaient autour de ce langage, et que ce dernier est, par exemple, davantage maintenable et léger que Java.

En ce qui concerne la génération de code conforme à ARINC 653, préalablement à ce projet, OCARINA ciblait uniquement les environnements POK, ainsi que l'hyperviseur XtratuM utilisé avec RTEMS.

La licence de ce générateur est la même que pour l'édition professionnelle du compilateur gratuit pour ADA, appelée « GNAT ». Il s'agit d'une version modifiée du GPL (« GNAT Modified General Public License »).

OCARINA peut se compiler et s'exécuter tant dans les environnements Linux, Mac OS X, Solaris, FreeBSD que Windows. Il devrait en théorie fonctionner sur toute cible dotée d'un compilateur GNAT.

2.2.1.2 SysML

SysML [26] est un profil de UML 2.0 qui vise la modélisation dans le cadre de l'ingénierie de

systèmes, ce qui implique également la modélisation d'éléments non-logiciels (ex. : matériel, information, processus, personnel, etc.). Pour ce faire, comparativement aux autres langages, SysML offre deux types de diagrammes additionnels, soit un diagramme de requis, et un diagramme de paramètres. Il modifie aussi la sémantique de la définition des blocs UML 2.0 et des diagrammes de blocs internes.

SysML offre ainsi une bonne intégration avec le monde physique, peut modéliser le temps de façon continue, et supporte la traçabilité des requis.

2.2.1.3 Modeling and Analysis of Real-Time and Embedded systems (MARTE)

Tout comme SysML, MARTE [26] est également un profil de UML 2.0 visant à modéliser à la fois les aspects matériels et logiciels des systèmes embarqués temps-réel. Il permet également de procéder à des analyses quantitatives sur les modèles. Ses objectifs et les concepts qu'ils couvrent sont ainsi très près de ceux de AADL, ce qui en fait en quelque sorte l'équivalent UML de AADL. MARTE peut aussi interagir avec AADL via une annexe définie dans sa norme.

2.2.2 Projets

Des projets ont été réalisés dans les dernières années autour du MBE pour systèmes ARINC 653. Cette section en présente quelques-uns.

2.2.2.1 MASIW

Ce projet [26] fut lancé conjointement par l'institut en programmation de systèmes de l'académie des sciences de Russie (ISPRAS) et par l'institut d'état de recherche en systèmes d'aviation (GosNIIAS), un centre scientifique d'état de la fédération russe. Il avait pour objectif de pousser la recherche visant à réduire le coût et le temps de développement des systèmes IMA en automatisant les étapes de conception et les processus d'intégration des environnements utilisés

par la GosNIIAS. Pour ce faire, l'objectif fut de combler les lacunes de savoir-faire autour de la construction de chaînes d'outils permettant ce type d'automatisation pour des projets spécifiques.

La chaîne d'outils proposée par ISPRAS et GosNIIAS s'appelle la « Modular Avionics System Integrator Workplace (MASIW) ». Le projet est toujours en cours, et le langage de description architecturale utilisé est **AADL**.

MASIW consiste en 4 modules logiques :

1. **Environnement de travail** : Module de base de l'outil qui gère les projets et présente leur contenu dans une vue arborescente.
2. **AADL** : Module de traitement AADL qui consiste en un éditeur de texte pour AADL textuel, ainsi qu'un greffon transformant le AADL textuel en une modélisation interne.
3. **Base de données (DB) de documents de contrôle des interfaces (ICD)**: Module responsable de l'interaction avec la ICD DB. Il extrait les objets demandés de la DB, les transforme en modèle AADL, et peut entreposer des changements dans la DB.
4. **Ordonnancement** : Module qui construit l'ordonnancement des partitions ARINC 653 pour un processeur donné. Il consiste en un greffon créant les ordonnancements à partir d'un des algorithmes enregistrés, ainsi qu'un greffon implémentant la visualisation graphique.

Les fichiers de configuration d'ordonnancement peuvent être générés selon le format défini dans la norme ARINC 653, ou selon le format de VxWorks 653 de WindRiver.

MASIW inclut également un générateur de configuration pour réseaux AFDX, ainsi qu'un simulateur de ceux-ci. **Ce simulateur fournit des analyses de performances préliminaires relativement aux configurations AFDX avant leur implémentation.** On y note :

- les latences
- le moment et la cause des délais
- la distribution de la taille des queues
- la probabilité de pertes de messages

Cette chaîne d'outils est actuellement utilisée pour la conception et l'intégration des tests déployés à GosNIIAS dans le programme russe de IMA.

L'investigation de la possibilité d'intégrer à MASIW le générateur OCARINA ou d'autres outils basés sur le AADL (CHEDDAR, REAL, AADL2FIACRE, etc) fait partie des travaux qui seront abordés dans le futur par l'équipe du projet.

2.2.2.2 TASTE

Le projet TASTE (« The Assert Set of Tools for Engineering ») vise à faciliter le développement de systèmes en automatisant celui-ci le plus possible à travers un processus incluant des étapes de modélisation, d'analyse de faisabilité et de génération de code.

Un des éléments importants visé par le projet est d'être capable de faire le pont entre différents modules, et différentes fonctionnalités, implémentés ou modélisés dans des langages et des environnements différents [27].

Ainsi, TASTE [62] supporte deux langages de modélisation complémentaires :

1) AADL pour définir l'architecture matérielle et logicielle du système.
2) ASN.1 qui est la technologie principalement responsable de la communication entre les modules réalisés sous différents environnements, notamment ceux modélisant le niveau applicatif.

Ces deux langages de modélisation peuvent donc s'interfacer automatiquement avec les environnements suivants, qui modélisent l'implémentation de fonctionnalités et qui peuvent donc générer le code des applications :

- SCADE/KCG : Générateur de code qualifiable DO-178B qui produit du code C
- Simulink Coder [63] : Générateur de code C/C++ à partir de diagrammes Simulink, de diagrammes de Stateflow [64], et de fonctions Matlab

- SDL/SDL-RT sous Real-Time Developper Studio (RTDS) de PragmaDev [65] : Vérificateur de propriétés et générateur de code C [66]

Les implémentations peuvent également être rédigées directement en code C/C++, Ada, SystemC ou VHDL. Des bibliothèques externes de type « boîte noire » sont aussi supportées.

Cette intégration est possible principalement en raison du langage ASN.1, qui permet de définir les messages et les types de façon suffisamment abstraite pour être compatible avec tous ces modules éclectiques. Des outils existent ensuite pour transformer ces éléments abstraits en éléments reconnaissables par les différents environnements de modélisation ou d'implémentation qui sont ciblés.

La chaîne d'outils de TASTE est pour l'instant composée des outils suivant, qui ont chacun une licence d'utilisation propre :

- Les outils d'édition graphique de TASTE offerts par la compagnie Ellidiss, qui ont des licences commerciales.
- Les outils ASN1 et l'orchestrateur de Semantix, qui n'ont pas de coût de licence dans le cas d'un usage non commercial.
- OCARINA, le générateur de code présenté en 2.2.1.1.1, qui est un logiciel libre suivant la licence GPL.
- Les analyseurs d'ordonnancement CHEDDAR et MAST, qui sont des logiciels libres.
- Finalement, le GUI de TASTE, qui est la propriété de l'Agence Spatiale Européenne (en anglais : « European Space Agency » ou ESA) et qui est gratuit uniquement si les utilisateurs résident dans un pays qui finance les travaux de l'ESA.

TASTE peut générer des applications pour les architectures suivantes :

1. x86 avec les OS **Linux**, Mac OS X, FreeBSD, et **RTEMS**.
2. ARM avec RTEMS et Linux
3. SPARC (LEON) avec RTEMS et OpenRavenscar. Pour LEON/RTEMS, TASTE peut s'interfacer avec la carte RASTA qui fournit une interface pour les bus sériel, spacewire et 1553.

Suite aux travaux de Delange, Honvault et Windsor [25], et dans l'optique d'avoir un environnement d'exécution ARINC 653, le générateur de code OCARINA supporte également l'hyperviseur XtratuM comme cible, avec RTEMS comme POS. Celui-ci dispose donc dorénavant d'un environnement de modélisation AADL.

Cette réalisation permet entre autre à TASTE de faciliter le choix architectural entre les modèles fédéré et IMA pour un même système ou sous-système, puisqu'il lui est possible de générer les deux architectures.

D'autres travaux, cette fois de Delange, Hugues et Dissaux [13], ont proposés une approche pour valider l'implémentation d'un système en comparant des résultats de simulations avec des résultats d'exécution. Pour ce faire, les auteurs ont adapté TASTE en modifiant, entre autres, les traces obtenues à l'exécution d'un système pour que celles-ci présentent :

1) Le moment d'activation de chaque tâche
2) L'utilisation des ports de communication (taille des queues, moments des écritures et lectures, etc.)
3) D'autres métriques pertinentes observables lors de l'exécution

L'intrusivité de ce monitorage a été mitigée en stockant les métriques dans des tampons mémoires qui sont effacés lorsque l'exécution est terminée.

Ainsi, cette méthode permet de vérifier que les traces d'exécution sont conformes de façon fonctionnelle aux traces de simulation, mais également du côté des délais et du temps d'exécution prévus sur la plateforme cible. Le langage de modélisation AADL fut le seul utilisé dans ce projet.

2.2.2.3 Tables de configuration ARINC 653 via modèles

Le projet [23] proposé par Horváth et Varró de l'Université Polytechnique et Économique de Budapest, et par Schoofs de l'entreprise GMV, a implémenté une chaîne d'outils qui génère des

tables de configuration ARINC 653 à partir de modèles d'architecture haut-niveau. Celle-ci cible les systèmes VxWorks 653, ainsi que le simulateur SIMA.

Cette approche vise à combler les lacunes qui rendent pénible la vérification des tables de configuration, ARINC 653. Par exemple :

1) Valider les contraintes de conception liées aux configurations durant leur création (en anglais : « on-the-fly »)
2) Enregistrer de façon explicite les décisions de conception critiques prises par l'architecte du système
3) Fournir un système de traçage entre la définition « haut-niveau » des requis et les tables de configuration rédigées

Pour la génération de tables de configuration, de code source, ou d'autres artéfacts logiciels à partir de modèles, cette approche transforme d'abord le modèle indépendant de la plateforme (PIM) en un modèle correspondant qui est cette fois spécifique à la plateforme (PSM) à l'aide d'une transformation de modèle (MT).

Ce processus de MT prend en entrée le PIM, ainsi que d'autres modèles décrivant la plateforme (ex. : CPU, latences, bande passante, etc.) et ses interfaces. La transformation MT procède ensuite à une extraction des contraintes de conception (ex. : performance, dépendance, sécurité, etc.), et identifie les décisions de conception affectant ces contraintes. Cette étape peut être faite autant de façon automatique que dirigée par l'usager puisqu'elle implique des décisions critiques qui ne peuvent pas toutes être automatisées. Elle termine en allouant les ressources en procédant à l'ordonnancement, à l'optimisation, et en évaluant la qualité de la transformation.

Les éléments de traces obtenus lors de la création du PSM sont entreposés dans un document pour assurer une traçabilité.

Cette chaîne d'outils a été évaluée via une étude de cas utilisant une véritable application avionique, soit un système d'air conditionné, dans le cadre du projet DIANA [52].

À terme, cette chaîne d'outils devrait être intégrée dans l'environnement d'Eclipse, et visera l'obtention d'informations de traçabilité de bout en bout pour supporter la certification des activités de validation et vérification. De plus, les activités de vérification et de validation (V&V) devraient être intégrées au processus de développement pour fournir le plus tôt possible de la rétroaction sur les requis, les spécifications, la conception et l'implémentation.

2.3 Limites observées

Tel que présenté en section 2.1, il existe plusieurs alternatives à faible coût aux systèmes d'exploitations commerciaux supportant ARINC 653. Néanmoins, aucun de ces environnements d'exécution n'est idéal.

Tout d'abord, beaucoup demeurent dépendants de la plateforme matérielle cible, ce qui peut être limitatif, considérant que ceux-ci sont utilisés à un moment du flot de développement où l'exploration architecturale est possiblement toujours en cours. C'est le cas de XtratuM, AIR et POK. Des simulateurs de processeurs ou de plateformes comme Simics ou QEMU peuvent être utilisés, tel qu'expérimenté par Carmel-Veilleux [21], mais tous ne sont pas offerts sous une licence libre ou peu coûteuse.

Ensuite, plusieurs de ces environnements ne sont pas entièrement conformes à la norme ARINC 653. Bien que son extension LithOS semble corriger ce problème, XtratuM ne visait initialement pas la conformité avec le standard. De son côté, POK ne supporte qu'en partie l'interface APEX et ne supporte qu'une quantité très restreinte des fonctionnalités de la bibliothèque C. Seules les fonctionnalités mathématiques sont disponibles en grand nombre. Bien que cela puisse être intéressant pour le développement d'applications de calculs, cette réalité restreint beaucoup le développement d'applications. De plus, POK demeurant un prototype, plusieurs limitations fondamentales ont été observées, notamment la limitation à 1024 octets de l'espace mémoire qui peut être partagée entre les ports de communication d'une même partition, et l'utilisation d'un outil non validé pour la génération de la configuration du système à partir de fichiers XML. Le simulateur SIMA, par contre, a démontré sa conformité à la norme.

Un autre élément problématique avec la plupart de ces solutions est inhérent à beaucoup de logiciels libres, soit le manque de support et de documentations. Plusieurs des limitations découvertes sur POK l'ont été suite à des expérimentations, et ne figuraient pas dans la documentation officielle. Un manque de documentations concernant XtratuM a également été souligné par Carmel-Veilleux [21]. En date du 15 juin 2012, les composants du projet AIR-II n'étaient pas disponibles pour téléchargement, et l'hyperviseur XtratuM n'était plus disponible en téléchargement depuis plus d'un mois sur le site officiel du projet. Bien que la liste de diffusion des développeurs (en anglais : « mailing list ») de POK soit très active, il n'en existe pas pour les autres alternatives, et il est très difficile de rejoindre les contacts officiels. Cette situation peut être très dérangeante dans un contexte de recherche ou dans un contexte industriel.

Finalement, considérant les avancées nombreuses autour du MBE pour les systèmes IMA, notamment dans le cadre des projets TASTE et MASIW, le potentiel de tels flots de conception est de plus en plus évident, tant du côté des simulations préliminaires que de la génération automatique de configuration ou de code. Conformément à nos objectifs, il va sans dire qu'un environnement d'exécution ARINC 653 idéal devra être intégré à un tel flot. Hors, ce n’est pas le cas du simulateur SIMA.

Cette revue de littérature confirme donc la problématique identifiée dans ce mémoire. Il y a actuellement absence d'un environnement d'exécution ARINC 653 qui serait à la fois peu coûteux, indépendant de la plateforme matérielle, conforme à ARINC 653, utile pour des simulations en début de développement, et intégré à un environnement de MBE, tout en offrant une documentation et un support satisfaisant.

CHAPITRE 3 FLOT DE CONCEPTION PROPOSÉ

Suite aux conclusions de la revue de littérature, et en fonction de la problématique soulevée, le présent chapitre présentera une proposition de flot de conception basé sur une approche de MBE qui comporte un environnement d'exécution ARINC 653 à faible coût. Les prochaines sections détailleront les technologies retenues, ainsi que les raisons qui ont motivé ces choix.

3.1 Description du flot de conception proposé

La figure 3-1 présente la composition du flot de conception proposé.

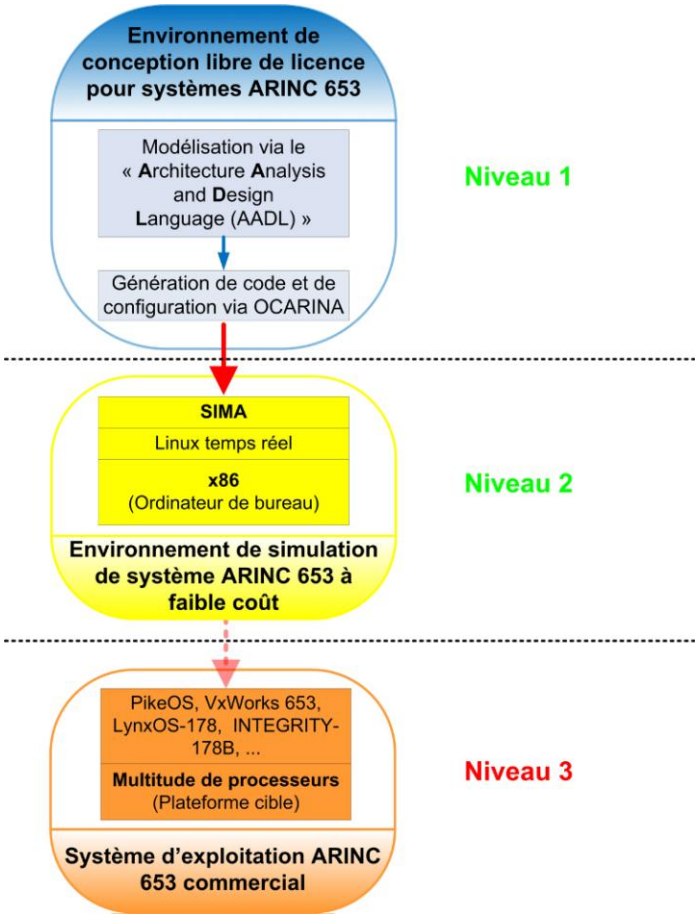

Figure 3-1: Flot de conception proposé dans le cadre de ce mémoire

Tel que présenté, l'environnement de modélisation retenue est le langage AADL, où une extension au générateur de code et de configuration Ocarina permettra de cibler le simulateur SIMA, qui sera ici utilisé comme environnement d'exécution pour fins de simulation. Bien que seuls les niveaux 1 et 2 du diagramme seront réalisés dans le cadre de ce mémoire, l'objectif ultime demeure de pouvoir déployer le système développé vers un environnement commercial, soit vers le niveau 3 du diagramme, à la fin du flot.

3.1.1 Environnement de simulation

Tel que mentionné précédemment, les contraintes principales de l'environnement d'exécution intégré dans ce projet de recherche sont d'être abordable, conforme à ARINC 653, utile en début de développement, et d'utiliser une approche de MBE. Pour satisfaire toutes ces contraintes, nous avons donc décidé de sélectionner le simulateur SIMA, qui s'est démarqué pour plusieurs raisons.

Tout d'abord, il ne contraint pas les concepteurs à identifier et acquérir une plateforme matérielle spécifique pour exécuter les systèmes. Sans être complètement indépendant de la plateforme matérielle, puisqu'il n'existe pour l'instant que sous les architectures IA-32 et IA-64, le fait qu'il puisse s'exécuter sur la plupart des ordinateurs de bureau remplit adéquatement notre objectif d'être utile en début de développement. C'est-à-dire à un moment où les concepteurs pourraient ne pas encore avoir pris de décision sur l'architecture ciblée.

Ensuite, il fournit aux développeurs d'applications un environnement de développement très flexible, où l'on retrouve une interface de programmation APEX complète et conforme à la norme ARINC 653, ainsi que l'accès à toutes les fonctionnalités du langage C ou C++ disponibles, à l'exception de POSIX. Cette latitude évite d'imposer des limitations d'API aux développeurs qu'ils ne retrouveraient pas avec un OS commercial.

Également, bien que son code source ne soit pas ouvert, et que l'outil ne soit pas sans frais, il possède un atout intrinsèque aux solutions commerciales, soit l'offre d'un soutien technique et une documentation plus que satisfaisante, ce qui n'était pas le cas de la plupart des solutions alternatives. Dans un contexte industriel, et même dans le milieu académique, un tel soutien permet d'économiser beaucoup de temps, et donc d'argent. De plus, le faible coût de la licence respecte toujours notre objectif initial.

Évidemment, ce simulateur n'est pas parfait, et comporte des limitations. Tout d'abord, bien que l'utilisation d'une version temps réel de Linux (e.g. avec contraintes dures) accélère grandement l'exécution par rapport à une distribution standard (e.g. avec contraintes douces), il reste que les latences sous SIMA sont beaucoup plus élevées que sous un système embarqué. Lors des observations expérimentales de l'entreprise, la latence moyenne a varié de 1 à 62 µs selon le cas. Pour cette raison, l'entreprise garantit les échéances à partir de 100 µs en montant. À titre indicatif, les latences observées pour une version qui n'est pas « temps réel » de Linux variaient entre 1 et 10 ms.

Parmi les autres limitations importantes, rappelons que le MOS ne supporte pour l'instant que les ports de type UPD, que la configuration mémoire n'est pas considérée, que le système n'est pas à l'abri d'une panne, et que le transport des messages ne s'effectue qu'à des moments prédéterminés lors d'une fenêtre d'exécution.

Finalement, une problématique qui sera abordée dans nos travaux expérimentaux du chapitre 4 touchera également le manque de savoir-faire autour de cet outil dans le milieu académique québécois.

Néanmoins, dans le contexte où l'environnement d'exécution que nous recherchons vise avant tout la simulation en début de développement, nous avons jugé toutes ces limitations acceptables.

3.1.2 Environnement de modélisation

Pour l'instant, il n'existe aucun environnement permettant de configurer SIMA automatiquement à partir d'une interface usager graphique, ou encore, à partir d'un langage de modélisation. L'interface graphique appelée CONFIGUIMA, qui fut présentée en 2.1.1.5, est actuellement en développement par GMV et devrait compter SIMA parmi ses cibles. Par contre, cela n'offrira pas les mêmes possibilités qu'amènerait, par exemple, l'intégration de SIMA à un environnement de modélisation (ex. : AADL), notamment quant à la possibilité de faire des analyses de faisabilité sur le modèle ou de générer une partie du code source d'initialisation.

Ainsi, il était nécessaire de pallier à ce problème pour atteindre nos objectifs initiaux. Nous avons donc décidé d'étendre un générateur de code existant, prenant en entrée une modélisation du système, afin qu'il puisse cibler le simulateur SIMA.

Notre attention c'est ainsi porté sur OCARINA, le générateur au code source ouvert qui a été présenté en 2.2.1.1.1. Le choix du générateur Ocarina s'est imposé tout naturellement à notre équipe. La raison principale étant évidemment que celui-ci est ouvert, et qu'il nous sera possible de l'adapter pour nos besoins, contrairement à d'autres générateurs commerciaux. De plus, plusieurs projets s'activent actuellement autour de cet outil, notamment TASTE qui implique l'ESA. Cet intérêt scientifique devrait dynamiser la communauté des développeurs d'Ocarina, et par le fait même, nous fournir un support technique via nos pairs durant le développement. Finalement, le langage AADL utilisé par cet outil convient parfaitement à nos objectifs de modélisation de système, d'autant plus qu'il en existe un profil ARINC 653. Ocarina cible déjà les environnements d'exécution ARINC 653 POK et XtratuM, ce qui pourrait faciliter notre tâche. La figure 3-2 présente autrement le changement qui sera apporté aux flots actuels de SIMA et Ocarina.

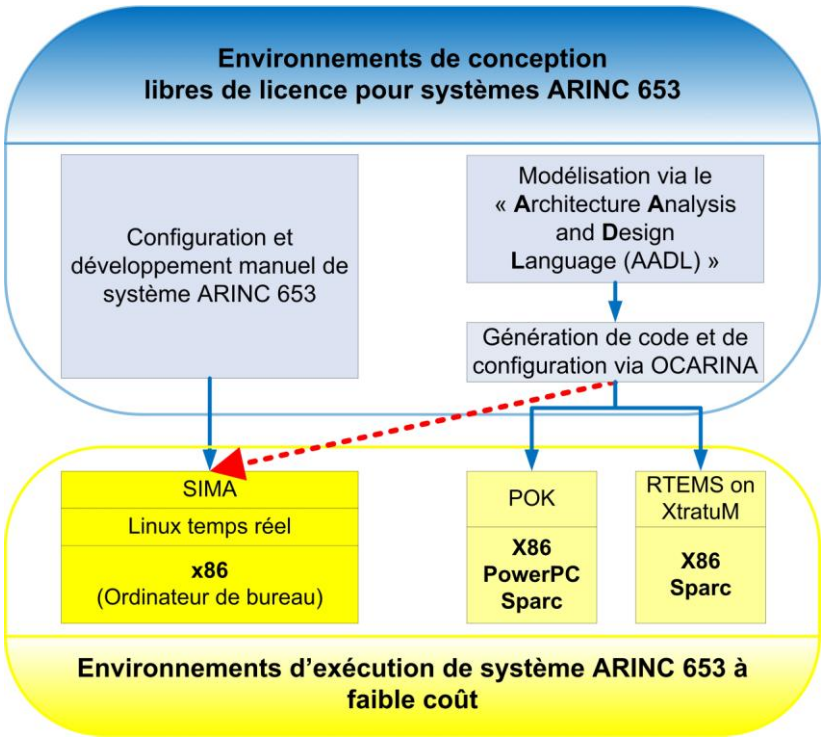

Figure 3-2: Rajout d'une nouvelle cible au générateur Ocarina

3.1.3 Environnement de déploiement

Considérant que la norme ARINC 653 vise la portabilité des configurations et des systèmes développés, nous considérons tous les systèmes d'exploitation commerciaux disponibles comme étant des cibles envisageables pour l'étape de déploiement de notre flot. Toutefois, cette étape de transition du niveau 2 vers le niveau 3 de la figure 3-1 ne sera ni automatisée, ni réalisée manuellement dans le cadre de ce mémoire. L'implémentation complète du flot proposé est un objectif du projet AVIO 509, et sera complétée prochainement par un autre membre de l'équipe. Dans ce qui suit, nous nous concentrerons donc sur les niveaux 1 et 2.

3.2 Réalisation du flot de conception

Dans l'objectif d'intégrer le simulateur SIMA dans le générateur Ocarina, nous avons eu l'opportunité d'avoir comme point de départ quelques cibles déjà existantes visant la conformité à la norme ARINC 653. Puisqu'un des objectifs de cette norme est la portabilité des applications,

la configuration et le code d'initialisation de celles-ci sont censés ne pas changer (ou très peu) d'un environnement d'exécution à l'autre. C'est particulièrement le cas entre les environnements SIMA et POK. Ainsi, dans le cadre de cette expérimentation, nous avons sélectionné la cible POK comme point d'arrimage pour parvenir à étendre Ocarina vers SIMA.

La méthodologie mise de l'avant pour ce projet fut donc la suivante :

1) *Évaluation du travail à réaliser :* Identification des éléments de configuration unique à SIMA qui le distinguent des autres cibles comme POK, et qui devront être intégrés dans le générateur.
2) *Preuve de concept* : Portage de quelques systèmes générés pour une autre cible (i.e. POK) vers SIMA afin de valider les observations précédentes.
3) *Implémentation* : Modification du générateur Ocarina.
4) *Cas d'utilisation :* Test du générateur étendu avec l'application de MCDU présentée en 4.1.2.

Les étapes 1 à 3 seront présentées en détails dans les sections suivantes, tandis que l'étape 4 sera plutôt présentée au prochain chapitre.

3.2.1 Évaluation du travail à réaliser

3.2.1.1 Fichiers de configuration à générer

3.2.1.1.1 Configuration des concepts ARINC 653

Un système sous SIMA se configure via deux fichiers XML. Un des fichiers XML caractérise les éléments et les concepts propres à ARINC 653 conformément à la norme[6]. La plupart des éléments de ce fichier sont descriptibles en AADL, et peuvent être générés. Par exemple, la cible POK d'Ocarina en génère déjà une grande partie. Certaines particularités de SIMA devront cependant être prises en compte. Un exemple d'un tel fichier XML est présenté en annexe 2 pour référence.

La plus importante particularité concerne la gestion des erreurs que SIMA définit dans ce fichier XML. Considérant que chaque environnement d'exécution ARINC 653 possède habituellement ses propres identificateurs d'erreurs, la gestion des erreurs sous ce simulateur est très différente, par exemple, de la cible POK.

3.2.1.1.1.1 Gestion des erreurs

Le tableau 3.1, extrait de la documentation de SIMA [15], présente la classification des erreurs telle que gérée par SIMA selon les différents états du système. Ainsi, un même type d'erreur (par exemple, une erreur de segmentation) peut être traitée différemment selon l'état du système. Ces six états sont : l'initialisation du MOS, l'exécution du MOS, l'initialisation de la partition, l'exécution de l'application d'une partition, l'exécution du gestionnaire d'erreur de la partition, et finalement, l'exécution du gestionnaire d'erreur du MOS.

Chaque erreur peut être gérée au niveau du module, de la partition, ou du processus. Évidemment, la gestion au niveau du processus ne peut se faire que dans l'état 5 de SIMA, tandis que les états 1 et 2 doivent gérer leurs erreurs au niveau du module.

Les sections 2.4.1, 2.4.2 et 2.4.3 de la partie 1 de la norme ARINC 653 présentent une normalisation des types d'erreurs selon le niveau (module, partition, processus), ainsi que les actions possibles selon le cas. Celles-ci sont présentées dans le tableau 3.2. Le vocabulaire utilisé par le langage AADL pour représenter ces réactions est également inclut dans le tableau.

Tableau 3.1: Gestion des erreurs sous SIMA selon l'état du système

État du système		**Erreurs possibles**		**État du système**		**Erreurs possibles**	
ID	Nom	ID	Nom	ID	Nom	ID	Nom
1	Démarrage du MOS	2	Configuration du module	5	Exécution de l'application d'une partition	5	Erreur de segmentation
		4	Initialisation du module			6	Erreur temporelle
		5	Erreur de segmentation			7	Instruction illégale
		7	Instruction illégale			8	Erreur numérique
		8	Erreur numérique			9	Débordement de pile
		9	Débordement de pile			10	Erreur de l'application
		11	Mauvais code d'opération pour une instruction			11	Mauvais code d'opération pour une instruction
		12	Panne de courant			12	Panne de courant
2	Exécution du MOS	5	Erreur de segmentation	6	Exécution du gestionnaire d'erreur de la partition	10	Erreur de l'application
		6	Erreur temporelle	9	Exécution du gestionnaire d'erreur du module	5	Erreur de segmentation
		7	Instruction illégale			6	Erreur temporelle
		8	Erreur numérique			7	Instruction illégale
		9	Débordement de pile			8	Erreur numérique
		11	Mauvais code d'opération pour une instruction			9	Débordement de pile
		12	Panne de courant			11	Mauvais code d'opération pour une instruction
4	Éxécution du code propre au POS	3	Configuration de la partition			12	Panne de courant
		5	Erreur de segmentation				
		6	Erreur temporelle				
		7	Instruction illégale				
		8	Erreur numérique				
		9	Débordement de pile				
		11	Mauvais code d'opération pour une instruction				
		12	Panne de courant				

Tableau 3.2: Actions possibles selon niveau du gestionnaire d'erreur

Réaction	Actions définies par AADL	Niveau
Ignorer (IGNORE)	*Ignore*	Module et partition
Éteindre le module (SHUTDOWN)	*Module_Stop*	Module
Redémarrer le module (RESET)	*Module_Restart*	Module
Arrêter la partition (IDLE)	*Partition_Stop*	Partition
Démarrage à chaud de la partition (WARM_START)	*Partition_Restart*	Partition
Démarrage à froid de la partition (COLD_START)	*Partition_Restart*	Partition
Ignorer et consigner l'erreur	*Nothing*	Processus
Ignorer l'erreur un certain nombre de fois avant de réagir	*Nothing*	Processus
Arrêter le processus fautif, et le réinitialiser du début	*Process_Restart*	Processus
Arrêter le processus fautif, et démarrer un autre processus	*Process_Stop_And_Start_Another*	Processus
Arrêter le processus fautif, et présumer que la partition détectera la situation et réagira	*Process_Stop*	Processus
Redémarrer la partition à chaud ou à froid	*Partition_Restart*	Processus
Arrêter la partition (IDLE)	*Partition_Stop*	Processus

Pour l'implémentation du générateur de code, il faudra donc faire un parallèle entre les types d’erreurs normalisés sous ARINC 653, les types définissables sous AADL, et les types définis sous SIMA. Le tableau 3.3 présente notre proposition d’équivalence. Notez que la section ombragée de cette correspondance avait déjà été suggérée par les développeurs de SIMA [15].

Tableau 3.3: Proposition d’équivalence entre les identifiants d'erreurs de SIMA et de AADL

Norme ARINC 653		SIMA			AADL	
ID	NOM DE L'ERREUR	ID	État	NOM DE L'ERREUR	ID	NOM DE L'ERREUR
0	Échéance ratée d'un processus	6	5	Erreur temporelle	7	Deadline_Miss
1	Erreur de l'application d'un processus	10	5	Erreur de l'application	8	Application_Error
2	Erreur numérique d'un processus	8	5	Erreur numérique	9	Numeric_Error
3	Requête illégale d'un processus	7, 11	5	Instruction illégale, Mauvais code d'opération pour une instruction	10	Illegal_Request
4	Débordement de pile d'un processus	9	5	Débordement de pile	11	Stack_Overflow
5	Violation mémoire d'un processus	5	5	Erreur de segmentation	12	Memory_Violation
6	Faute matérielle durant un processus	N/A			13	Hardware_Fault
7	Panne de courant durant un processus	12	1, 3, 4, 5, 6, 7, 8	Panne de courant	14	Power_Fail
N/A	Erreur durant la configuration du module	2	1	Configuration du module	0	Module_Config
N/A	Erreur durant l'initialisation de la partition	3	4	Configuration de la partition	4, 6	Partition_Config, Partition_Init
N/A	Autres erreurs durant l'initialisation du module	4, 5, 7, 8, 9, 11	1	Initialisation du module, Erreur de segmentation, Instruction illégale, Erreur numérique, Débordement de pile, Mauvais code d'opération pour une instruction	1	Module_Init
N/A	Erreur durant l'exécution d'une fonction système, ou d'un changement de partition	5, 6, 7 ,8, 9, 11,	2, 9	Erreur de segmentation, Erreur temporelle, Instruction illégale, Erreur numérique, Débordement de pile, Mauvais code d'opération pour une instruction	2	Module_Scheduling
N/A	Panne de courant durant l'exécution du module	12	2, 9	Panne de courant	14	Power_Fail
N/A	Erreur durant la gestion des processus par le POS	5, 6, 7 ,8, 9, 11	4	Erreur de segmentation, Erreur temporelle, Instruction illégale, Erreur numérique, Débordement de pile, Mauvais code d'opération pour une instruction	3	Partition_Scheduling
N/A	Erreur durant le gestionnaire d'erreur de la partition	10	6	Gestionnaire d'erreur de la partition	5	Partition_Handler

3.2.1.1.1.2 Autres particularités

Bien qu'en apparence superflus, certains autres nœuds ou attributs de l'arborescence XML sont nécessaires pour le bon fonctionnement de SIMA. Puisque plusieurs ont été omis par Ocarina dans le cadre de ses autres cibles ARINC 653, ils devront être intégrés dans ce projet. Aux fins de comparaison, le tableau 3.4 présente des extraits de configuration XML nécessaires à POK et à SIMA, en surlignant les éléments qui devront être rajoutés par Ocarina.

Tableau 3.4: Éléments de configuration à rajouter sous Ocarina

Configuration A653 actuellement générée (cible : POK)	**Configuration A653 sous SIMA**
<ARINC_653_Module xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=".\A653_Part1_rev1.xsd" ModuleName="name">	<ARINC_653_Module xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=".\A653_Part1_rev1.xsd" ModuleName="name" **ModuleVersion="18-Mar-2003"**>
<Channel ChannelIdentifier="1">	<Channel ChannelIdentifier="1" **ChannelName="name"**>
<Queueing_Port Direction="SOURCE" Name="name" MaxNbMessages="2" MaxMessageSize="8"/>	<**Queuing_Port** Direction="SOURCE" Name="name" MaxNbMessages="2" MaxMessageSize="8"/>
<Partition PartitionName="pr1" PartitionIdentifier="1" EntryPoint="main" SystemPartition="true">	<Partition PartitionName="pr1" PartitionIdentifier="1" EntryPoint="**Initial**" **Criticality="LEVEL_A"** SystemPartition="true">
<Window_Schedule WindowStartSeconds="0.0000" WindowIdentifier="0" WindowDurationSeconds="1.0000"/>	<Window_Schedule WindowStartSeconds="0.0000" WindowIdentifier="0" WindowDurationSeconds="1.0000" **PartitionPeriodStart="true"**/>
<Memory_Requirements SizeBytes="15000" Type="DATA" Access="READ-WRITE"/>	<Memory_Requirements SizeBytes="15000" Type="DATA" Access="**READ_WRITE**"/>

3.2.1.1.2 Configuration du simulateur

Le second fichier XML configure des éléments propres au simulateur. Malheureusement, certains ne peuvent tout simplement pas être décrits sous AADL pour l'instant, et seront donc pris en

charge statiquement par le générateur. C'est le cas, par exemple, de la granularité du système d'exploitation, des numéros de port physique assignés aux ports APEX, du temps alloué au simulateur pour réaliser ses communications inter partition (voir section 2.1.4.2.1), et de la clé d'un segment de mémoire partagée assigné à chaque partition. D'autres éléments comme le nombre de partitions et la localisation des fichiers de script utilisé pour lancer les applications (voir section 2.1.4.5) pourront être générés avec les informations disponibles.

3.2.1.2 Code source à générer

L'objectif d'Ocarina est de générer tout le code source du système faisant usage des services APEX. Ainsi, il vise d'une part à générer le code d'initialisation de la partition qui créé les processus, les ports ARINC 653, et les autres éléments de communication inter ou intra partition offerts par APEX. D'une autre part, pour chaque tâche créée, s'il y a lieu, il se charge de générer le code pour les requêtes de lecture ou d'écriture sur les ports, ainsi que pour l'usage des structures de communication intra partition (i.e. EVENTS, BLACKBOARD, etc.). Les données reçues via ces services sont ensuite transmises aux fonctionnalités développées par l'usager dont le code source est préalablement lié au modèle AADL.

Malgré la conformité de SIMA à ARINC 653, certaines parties du code généré diffèrent des cibles précédentes d'Ocarina. Des modifications au générateur devront donc être apportées. Plusieurs de ces changements étant mineurs, ils sont présentés dans le tableau 3.5 afin d'éviter d'alourdir le texte. Les éléments entre accolades y présentent l'énumération des valeurs possibles. Les éléments en caractères **gras** soulignent les particularités.

En dehors de ces modifications, précisons les bibliothèques utilisées par le code source, c'est-à-dire les inclusions propres à SIMA, qui devront être insérées correctement par le générateur. Ce dernier devra également tenir compte que l'unité de temps utilisée par SIMA, tant pour les mises en attentes que les périodes, est la nanoseconde, contrairement à la milliseconde utilisée notamment par POK.

Tableau 3.5: Modifications à apporter au code d'initialisation généré par Ocarina

Code d'initialisation de partition générée (cible : POK)	Code d'initialisation de partition sous SIMA

`PROCESS_ID_TYPE arinc_thread;`
 	
`int main(){`
`PROCESS_ATTRIBUTE_TYPE tattr;`
`RETURN_CODE_TYPE ret;`
`  tattr.ENTRY_POINT = function_name;`
`  tattr.DEADLINE = {0…∞};`
`  tattr.PERIOD = {0…∞};`
`  tattr.TIME_CAPACITY = {0…∞};`
 	
`CREATE_PROCESS (&(tattr),`
`&(arinc_thread), &(ret));`
 	
`}`
	`PROCESS_ID_TYPE arinc_thread;`
	
	`int entry_point(){`
	`PROCESS_ATTRIBUTE_TYPE tattr;`
	`RETURN_CODE_TYPE ret;`
	`  tattr.ENTRY_POINT = function_name;`
	`  tattr.DEADLINE = {SOFT, HARD};`
	`  tattr.PERIOD = {0…∞};`
	`  tattr.TIME_CAPACITY = {0…∞};`
	`  tattr.BASE_PRIORITY = {1…63};`
	`  strcpy(tattr.NAME, "processname");`
	
	`CREATE_PROCESS (&(tattr),`
	`&(arinc_thread), &(ret));`
	
	`START(arinc_thread, &(ret));`
	`}`

3.2.1.3 Autres fichiers nécessaires

Parmi les autres éléments nécessaires à l'exécution d'un système sous SIMA, on retrouve le fichier de compilation (de l'anglais : « Makefile »), les scripts de lancement de partitions et la structure des répertoires. Puisque les options de compilation dépendent beaucoup du code source développé par l'usager, dont le contenu est inconnu du modèle AADL, seule la génération d'un canevas sera possible. Par contre, les scripts de lancement des applications et la structure de répertoire générée étant relativement statiques, ceux-ci devraient pouvoir être générés automatiquement. La figure 3-3 présente le travail qu'aura à faire le générateur à l'égard de la structure de répertoires. On y souligne également les fichiers qui sont déjà, en totalité ou en partie, générés pour une des cibles ARINC 653 existantes, soit POK.

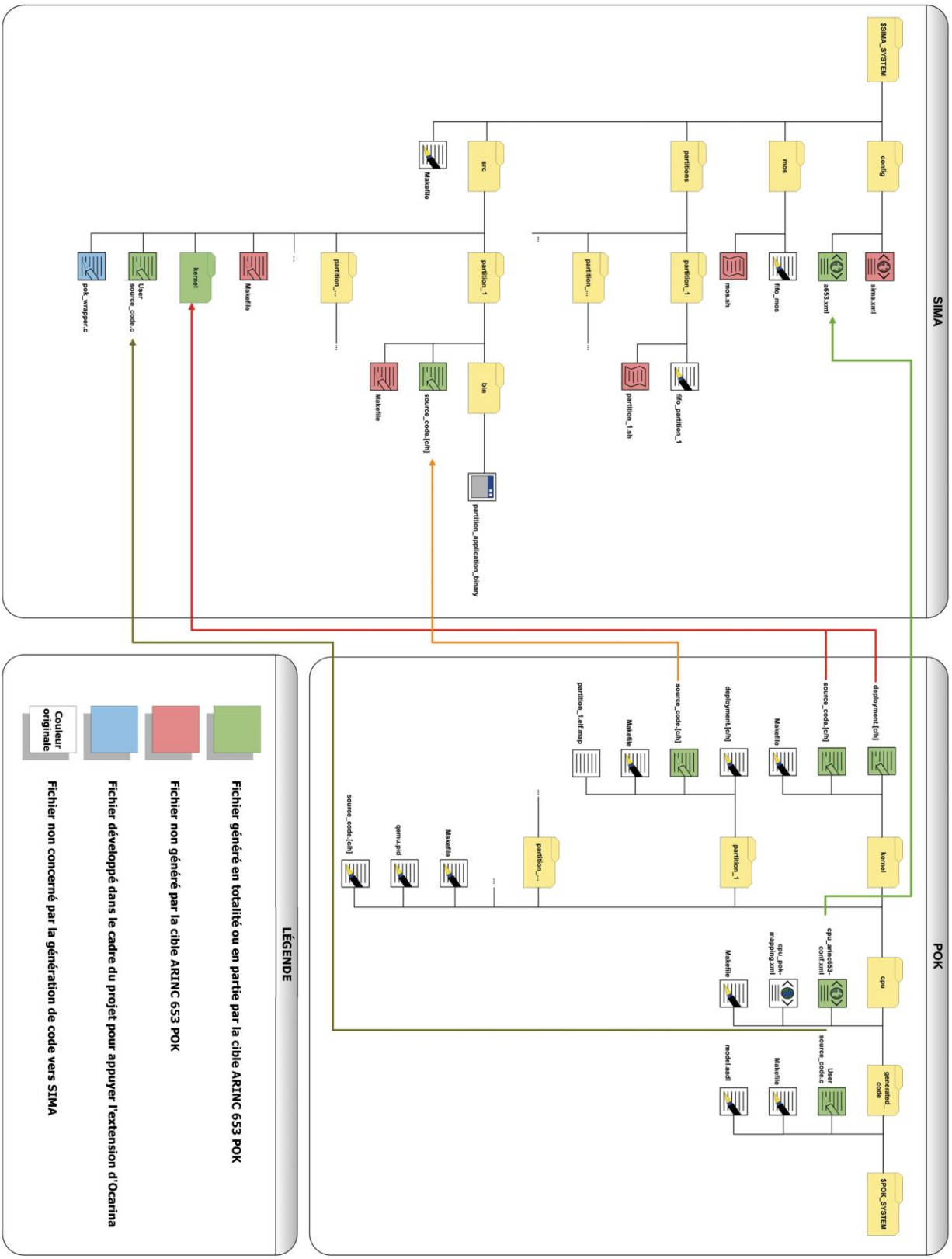

Figure 3-3: Travail à faire pour cibler SIMA par rapport à une autre cible ARINC 653

3.2.2 Preuves de concepts

Avant d'entreprendre l'implémentation de l'extension, nous avons d'abord voulu valider que la prise en charge de toutes les particularités répertoriées ci-haut par le générateur sera suffisante pour obtenir un système fonctionnel sous SIMA à partir d'un modèle AADL.

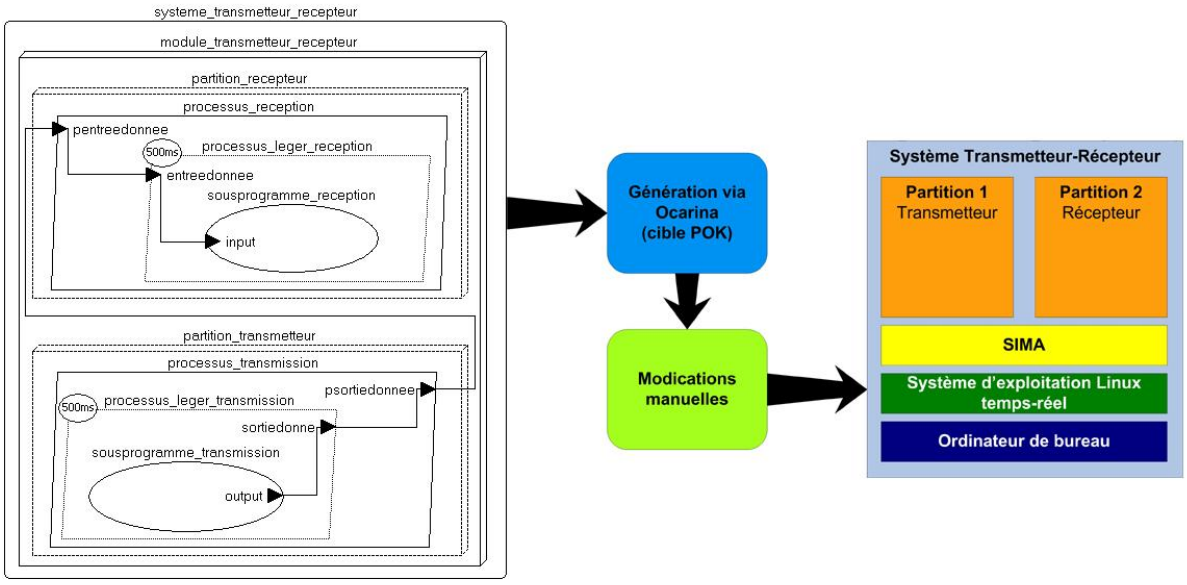

Figure 3-4: Preuve de concept de génération de code vers SIMA à partir d'un modèle AADL

Ainsi, nous avons fait quelques preuves de concept en incorporant manuellement les modifications identifiées en 3.2.1 à de petits systèmes générés à partir de modèle AADL pour une autre cible ARINC 653 (i.e. POK). Les modèles AADL utilisés étaient déjà fournis comme exemples dans le cadre du projet POK, et couvraient une multitude de concepts ARINC 653. La figure 3-4 présente une des preuves de concepts réalisées ici, dans laquelle on retrouve le modèle AADL d'un simple système de transmetteur-récepteur via ports d'échantillonnages, qui a été envoyée à la version actuelle d'Ocarina, et auquel nous avons intégré ensuite les particularités de SIMA répertoriées et que nous avons finalement exécuté sur le simulateur. Les autres preuves de concept réalisées comprennent un système de transmetteur-récepteur par ports de mise en file

d’attente, et un système illustrant l’utilisation d’évènements.

3.2.3 Implémentation

La réalisation de cette extension à Ocarina étant un objectif, non seulement du présent projet, mais également de l'ensemble du projet AVIO 509, l'implémentation de celle-ci dans le générateur a été réalisée dans le cadre des travaux de L. Bao [67], un autre étudiant membre du projet. Ses travaux ont fait suite aux résultats obtenus aux étapes 3.2.1 et 3.2.2. Pour cette partie du projet, notre participation s’est concentrée à un rôle consultatif (participation à la révision du design et à l’implémentation de certains correctifs). Toutefois, l'essentiel des modifications répertoriées a été implémenté par L. Bao. La contribution du présent mémoire s'est donc concentrée dans les autres étapes du projet, soit l'étude et l'identification des particularités de SIMA par rapport aux autres cibles ARINC 653, et la réalisation des preuves de concept. Tel qu’il sera présenté au prochain chapitre, nous avons également été maîtres d’œuvre d’un cas d’utilisation pour la première version de ce générateur dans le but d’en identifier les limitations et les avantages les plus évidents.

CHAPITRE 4 ÉTUDE DE CAS

Ce chapitre présente les travaux expérimentaux entrepris dans le cadre de ce projet de maîtrise. Nous présenterons dans un premier temps les caractéristiques de l'application avionique réalisée, ainsi que la façon dont elle a été utilisée dans le cadre de l'étude de cas visant à illustrer les capacités de SIMA. Dans un second temps, toujours à l'aide de l'application développée, nous présenterons un cas d'utilisation du générateur étendu présenté au chapitre 3.

4.1 Expérimentations sous SIMA

Comme indiqué précédemment, il existe peu de savoir-faire dans le milieu académique québécois autour des environnements d'exécutions alternatifs comme SIMA. Afin de pallier à cet obstacle, une application avionique de base sera implémentée sous cet environnement dans le cadre d'une étude de cas.

Dans un premier temps, cela permettra d'évaluer de façon expérimentale les capacités et les limitations du simulateur. Des caractéristiques comme la performance, la communication extra-partition, le déploiement sur une distribution embarquée de Linux, et l'usage de code orienté-objet (C++) seront étudiées. L'application devra donc couvrir ces éléments.

Dans un second temps, cette application sera utilisée comme cas d'utilisation pour notre extension à Ocarina qui sera présentée à la section 4.2. Celle-ci visera la génération automatique des fichiers de configuration, ainsi que du code source initialisant le système et appelant les services APEX. Un modèle AADL décrivant l'intégration de l'étude de cas dans un système ARINC 653 devra également être rédigé.

4.1.1 Description de l'application développée

La nature de l'application et du système a été proposée par CMC Électronique, un de nos partenaires industriels. Il s'agit d'un système représentant une unité de contrôle et d'affichage

multiusage (de l'anglais : « Multipurpose Control and Display Unit » ou MCDU). Le rôle de cette unité est de communiquer avec un ou plusieurs sous-systèmes avioniques, par exemple avec un système de gestion de vol (de l'anglais : « Flight Management System » ou FMS), et d'en afficher les options de contrôle et les messages destinés à l'usager sur un écran. L'usager est donc en mesure de sélectionner un sous-système, et de lui envoyer des commandes via un clavier alphanumérique. En bref, il s'agit globalement d'une interface de contrôle. La figure 4-1 présente l'aspect d'un MCDU.

Figure 4-1: Unité de contrôle et d'affichage multiusage

Dans le contexte d'une étude de cas visant la démonstration du simulateur sélectionné au chapitre 3, le choix d'un système de MCDU est intéressant pour plusieurs raisons. Tout d'abord, son fonctionnement et son protocole de communication sont définis par la norme ARINC 739A-1 [18], ce qui implique qu'il devrait pouvoir interagir avec tout FMS qui respecte la norme. De plus, cela met à notre disposition une documentation détaillée pour son implémentation. Le

format des messages et des données échangées, ainsi que la séquence de ceux-ci, sont spécifiés très clairement. Les temps de réponse maximaux requis pour leur transfert également. Puisque CMC Électronique a mis à notre disposition un simulateur de sous-systèmes avioniques, qui simule entre autres un FMS, le développement d'un système de MCDU nous permettra d'interagir avec celui-ci.

Finalement, l'application sera nécessairement implémentée en code orienté-objet, ce qui explorera l'usage du langage C++ et de ses bibliothèques sous l'environnement SIMA.

4.1.2 Implémentation de l'application

4.1.2.1 Caractéristiques de l'environnement de développement

L'application est implémentée dans le langage C/C++, en utilisant les versions 4.4 des compilateurs GCC et G++. L'environnement de développement et d'exécution du système comprend la version 1.0 de SIMA, ainsi que la distribution 11.10 de Linux Ubuntu 32 bits. La version du noyau de l'OS est 3.0.7-rt20, et le correctif PREEMPT-RT fut appliqué pour l'atteinte d'un comportement temps-réel.

Les éléments graphiques sont développés via la version 2.1 de l'API OpenGL (de l'anglais : « OpenGL utility library » ou GLU). La version 2.6.0 des utilitaires de développement OpenGL (de l'anglais : « OpenGL Utility Toolkit » ou GLUT) est également utilisée.

Finalement, l'application fait également usage de la version 4 du cadre de développement (de l'anglais : « framework ») Qt pour l'affichage de la fenêtre contenant l'application OpenGL.

4.1.2.2 Structure du système

La figure 4-2 présente sous forme de schéma le fonctionnement envisagé du système que nous développerons. Le modèle AADL qui sera rédigé devra respecter cette configuration. Il sera composé de trois modules :

- L'affichage graphique du MCDU développé pour ce projet.
- Le système ARINC 653 effectuant les traitements et les calculs internes du MCDU
- L'application de FMS fourni par CMC

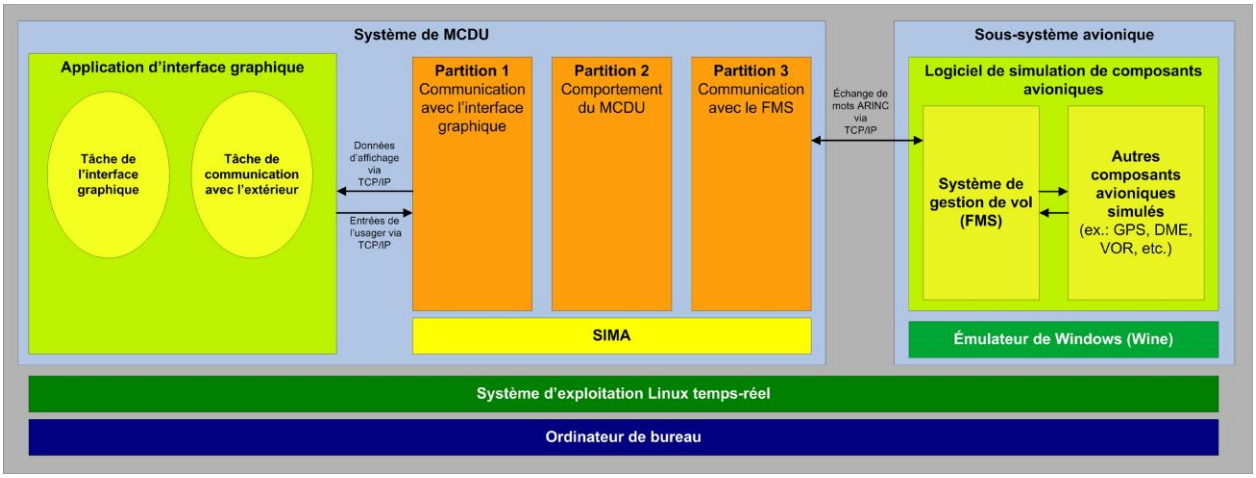

Figure 4-2: Environnement d'exécution et architecture du système MCDU

L'architecture du système ARINC 653 sera composée de trois partitions :

- Une partition s'occupant d'envoyer les données d'affichage à l'interface graphique via TCP/IP.
- Une partition calculant les états du MCDU, et s'occupant de l'encodage et du décodage des messages reçus ou à transmettre.
- Une partition s'occupant de la réception et de la transmission des messages avec un système de FMS extérieur via TCP/IP.

Ce choix de partitionnement fut principalement motivé par un objectif de modularité visant à isoler l'application du MCDU de ses interfaces de communication avec l'extérieur. De la sorte,

seuls les partitions d'interface devront être modifiées advenant l'utilisation d'un FMS ou d'une interface graphique alternative.

Selon cette architecture, deux ports de communication TCP/IP échangent des données entre l'interface graphique et le système ARINC 653. Un port transmet les sélections de l'usager sur le clavier alphanumérique au système, tandis qu'un autre transmet à l'interface graphique les mises à jour visuelles calculées par le système de MCDU à partir des informations reçues du FMS.

Parallèlement, la communication entre le FMS et le MCDU échange plutôt des mots (de l'anglais : « labels ») ARINC via TCP/IP selon un protocole défini dans un document interne à CMC électronique qui a été utilisé comme référence lors de l'implémentation. Toute communication est initiée par le MCDU, qui peut soit demander une réception ou une transmission de message au FMS. Il n'y a donc ici qu'un seul port TCP/IP bidirectionnel.

Bien que le format et la séquence des mots ARINC échangés visent à être conformes à la norme ARINC 739, le FMS fourni par CMC électronique utilise plutôt une version personnalisée de ce protocole. Ainsi, afin d'établir une communication stable entre ces deux composants, il a été nécessaire d'adapter le fonctionnement du MCDU à ce protocole modifié. Le MCDU développé diffère donc de la norme sur ce point. À titre indicatif, la figure 4-3 illustre un exemple de transfert de message entre un MCDU et un FMS, où sont soulignées certaines différences clés entre le protocole de la norme ARINC 739 et celui utilisé par le PTT de CMC.

Parmi les différences clés du protocole du PTT, notons la séquence de préparation du transfert qui est plus courte, et le fait que les mots de contrôle contenant les paramètres visuels du texte à afficher sont inexistants, puisque ces paramètres sont intégrés à l'envoi du texte, selon un format de mot propre au PTT.

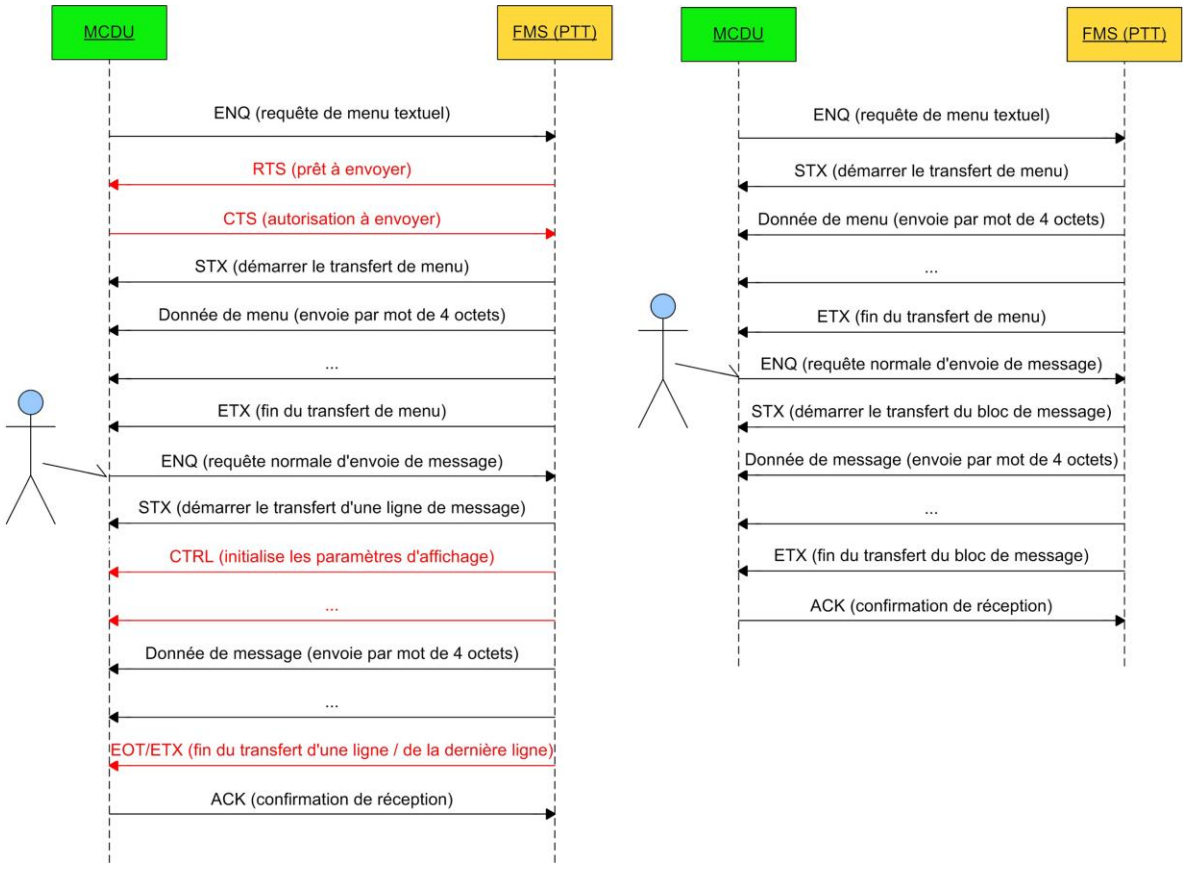

Figure 4-3: Transfert de message sous ARINC 739 (à gauche) et sous le PTT (à droite)

4.1.2.3 Structure de l'application

Les schéma UML présentés en annexe 1 illustrent les structures de l'application s'occupant de l'interface graphique, et de l'application contenue sur la partition 2 du système ARINC 653, soit l'engin de traitement du MCDU. De tout le code source rédigé, seul celui qui faisait usage des instances de ces classes, c'est-à-dire l'implémentation des tâches périodiques, ainsi que l'implémentation manuelle des services APEX, n'est pas présenté.

Lors du développement, CMC électronique a mis à notre disposition une partie du code source de l'implémentation du module de MCDU contenu dans le PTT. Ce dernier est utilisé lors de leurs simulations. Bien qu'il était impossible d'importer sans modifications ces fonctionnalités en raison de sections de code spécifiques à la plateforme, certaines ont été récupérées pour notre

étude de cas. Afin d'être transparent, l'implémentation des éléments coloriés en rouge sur ces schéma UML fut importée de ce code fourni par CMC. Les éléments surlignés en jaune furent également importés, mais ont été passablement modifiés. Cette situation fut nécessaire considérant qu'il n'existait aucune documentation papier de la version modifiée du protocole de communication ARINC 739 propre au FMS. Les fonctionnalités de décodage et d'encodage de mots ARINC ont nécessairement dû être importées.

4.1.2.4 Usage de code orienté-objet sous SIMA

Étant donné que les bibliothèques de services APEX qui sont fournies par simulateur SIMA sont développées en langage C, l'intégration sous cet environnement d'une application en code orienté-objet C++, comme la nôtre, n'est pas directe. Bien qu'aucune mention ne soit faite à cet effet dans la documentation du simulateur, nos expérimentations ont démontré que cette intégration est possible. Les déclarations et les inclusions de fichiers en C doivent simplement être d'englobés avec l'expression « `extern "C"{...}` ». C'est le cas de la bibliothèque « a653.h » qui fournit les services ARINC 653, mais également de la fonction de l'application faisant office de point d'entrée d'une partition (i.e. `int entry_point(){...}`). Les fichiers objets des archives « pos.a » contenant l'implémentation des services APEX doivent également être inclus individuellement lors de la compilation.

4.1.3 Exécution de l'application

Préalablement à son exécution, le système décrit en 4.1.2 a été configuré via le format de fichier XML utilisé par SIMA. Ce fichier peut être consulté en annexe 2.

4.1.3.1 Déploiement d'un système SIMA

Bien que le déploiement ne soit pas un objectif du simulateur SIMA, selon ses concepteurs, il serait théoriquement possible d'exécuter celui-ci sur un système embarqué, moyennant le respect des contraintes suivantes[8] :

- Le choix d'un processeur de la famille x86
- L'installation d'une version embarquée de Linux dotée au minimum des bibliothèques « librt » et « libphtread », ainsi que d'un noyau supérieur à 2.6
- La présence d'une mémoire suffisante pour contenir la distribution Linux choisie, le simulateur SIMA, et les applications du développeur.

L'entreprise GMV est ouverte à produire des versions de SIMA compatibles avec des architectures alternatives, mais cela reste conditionnel à une demande en ce sens de l'un de ces clients actuels ou futurs. Le coût estimé pour un tel port varie selon l'architecture visée[9]. Par exemple, un port vers l'architecture PowerPC, qui est répandue dans le domaine avionique, oscillerait entre 2000$ et 5000$. D'un autre côté, un port vers l'architecture Microblaze, peu maîtrisée par l'entreprise, est évalué à plus de 10000$.

Néanmoins, puisque le déploiement d'un système SIMA pourrait logiquement permettre à un système d'atteindre des performances de simulation plus près de la réalité qu'une exécution sur un ordinateur de bureau, nous souhaitons démontrer la faisabilité de cette approche. Ainsi, nous démontrerons que le système MCDU de ce mémoire peut s'exécuter sur une distribution embarquée de Linux, c'est-à-dire avec une quantité fortement réduite de paquets et de bibliothèques logicielles.

Dans le cadre de notre expérimentation, c'est une distribution de Linux embarqué dérivé de Debian qui a été sélectionnée [68]. Nous avons fait usage du simulateur QEMU [69] afin

---

[8] T. Schoofs, GMV, Discussions électroniques via courriel, 8 avril 2011

[9] T. Schoofs, GMV, Discussions électroniques via courriel, 7 juin 2011 et 21 septembre 2011

d'émuler un processeur de type x86 pour y installer cet OS. Bien qu'une carte matérielle aurait pu être préférable, cette situation sera néanmoins suffisante pour atteindre notre objectif qui est de démontrer la faisabilité de l'approche. Les résultats seront présentés au prochain chapitre.

4.2 Cas d'utilisation pour le générateur étendu

Puisque nous disposions, dans le cadre de ce projet, d'une source limitée d'application ARINC 653 dotée d'un modèle AADL correspondant, il nous est difficile de valider rigoureusement le générateur étendu qui a été développé. Néanmoins, afin d'offrir un cas d'utilisation représentatif, capable au minimum d'illustrer les avantages et les contraintes issues de cette génération de code, nous avons décidé d'utiliser le système de MCDU, développé dans le cadre de ce projet, comme cas d'utilisation pour cette nouvelle cible d'Ocarina.

Pour ce faire, un modèle AADL a été développé à partir de la configuration réalisée manuellement lors de l'étude de cas de la section 4.1.2. Les éléments modélisés couvrent principalement les concepts logiciels (module, partition, processus, etc.), à l'exception de la configuration de la mémoire qui est matériel. Il en va de même pour les implémentations liées, puisque seul du code source en C fut intégré au modèle. Les prochaines figures présentent quelques extraits du modèle.

```
processor implementation mcdu module.impl
subcomponents
   partition_com_fms :
      virtual processor partition.com_fms;
   partition mcdu application :
      virtual processor partition.mcdu_application;
   partition_com_gui :
      virtual processor partition.com_gui;
properties
   ARINC653::Module_Major_Frame => 1000ms;
   ARINC653::Partition_Slots => (350ms, 350ms, 300ms);
   ARINC653::Slots_Allocation =>
      (reference (partition com gui),
      reference (partition_com_fms),
      reference (partition_mcdu_application)
      );
end mcdu_module.impl;
```

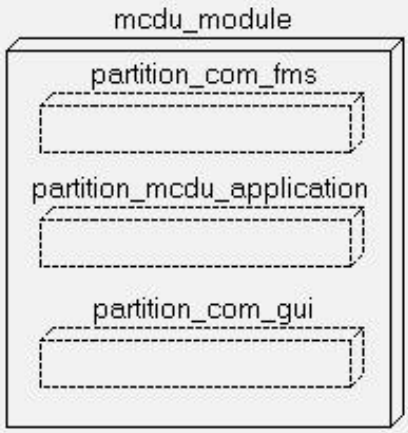

Figure 4-4: Modélisation du module du système MCDU

Le concept de module du système ARINC 653 du MCDU est modélisé à l'aide du composant « processor » du langage AADL, tel que présenté à la figure 4.4. Les partitions qui le composent sont indiquées, ainsi que les caractéristiques d'ordonnancement du cadre temporel principal.

```
process process_com_gui
features
   out_userinputs :
      out event data port userInputsArray
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 32;
       Queue Processing Protocol => Fifo;
      };
   in_display : in event data port CScreenText
      {Overflow_Handling_Protocol => DropOldest;
       Queue Size => 20;
       Queue_Processing_Protocol => Fifo;
      };
   in_annunc : in event data port annunciatorArray
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 24;
       Queue_Processing_Protocol => Fifo;
      };

properties
   Source_Code_Size => 10 KByte;
   Source_Data_Size => 15 KByte;
end process_com_gui;

process implementation process_com_gui.impl
subcomponents
   thr1 : thread gui_comm_thread.impl;
connections
   port in_display -> thr1.t_in_display;
   port in_annunc -> thr1.t_in_annunc;
   port thr1.t_out_userinputs -> out_userinputs;
end process_com_gui.impl;
```

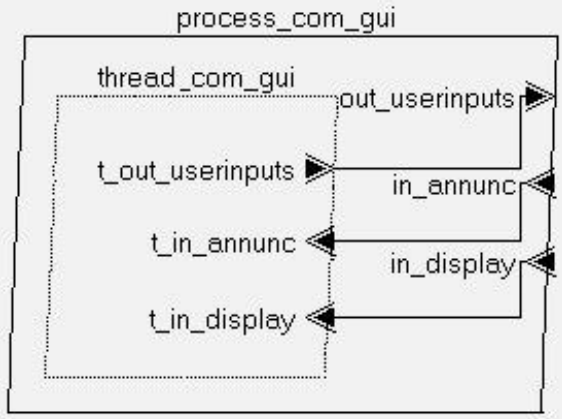

Figure 4-5: Modélisation d'une application de partition du système MCDU

L'application ARINC 653 qui est liée à une partition est représentée à l'aide du composant « process ». La figure 4.5 présente l'application de communication avec le GUI. Les ports de mise en file d'attente sont quant à eux modélisés avec le composant « event data port ». Nous avons précisé leurs caractéristiques, notamment leur profondeur (« Queue_Size »), le protocole de traitement des messages (« Queue_Processing_Protocol ») et le protocole de gestion des débordements (« Overflow_Handling_Protocol »). La taille maximale du message est définie

implicitement par le type de donnée permis sur le port. Les tâches composant l'application, ainsi que leurs connexions sur les ports de mise en file d'attente, sont également indiquées,

```
thread thread_com_gui
features
   t_in_display :
      in event data port CScreenText
        {ARINC653::Timeout => 29 ms;};
   t_in_annunc :
      in event data port annunciatorArray
        {ARINC653::Timeout => 29 ms;};
   t_out_userinputs :
      out event data port userInputsArray
        {ARINC653::Timeout => 29 ms;};
properties
   Dispatch_Protocol => Periodic;
   Compute_Execution_Time => 0 ms .. 175 ms;
   Period => 500 Ms;
   Source_Stack_Size => 400000 bytes;
   Initialize_Entrypoint  =>
      classifier (mcdu::initguicom_subprog);
end thread_com_gui;



thread implementation thread_com_gui.impl
calls
   callsend : { guicomspg :
                subprogram gui_comm_subprog;};
connections
   parameter t_in_display ->
                       guicomspg.indisplay;
   parameter t_in_annunc ->
                       guicomspg.inannunc;
   parameter guicomspg.outuserinputs ->
                       t_out_userinputs;
end thread_com_gui.impl;
```

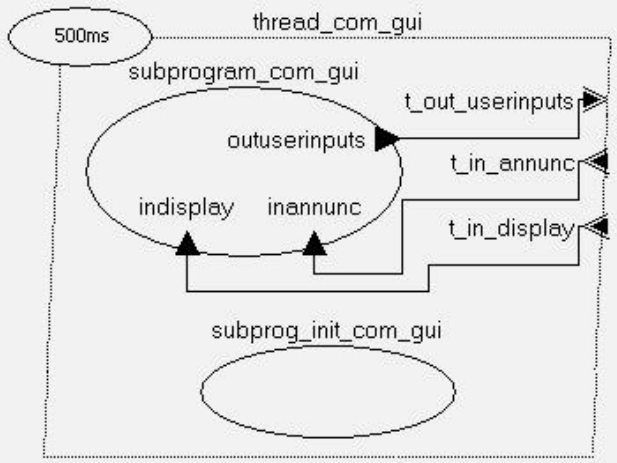

Figure 4-6: Modélisation d'une tâche du système MCDU

Les tâches exécutées à l'intérieur d'une application de partition sont représentées à l'aide du composant « thread ». La figure 4.6 présente la seule tâche associée à la communication avec le GUI du MCDU. Nous avons défini les ports de mise en file d'attente auxquels elle se connecte, la contrainte temporelle du transfert de message (« Timeout »), le type de la tâche (« Dispatch_Protocol »), son temps d'exécution requis (« Compute_Execution_Time »), sa période (« Period »), la taille de sa pile (« Source_Stack_Size »), ainsi que ses appels à des sous-programmes implémentés par l'usager (« Initialize_Entrypoint» et « calls »).

L'implémentation des fonctionnalités usagers décrites dans les tâches est référencée par le

composant « subprogram ». La figure 4.7 présente le sous-programme de la fonctionnalité périodique de la tâche de communication avec le GUI. Nous avons modélisé les paramètres d'entrée de la fonction (« in parameter »), ses valeurs de retour (« out parameter »), le langage du code source (« source_language »), le nom de la fonctionnalité (« source_name »), ainsi que le répertoire où récupérer le fichier binaire compilé de cette fonction (« source_text »).

```
subprogram subprogram com gui
features
   indisplay : in parameter CScreenText;
   inannunc : in parameter annunciatorArray;
   outuserinputs : out parameter
userInputsArray;
properties
   source_name => "commwithmcduapp";
   source_language => C;
   source_text => ("../../mcduguiCOM.o");
end subprogram_com_gui;
```

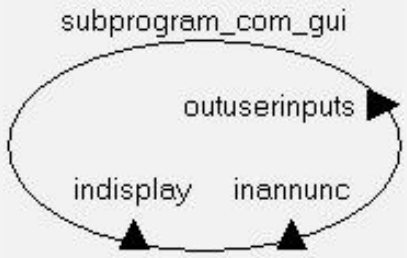

Figure 4-7: Modélisation d'un sous-programme du système MCDU

Dans le cadre de cette étude de cas, les fonctionnalités des différentes partitions du MCDU ont dû être retravaillées pour en extraire les services APEX, qui sont maintenant gérés par le générateur de code, et pour les conformer au format de ces sous-programmes AADL; dorénavant, ce sont les paramètres de la fonction qui sont utilisés pour la réception et la transmission de message.

```
memory main_memory
end main_memory;

memory partitionmemory
properties
   Word_Count => 128000;
   ARINC653::Memory_Kind => memory_code;
   ARINC653::Access Type => read write;
end partitionmemory;

memory implementation main_memory.impl
subcomponents
   part1: memory partition_memory;
   part2: memory partition_memory;
   part3: memory partition_memory;
end main_memory.impl;
```

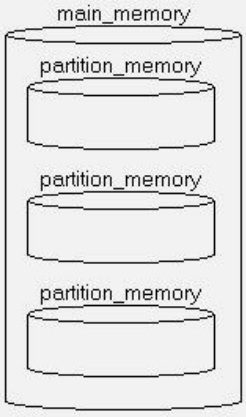

Figure 4-8: Modélisation de la mémoire du système MCDU

La configuration de la mémoire du système se fait via le composant « memory ». La figure 4.8 présente la structure la mémoire du système MCDU. Nous avons défini la mémoire allouée à chaque partition, sa taille, son type, et ses droits d'accès.

```
system implementation node.impl
subcomponents
  sys : processor mcdusys.impl;
  pr1 : process com process.impl;
  pr2 : process mcduapp_process.impl;
  pr3 : process gui_process.impl;
  mem : memory mainmemory.impl;

connections
  port pr1.out_arinc_send ->
      pr2.in_arinc_send;
  port pr2.out_arinc_rec ->
      pr1.in arinc rec;
  port pr2.out_display ->
      pr3.in_display;
  port pr2.out_annunc ->
      pr3.in_annunc;
  port pr3.out userinputs ->
      pr2.in_userinputs;

properties
  actual_processor_binding =>
  (reference (sys.partition_com_gui))
      applies to pr3;
  actual_processor_binding =>
  (reference(
      sys.partition_mcdu_application))
      applies to pr2;
  actual_processor_binding =>
  (reference (sys.partition_com_fms))
      applies to pr1;
  actual_memory_binding =>
  (reference (mem.part1))
      applies to pr1;
  actual memory binding =>
  (reference (mem.part2))
      applies to pr2;
  actual_memory_binding =>
  (reference (mem.part3))
      applies to pr3;
end node.impl;
```

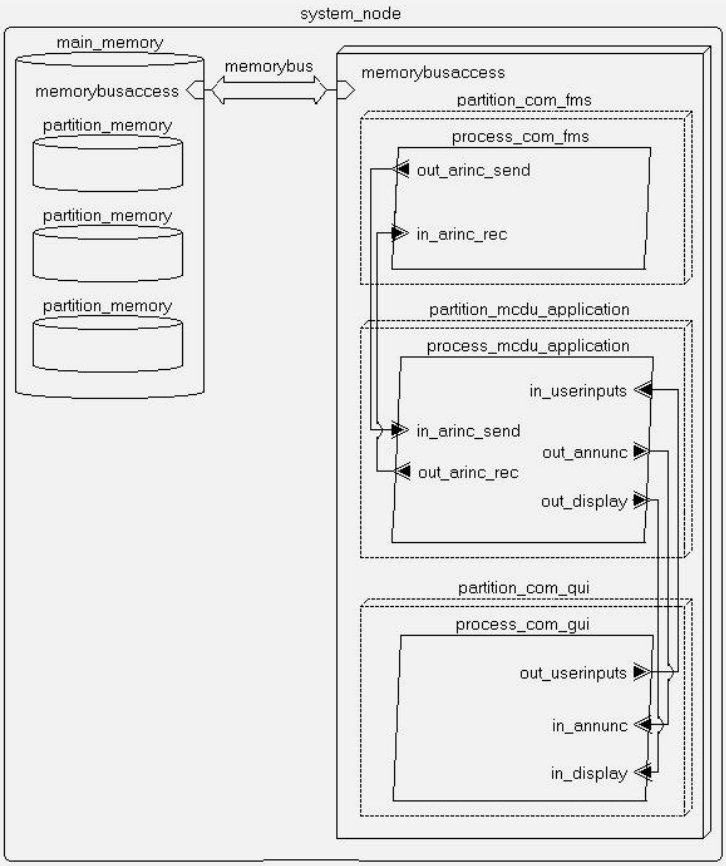

Figure 4-9: Modélisation des caractéristiques globales du système MCDU

Finalement, les caractéristiques globales du système peuvent être détaillées avec le composant « system ». La figure 4.9 présente le système de MCDU. Notre modèle associe les partitions instanciées avec leur application respective (« actual_processor_binding »), alloue la mémoire aux applications (« actual_memory_binding »), et connecte les ports APEX des différentes

partitions entre eux (« connections »).

Pour référence, le modèle du système est illustré graphiquement dans son entièreté à la figure 4-10, et la version textuelle complète du modèle est disponible en annexe 3. C'est d'ailleurs en mode textuel qu'a été rédigé le modèle dans le cadre de ce projet. Bien qu'une édition graphique sous l'environnement ADELE de TOPCASED, mentionné à la section 2.2.1.1, fut initialement envisagée, de multiples problèmes fonctionnels furent rencontrés lors de son utilisation. Ces aléas nous ont convaincus de procéder de façon textuelle, puisque cette dernière approche nous semblait plus stable en attendant la maturité de ces outils graphiques.

Les résultats de ce cas d'utilisation, notamment le comparatif entre le système MCDU configuré manuellement et celui généré, seront présentés au prochain chapitre.

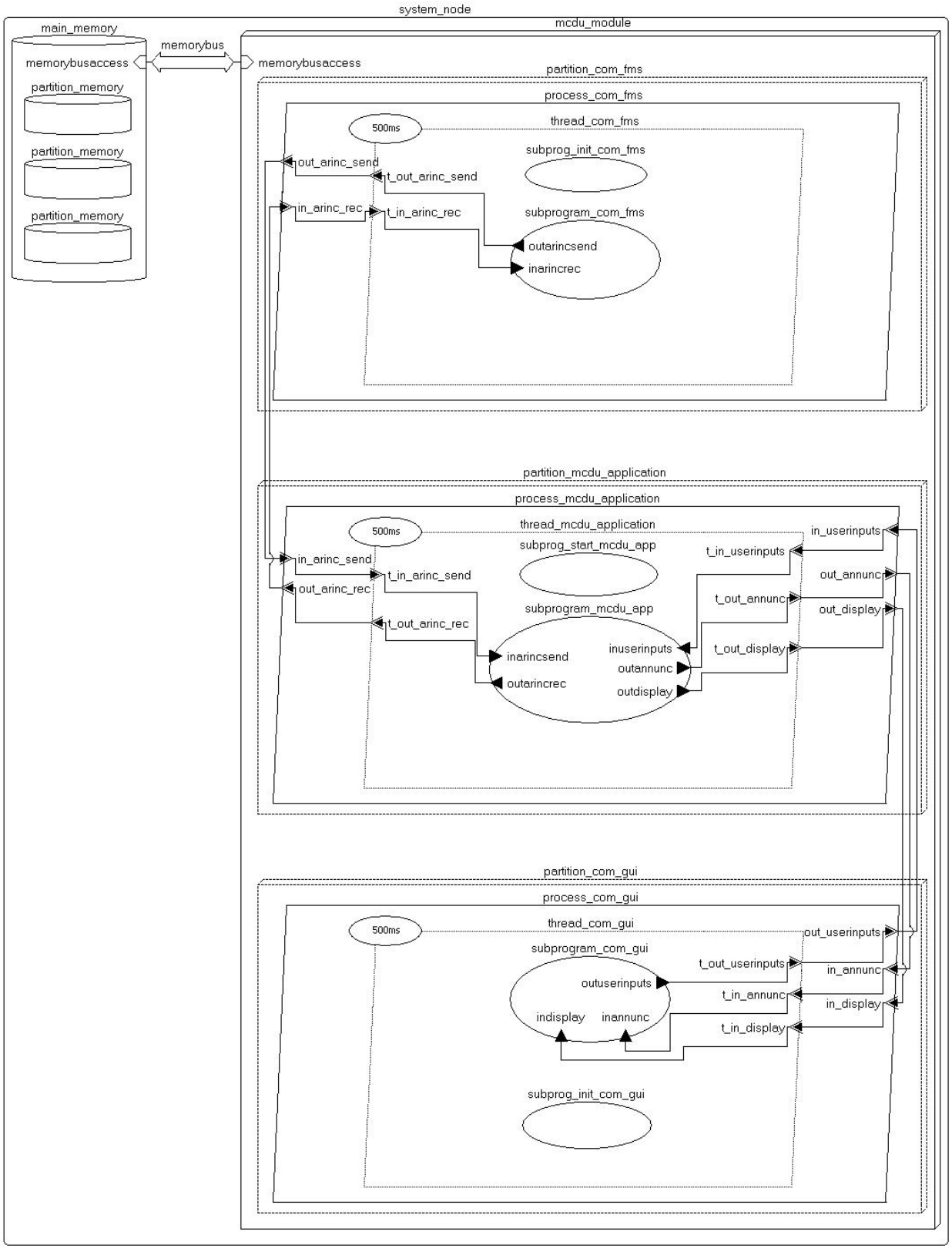

Figure 4-10: Modèle AADL du système de MCDU

CHAPITRE 5 RÉSULTATS

Ce chapitre présente les résultats obtenus suite à la réalisation des travaux expérimentaux présentés dans le chapitre précédent. Dans un premier temps, les résultats sur le développement et l'exécution du système de MCDU sous SIMA seront présentés, suivis de quelques configurations exploratoires additionnelles. Dans un second temps, ce chapitre présentera l'état actuel de l'extension au générateur de code qui a été réalisée, ainsi que les résultats du cas d'utilisation en faisant usage.

5.1 Étude de cas sous SIMA

Comme mentionné précédemment, le développement de cette étude de cas visait premièrement à contribuer au développement du savoir-faire de notre équipe de recherche avec cet outil de simulation, ainsi qu'avec les applications conformes à ARINC 653. À cette fin, cette section présentera diverses observations expérimentales sur le développement de système dans l'environnement SIMA, ainsi que sur leur exécution subséquente. Il ne s'agit pas ici d'un examen exhaustif de toutes les fonctionnalités du simulateur, mais bien les simples constats découlant d'une étude de cas. L'objectif demeure que ces observations puissent être utiles et pertinentes dans le cadre de futurs travaux faisant usage de cet outil. Nous observerons donc la performance, la communication extra-partition, le déploiement sur une distribution embarquée de Linux, et l'usage de code orienté-objet (C++).

5.1.1 Statistiques sur le système

Le système a été exécuté sur un ordinateur portable doté d'un processeur Intel Core 2 Duo T7200 cadencé à 2,00 GHz, d'une mémoire DDR2 SDRAM de 2 Go et d'un processeur graphique de type GMA 950 d'Intel. Le logiciel de « CMA-9000 FMS Part-Task Trainer » fut exécuté sur la même machine, par-dessus le logiciel Wine 1.3.28, un simulateur du système Windows.

Le noyau Linux temps-réel ayant été installé, nous avons évalué la granularité du système en exécutant l'application « cyclictest [70]» durant une trentaine de minutes. La latence maximale observée fut de 64 µs. Il est ainsi raisonnable d'estimer qu'un temps de réaction du système peut être garanti pour une granularité de 100 µs et plus.

5.1.2 Configuration initiale

À la suite des travaux expérimentaux, le système de MCDU proposé en 4.1.2 a été implémenté avec succès. Les opérations nécessaires de déverminage inhérentes à tout développement logiciel ont ainsi été menées avec succès sous l'environnement SIMA.

Sous cette configuration, les fenêtres d'exécution des partitions de communication avec le GUI et avec le FMS sont de 350 ms, tandis que celle de l'unité de traitement du MCDU est de 300ms. Le cycle temporel principal est donc de 1000 ms, soit la fréquence maximale observée à laquelle le MCDU doit envoyer le mot confirmant son bon fonctionnement au FMS. Les tableaux 5.1, 5.2 et 5.3 présentent quelques statistiques et quelques résultats d'exécution sur le système résultant.

Tableau 5.1: Statistiques générales sur le système de MCDU développé

Partitions / Applications	**Taille du binaire**	**Lignes de code**
Unité de traitement du MCDU	316 Ko	4257
Communication avec FMS	136 Ko	751
Communication avec GUI	175 Ko	577
Fichiers de code source partagés	*N/A*	1787
Total du système IMA	627 Ko	7372
Unité de GUI du MCDU	1229 Ko	5267
Total de tout les systèmes	1856 Ko	12639

Tableau 5.2: Nombres d'échéances ratées par partition après 5 minutes de simulation

Partitions / Applications	**Échéances ratées par simulation**										
	Sim. 1	Sim. 2	Sim. 3	Sim. 4	Sim. 5	Sim. 6	Sim. 7	Sim. 8	Sim. 9	Sim. 10	**Total**
Unité de traitement du MCDU	2	2	0	0	0	0	2	0	0	2	**8 (0,3%)**
Communication avec FMS	20	23	12	26	19	23	19	19	14	23	**198 (6,6%)**
Communication avec GUI	2	1	0	2	2	1	2	1	2	1	**14 (0,5%)**

Tableau 5.3 : Statistiques d'exécution après 5 minutes de simulation

Numéro de simulation	**Nombre de blocs de données reçus**	**Nombre de transferts de page initiés**	**Nombre de transferts de page interrompus**
1	436	13	1
2	301	10	3
3	277	9	3
4	381	11	2
5	264	9	4
6	421	11	1
7	317	11	3
8	380	10	2
9	397	11	2
10	329	11	3
MOYENNE	**350,3**	**10,6**	**2,4 (22,6%)**

Au regard du tableau 5.3, il apparait que le transfert des messages contenant les informations à afficher entre le PTT et le MCDU est fréquemment interrompu lors des simulations. En moyenne, 22,6% des chargements de page sont ainsi affectés. Après analyse, il apparaît que cette situation

survient en raison de latences issues du transfert de message entre les deux entités, et des retards accumulés, qui empêchent le MCDU d'envoyer ses accusés de réception au FMS dans les délais requis. Une nouvelle requête de transfert est alors nécessaire pour le renvoi de la page.

Ces accusés de réception retardataires sont attribuables aux délais parfois très longs de transmission ou de réception de message avec le PTT via TCP/IP. Ceux-ci entrainent également la partition à rater ses échéances, tel qu'indiqué dans le tableau 5.2, ce qui décale son comportement pour les fenêtres d'exécution suivantes, et amene parfois la transmission des accusés de réception plusieurs cycles après l'envoie par l'unité de traitement du MCDU.

La figure 5.1 illustre graphiquement ce problème. Il est impossible de le régler en rallongeant la fenêtre d'exécution de cette partition au pire cas, puisque cela rallongerait également le temps de réponse du système de MCDU, nuisant conséquemment au transfert des messages avec le FMS.

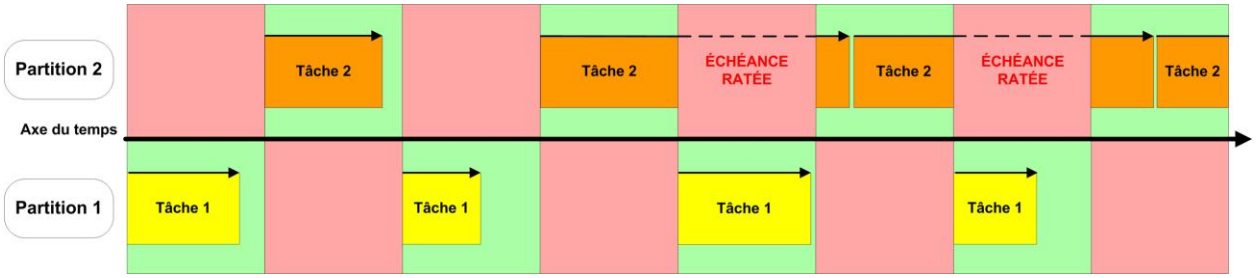

Figure 5-1 : Illustration du problème de désynchronisation causé par une échéance douce ratée

5.1.3 Configuration corrigée

Afin de mitiger ce problème de latence, qui est spécifique à notre système, la solution mise de l'avant est de revoir la configuration et l'architecture du système en adoptant celle présentée à la figure 5.2.

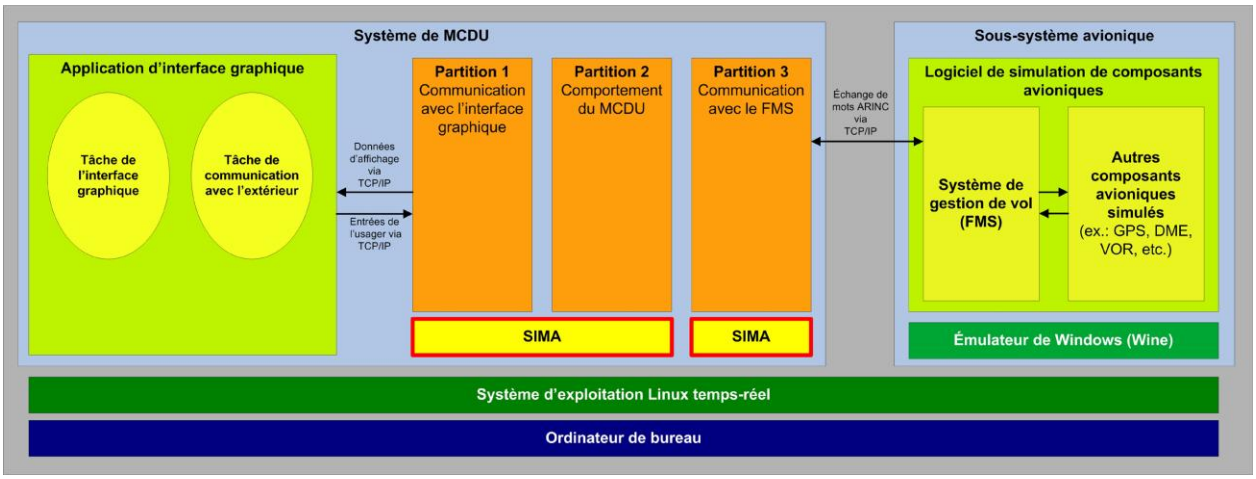

Figure 5-2 : Environnement d'exécution et architecture corrigée du système MCDU

Ainsi, la partition chargée des communications avec le FMS via TCP/IP est dorénavant exécutée sur un système distinct, avec son propre MOS, où elle est la seule partition. Son application est dorénavant une tâche apériodique s'exécutant en continu, c'est-à-dire sans échéance particulière. Les latences des communications demeurent tout aussi inégales, mais n'affecte plus autant son comportement. Advenant une grosse latence, elle est désormais en mesure de rattraper son retard puisqu'elle s'exécute en parallèle avec l'autre système. Pour les besoins de ce mémoire, afin de conserver un comparatif équitable entre les configurations, la durée du cycle temporel principal et des fenêtres d'exécution des deux autres partitions demeurent inchangées. Le temps auparavant attribué à la partition retirée est ainsi laissé inutilisé.

Parallèlement, un autre problème de conception fut identifié. Le temps de réponse inégal de la partition des communications avec le FMS générait fréquemment des débordements de pile sur les ports de messages. En effet, puisque des messages pouvaient s'accumuler sur la pile du port dans les moments où le temps de réponse du FMS était plus rapide que le cycle du MCDU, ou encore lorsque le MCDU ratait une échéance, et que celui-ci n'effectuait qu'une seule lecture sur ces ports à chaque cycle, cela pouvait causer un décalage dans la réception des messages, et parfois un débordement. Les appels aux services APEX de lectures ont donc été modifiés pour que chaque partition puisse lire tous les messages sur sa pile à chaque cycle dans la mesure où la quantité de données reçues reste inférieure à la capacité de traitement de la partition. Par

exemple, un maximum de 256 mots ARINC peut être envoyé à la fois au FMS. Il en va de même pour le traitement des mots par l'engin MCDU.

Concernant la communication TCP/IP entre le MCDU et son interface graphique, dont la partition a également raté des échéances selon le tableau 5.2, le problème a été résolu simplement. Considérant que les mêmes informations sont constamment renvoyées entre ces deux partitions, et qu'un échec en transmission de message n'est pas très dommageable, la lecture sur leurs ports a été modifiée pour être non-bloquante après une durée d'attente assurant le respect des contraintes temporelles en cas d'échec.

À la suite des résultats de simulations, présentés dans les tableaux 5.4 et 5.5, si peu de changements sont observés au niveau des échéances ratées pour les deux partitions soumises à des contraintes temporelles, les statistiques d'exécution montrent une nette amélioration au niveau des transferts de page réussis. Dorénavant, seuls 1,4% des transferts de page sont ici interrompus.

L'environnement SIMA ne permet pas d'aller beaucoup plus loin dans l'analyse des résultats d'une simulation. Ce dernier ne dispose pas encore de son propre environnement de traçage, et il est impossible, selon les développeurs de SIMA, d'en utiliser un propre à Linux comme le « Linux Trace Toolkit Next Generation (LTTng) [71] » puisqu'il affecte de façon indésirable son ordonnanceur[10]. Néanmoins, ce simulateur offre tout de même un environnement d'exécution fonctionnel pour une application ARINC 653 qui permet d'identifier des problèmes de conception sans pour autant faire usage d'une plateforme matérielle ou d'un système d'exploitation commercial, ce qui est précisément l'objectif recherché.

[10] T. Schoofs, GMV, Discussions électroniques via courriel, 14 novembre 2011

Tableau 5.4: Nombres d'échéances ratées par partition après 5 minutes de simulation

Partitions / Applications	**Échéances ratées par simulation**										
	Sim. 1	Sim. 2	Sim. 3	Sim. 4	Sim. 5	Sim. 6	Sim. 7	Sim. 8	Sim. 9	Sim. 10	**Total**
Unité de traitement du MCDU	0	0	0	1	2	2	0	2	0	0	**7 (0,2%)**
Communication avec GUI	1	2	2	1	2	1	2	1	0	0	**12 (0,4%)**

Tableau 5.5 : Statistiques d'exécution lors de simulations de 5 minutes

Numéro de simulation	**Nombre de blocs de données reçus**	**Nombre de transferts de page initiés**	**Nombre de transferts de page interrompus**
1	457	13	0
2	463	13	0
3	438	14	0
4	333	12	2
5	482	14	0
6	505	15	0
7	514	15	0
8	525	16	0
9	506	16	0
10	506	15	0
MOYENNE	**472,9**	**14,3**	**0,2 (1,4%)**

5.1.4 Exécution du système sur une distribution embarquée de Linux

La figure 5-3 présente la configuration du système mise de l'avant pour cette expérimentation présentée à la section 4.1.3.1. Pour l'installation de l'OS, la taille mémoire nécessaire au disque virtuel utilisé par QEMU a été établie à 190 Mo, puisque la taille totale utilisée par notre système était de 177 Mo. En effet, la version de base de l'OS embarqué utilisé requiert un espace mémoire minimal de 128 Mo, auquel quelques paquets, notamment « tftp », ont dû être rajoutés. À cela s'ajoutent les 627 Ko du système MCDU, et la taille minimale de 1084 Ko du simulateur SIMA, c'est-à-dire la taille de l'outil MOS, ainsi que des fichiers de configuration et de scripts nécessaires. La taille totale de ce système SIMA, si l'on omet le système d'exploitation, est uniquement de 1,7 Mo.

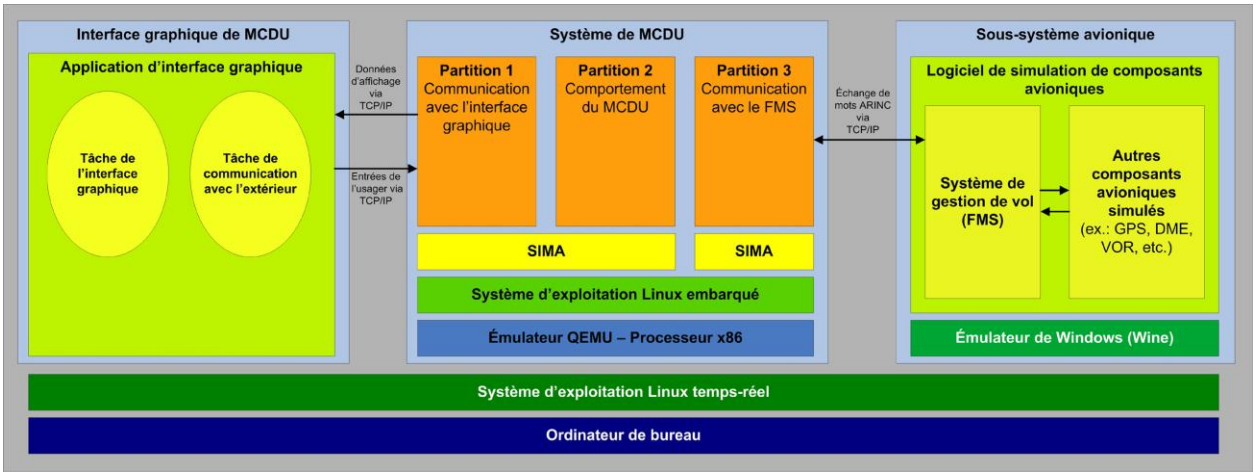

Figure 5-3: Environnement du système MCDU lors de l'expérimentation de déploiement

L'opération a été réalisée avec succès. Évidemment, le simulateur de processeur QEMU étant exécuté sur le système hôte décrit en 5.1.1, les latences du système sont vraiment très élevées. En effet, la latence maximale observée après 30 minutes de l'outil « cyclictest » est de 37636 µs, comparativement à 64 µs lorsqu'exécuté directement sur l'hôte. Cette situation a évidemment impacté les performances du système comme en témoignent les simulations présentées dans les tableaux 5.6 et 5.7.

Tableau 5.6: Nombres d'échéances ratées par partition après 5 minutes de simulation

Partitions / Applications	**Échéances ratées par simulation**										
	Sim. 1	Sim. 2	Sim. 3	Sim. 4	Sim. 5	Sim. 6	Sim. 7	Sim. 8	Sim. 9	Sim. 10	**Total**
Unité de traitement du MCDU	0	0	0	0	2	0	2	0	1	0	**5 (0,2%)**
Communication avec GUI	2	1	1	1	1	1	1	1	1	1	**11 (0,4%)**

Tableau 5.7 : Statistiques d'exécution lors de simulations de 5 minutes

Numéro de simulation	**Nombre de blocs de données reçus**	**Nombre de transferts de page initiés**	**Nombre de transferts de page interrompus**
1	454	6	2
2	567	8	0
3	555	7	0
4	565	8	0
5	508	9	0
6	527	8	0
7	516	7	0
8	518	7	0
9	494	8	0
10	510	8	0
MOYENNE	**521,4**	**7,6**	**0,2 (2,6%)**

On remarque que le ratio moyen de mots ARINC envoyé par page est plus de 2 fois supérieur sous cette configuration. Cela signifie que le FMS a répété deux fois plus souvent les messages envoyés au MCDU, et donc, qu'il tardait à recevoir ses accusés de réception. Cette situation explique donc la quantité plus petite de transfert de page initiés. À la lumière de ces données, il semblerait que le système soit donc plus lent à réagir qu'auparavant, ce qui est compréhensible compte tenu des latences plus élevées dans l'environnement QEMU.

Néanmoins, cette expérimentation confirme la possibilité de déployer un système SIMA sur une version embarquée, donc réduite, de Linux, dans la mesure où une taille mémoire suffisante est disponible sur la plateforme matérielle pour son installation.

Il est probable qu'une telle simulation sur la plateforme matérielle ciblée offrirait des résultats plus représentatifs du système final, ce qui pourrait être intéressant dans un contexte d'exploration architecturale. En effet, le contexte matériel de cette carte diffèrerait assurément de celui de l'ordinateur de bureau utilisé (e.g. taille de la mémoire RAM ou de la mémoire cache, performance du processeur). Dans le cadre de travaux futurs, il serait donc pertinent de vérifier cette hypothèse, conditionnellement à l'acquisition de versions de SIMA exécutables sur des processeurs alternatifs (ex. : ARM, PowerPC, etc.), et d'un système d'exploitation ARINC 653 commercial utilisé comme référentiel de comparaison.

5.2 Réalisation du générateur de code

5.2.1 Implémentation de l'extension

À la suite de mes travaux des sections 3.2.1 et 3.2.2, et de ceux de L. Bao de la section 3.2.3, notre équipe de recherche a eu à sa disposition une première version de l'extension au générateur de code Ocarina permettant de cibler le simulateur SIMA. Cette version du générateur prend en compte la plupart des concepts ARINC 653 qui peuvent être modélisés sous AADL. Les tableaux présentés en annexe 4 indiquent les caractéristiques supportées. Notre extension étant toujours en progrès, nous avons précisé deux niveaux de support, soit la capacité du générateur à lire les informations du modèle AADL et à les enregistrer dans son arbre de représentation abstraite, et sa capacité à générer l'information dans les fichiers de configuration, ou de code source, qui en ont besoin. Aussi, contrairement aux éléments surlignés en vert, les éléments en jaune sont générés statiquement. Pour cette première version de l'extension, certaines caractéristiques ne sont pas supportées, ou du moins, ne le sont que partiellement.

La principale fonctionnalité toujours absente est la génération dynamique de la configuration du gestionnaire d'erreurs, conformément à la stratégie présentée en 3.2.1.1.1.1, à partir des informations du modèle. Une configuration de gestion d'erreur est générée statiquement par défaut, mais doit pour l'instant être modifiée manuellement par le concepteur si des comportements spécifiques sont requis. Tous les autres éléments de la configuration des concepts ARINC 653 sont générés dynamiquement à partir des informations du modèle, ou statiquement dans les cas où l'information n'est pas modélisable. Le concept des pseudo partitions a toutefois été écarté des fonctionnalités supportées pour cette première version.

Pour le fichier de configuration du simulateur, les informations qui lui sont spécifiques et qui ne peuvent être représentées en AADL sont générées parfois statiquement, comme la granularité du système qui est fixé à 100 µs, et parfois dynamiquement, comme les numéros de ports physiques qui sont alloués de façon incrémentale aux ports APEX. À moins de besoins particuliers des concepteurs, ce fichier est fonctionnel et ne nécessite pas de retouches. Un tableau dressant un portrait de la génération de ces éléments se trouve également en annexe 4.

Précisons également que cette version de notre extension génère automatiquement la structure des répertoires, les scripts de lancement de partitions dont SIMA a besoin, ainsi que des patrons de fichiers de compilations (de l'anglais : « makefile »). Bien que ces derniers soient déjà partiellement configurés, certains éléments ne peuvent pas être modélisés en AADL, comme la liste complète des fichiers de code source développé par l'usager, ou les bibliothèques logicielles nécessaire à leur compilation. Ces informations doivent donc être rajoutées manuellement.

Finalement, en ce qui concerne les appels aux services APEX dans le code source généré, les concepts non supportés sont: les sémaphores qui sont créés, mais non appelés, la priorité des processus qui est statiquement générée de façon croissante selon l'ordre d'apparition, et les contraintes temporelles qui sont douces par défaut.

Il est projeté que ces limitations seront corrigées dans une version ultérieure de l'extension.

5.2.2 Cas d'utilisation

Conformément à ce qui fut annoncé au chapitre précédent, le système MCDU fut utilisé comme cas d'utilisation pour le générateur. Le modèle AADL présenté à la figure 4-4, et joint en annexe 3, fut ainsi utilisé comme fichier source. Nous avons favorisé cette architecture plutôt que la version corrigée présentée en 5.1.3, puisque le concept de pseudo partition n'est pour l'instant pas supporté. La vérification et la compilation du modèle se sont déroulées avec succès, et la structure des répertoires générée fut conforme à ce qui fut annoncé à la figure 3-3. Une totalité de 54 fichiers furent donc générés, totalisant 898 lignes de codes.

Certaines particularités nécessaires au fonctionnement du système de MCDU, observées tard durant le développement, ont toutefois dû être intégrées manuellement. C'est notamment le cas de la lecture en boucle sur les ports de mise en file d'attente pour éviter les débordements, et des éléments à préciser pour l'inclusion de code C++. Puisque les travaux d'implémentation menés par L. Bao sont réalisés en parallèle à ceux de ce travail, ces dernières particularités répertoriées n'ont pas été implémentées dans la version du générateur disponible au moment de compléter ce mémoire. À titre indicatif, le tableau 5.8 indique donc le nombre de lignes de code qui furent générées initialement, ainsi que le nombre de lignes qui ont dû être retouchées ou ajoutées pour tenir compte de ces nouvelles particularités. Il est prévu que le support de ces nouvelles caractéristiques figurera dans la prochaine version.

Les tableaux 5.9 et 5.10 présentent des statistiques sur l'exécution de ce système généré. Comme en témoignent ces données, la qualité du fonctionnement du système semble similaire à ce qui avait été observé à la section 5.1.2 et aux tableaux 5.2 et 5.3.

Tableau 5.8 : Statistiques générales sur le système généré

Partitions / Applications	**Taille du binaire**	**Lignes de code des fichiers générés**			**Lignes de code usager**
		Initiales	Modifiées	Ajoutées	
Unité de traitement du MCDU	523 Ko	257	17	58	4038
Communication avec GUI	180 Ko	234	12	50	280
Communication avec FMS	142 Ko	225	11	32	603
Fichiers de code source partagés	*N/A*	182	0	0	1879
Total du système généré	845 Ko	898	40	140	6800

Tableau 5.9 : Nombres d'échéances ratées par partition après 5 minutes de simulation

Partitions / Applications	**Échéances ratées par simulation**										
	Sim. 1	Sim. 2	Sim. 3	Sim. 4	Sim. 5	Sim. 6	Sim. 7	Sim. 8	Sim. 9	Sim. 10	**Total**
Unité de traitement du MCDU	0	0	0	0	0	0	0	0	0	0	**0 (0,0%)**
Communication avec GUI	0	0	0	0	0	0	0	0	0	0	**0 (0,0%)**
Communication avec FMS	30	37	33	16	19	32	28	49	33	27	**304 (10,0%)**

Tableau 5.10 : Statistiques d'exécution après 5 minutes de simulation

Numéro de simulation	Nombre de blocs de données reçus	Nombre de transferts de page initiés	Nombre de transferts de page interrompus
1	311	8	3
2	380	11	2
3	297	10	3
4	196	9	5
5	322	10	2
6	340	10	2
7	435	13	1
8	401	12	2
9	441	13	1
10	382	12	2
MOYENNE	**350,5**	**10,8**	**2,3 (21,3%)**

5.2.3 Avantage de la génération automatique

Dans le contexte de ce cas d'utilisation, l'intégration de l'environnement SIMA à un flot de conception comprenant un générateur de code, a grandement accéléré la migration du système MCDU vers un environnement ARINC 653. Alors que la rédaction de la configuration et des fonctionnalités APEX prit un peu moins de 1 heure, lorsqu'effectuée de façon manuelle en section 5.1.2, elle ne prend que quelques secondes grâce à la génération de code.

De plus, lors de la reconfiguration du système, considérant que des informations devaient être modifiées de façon redondante dans une quantité de fichiers, soit deux fichiers XML et plusieurs fichiers de code source, il arrive fréquemment d'oublier une petite modification. Celle-ci engendre alors une erreur de configuration qui peut parfois prendre des dizaines de minutes à identifier. La génération automatique de ces fichiers élimine donc ces erreurs humaines, et ainsi augmente la rapidité de reconfiguration du système.

Finalement, même s'il va de soi que l'apprentissage de la modélisation AADL nécessite un certain temps, il n'était dorénavant plus nécessaire de connaitre par cœur les conventions d'appel des différents services APEX, puisque ceux-ci sont générés automatiquement.

5.2.4 Désavantage de la génération automatique

Malgré les résultats fonctionnels concluants indiqués précédemment, la nécessité d'inclure les exceptions identifiées en 5.2.2 dans le code généré a mis en évidence un problème inhérent au concept de génération de code qui est difficile à mitiger, soit sa rigidité. Dans la mesure où toutes les caractéristiques ne sont pas modélisables dans le langage choisi, c'est-à-dire AADL dans le cas présent, la génération des cas particuliers peut augmenter rapidement et devenir impossible à gérer.

C'est particulièrement le cas pour la gestion des valeurs de retour provenant des appels aux services APEX. Puisque ceux-ci sont maintenant abstraits du code usager et gérés par notre extension à Ocarina, il doit en être de même des réactions à entreprendre dans les cas où ceux-ci échouent. Or, ces réactions sont la plupart du temps spécifiques au contexte de l'application et ne peuvent pas être modélisées. De plus, même s'il reste avantageux d'avoir un canevas de code source quasi fonctionnel de généré automatiquement, il n'est jamais idéal d'avoir à le modifier. Tout d'abord, il est désavantageux de devoir répéter des modifications manuelles à chaque fois que le modèle est retouché par le concepteur. Ensuite, dans un contexte industriel où le générateur produit un code certifié, cette avenue est tout simplement prohibée. Ainsi, la seule solution durable serait de rajouter une extension au langage AADL afin de pouvoir modéliser les comportements du système selon les valeurs de retour de ces appels aux services APEX. Il en irait de même pour tout autre besoin particulier. Ces contraintes n'enlèvent rien au bénéfice de la génération de code, mais indiquent selon nous des éléments à améliorer qui pourraient faciliter l'adoption de cette technologie par industrie.

CONCLUSION

Dans le cadre de ce mémoire, nous sommes parvenus à démontrer qu'il est possible d'avoir un flot de conception peu dispendieux pour les systèmes avioniques modulaire intégrés qui fait à la fois usage d'une approche de développement basée sur les modèles, ainsi que d'un environnement de simulation.

Nous avons donc atteint notre objectif de proposer aux industriels une alternative aux environnements de développement commerciaux qui sont très coûteux, et dont les capacités de déploiement et de certification ne sont pas toujours nécessaires en début de développement, ou au moment de procéder à des étapes d'explorations architecturales. Du point de vue académique, nous avons également atteint notre objectif de développer un savoir-faire local sur ces outils et ces technologies à travers le développement d'une étude de cas mettant en application plusieurs concepts, notamment un protocole de communication avionique, l'interface de programmation APEX, et le partitionnement temporel et spatial.

Au niveau de l'environnement d'exécution alternatif, nous avons sélectionné un simulateur commercial de système ARINC 653 peu onéreux appelé SIMA. Ce système est conforme à la norme ARINC 653 et a l'avantage de donner accès à un support technique plus que satisfaisant, contrairement à beaucoup d'alternatives du côté des logiciels libres.

Nous avons développé une application avionique de base sous cet environnement, soit une application d'unité de contrôle et d'affichage multiusage (MCDU), et démontré qu'il était possible d'évaluer sommairement différentes configurations de système afin d'identifier celle qui sont le plus susceptibles de diminuer des problèmes de nature fonctionnelle. La communication de l'application avec un autre système via un protocole avionique exécuté sur TCP/IP a également été illustrée. De plus, un savoir-faire considérable sur le développement d'applications conformes à la norme ARINC 653, ainsi que sur le simulateur SIMA, fut apporté au projet AVIO 509. Nous avons également démontré qu'il serait possible de déployer le simulateur et le système à exécuter sur une distribution embarquée d'un système d'exploitation Linux, advenant que des développeurs souhaitent entreprendre des simulations sur la plateforme matérielle de destination

finale.

Ensuite, au niveau de la conception par modèles, nous avons démontré qu'il était possible d'étendre le générateur de code libre de licence Ocarina afin qu'il puisse générer la majorité de la configuration du système, ainsi que le code source des applications faisant usage des services APEX, pour l'environnement d'exécution SIMA. Seule une description de l'architecture du système dans le langage de modélisation AADL est nécessaire. Bien que la version actuelle de l'extension ne couvre pas pour l'instant tous les concepts, nous avons tout de même pu démontrer à travers un cas d'utilisation que cette solution est susceptible de faire économiser beaucoup de temps aux développeurs, notamment en réduisant les erreurs manuelles. Tant la configuration que la reconfiguration et le partage d'information entre ingénieurs devraient dorénavant être plus rapides. De plus, les travaux futurs de L. Bao devraient permettre de finaliser l'extension en corrigeant les lacunes répertoriées, et ainsi concrétiser davantage les résultats.

La réalisation d'un cas d'utilisation avec un système de MCDU a néanmoins soulevé certains bémols relativement à cette approche. Certaines particularités propres à notre système de MCDU ont imposé des modifications manuelles aux fichiers générés automatiquement. Bien que plusieurs d'entre elles seront supportées par une version ultérieure de notre extension à Ocarina, d'autres, comme la gestion de la valeur de retour des appels aux services APEX, pourront difficilement être intégrés puisqu'elles ne peuvent pas être modélisées dans la version actuelle du langage AADL. En résumé, cette partie de l’expérimentation nous indique que les résultats des technologies de génération de code peuvent parfois être rigides, et qu'ils peuvent difficilement accommoder des éléments trop spécifiques d’une application. Selon nous, la résolution de ce problème passera par des extensions aux langages de modélisation afin d'incorporer les concepts qui ressortiront comme étant les plus fréquemment rencontrés. Néanmoins, contrairement à la génération de code, la génération automatique de la configuration est tout à fait fonctionnelle, même dans l'état actuel de notre générateur, ce qui est selon nous très intéressant pour les industriels qui souhaiteraient dès maintenant adopter une approche de MBE, ou encore, de notre flot de conception.

Parmi les travaux futurs à envisager, afin de poursuivre sur les contributions de ce mémoire, il va de soi que la complétion du générateur de code afin de supporter notamment la gestion des erreurs et le langage C++ est le premier élément à souligner. Il serait également pertinent de réaliser la transition automatique d'un système simulé sous SIMA vers un environnement commercial comme PikeOS ou VxWorks 653 afin de disposer d'une version entièrement fonctionnelle du flot de conception proposé. Aussi, ce mémoire a procédé à une étude de cas pour identifier les principales limitations et les principaux avantages du flot proposé. La réalisation de tests et de simulations supplémentaires avec une plus grande gamme d'exemples serait de mise si l'on souhaite valider et tester plus rigoureusement les composants du flot proposé.

En définitive, nous croyons toutefois avoir contribué à offrir des pistes de solutions aux entreprises qui souhaiteraient faire usage de technologies alternatives à moindre coût en début de développement, et ainsi élargir les options à leur disposition pour concevoir des systèmes IMA, ou pour migrer certaines de leurs applications existantes vers un tel environnement. En terminant, nous espérons que cela aidera les entreprises à maîtriser davantage les outils nécessaires à la création de ces systèmes à partitionnement robuste dans l'espace et dans le temps, et ainsi atteindre l'objectif ultime de réduire les coûts de développement et de certification, en plus du poids et de la puissance dissipée, des appareils.

RÉFÉRENCES BIBLIOGRAPHIQUES

[1] C. B. Watkins et R. Walter, "Transitioning from federated avionics architectures to integrated modular avionics," dans *Proceedings of the 26th Digital Avionics Systems Conference (DASC), Dallas, Texas*, 2007, pp. 2.A.1-1 - 2.A.1-10.

[2] R. Walter et C. Watkins, "Genesis Platform," dans *Digital Avionics Handbook*, C. R. Spitzer, Éd., vol. 2: Taylor & Francis Group LLC, 2007, pp. 12-1 - 12-28.

[3] J. Rufino, S. Filipe, M. Coutinho, S. Santos, et J. Windsor, "ARINC 653 Interface in RTEMS," dans *Proceedings of the DASIA 2007: Data Systems In Aerospace Conference, Naples, Italie*, 2007.

[4] S. Santos, J. Rufino, T. Schoofs, C. Tatibana, et J. Windsor, "A portable ARINC 653 standard interface," dans *Proceedings of the 27th Digital Avionics Systems Conference (DASC), St-Paul, Minnesota*, 2008, pp. 1.E.2-1 - 1.E.2-7.

[5] T. Schoofs, S. Santos, C. Tatibana, et J. Anjos, "An Integrated Modular Avionics Development Environment," dans *Proceedings of the 28th Digital Avionics Systems Conference (DASC), Lisbonne, Portugal*, 2009, pp. 1.A.2-1 - 1.A.2-9.

[6] Aeronautical Radio Inc., "Avionics Application Software Standard Interface - Part 1 - Required Services," Airlines Electronic Engineering Committee, ARINC characteristic, 653P1-3, 2010.

[7] Aeronautical Radio Inc., "Avionics Application Software Standard Interface - Part 2 - Extended Services," Airlines Electronic Engineering Committee, ARINC characteristic, 653P2-1, 2008.

[8] Aeronautical Radio Inc., "Avionics Application Software Standard Interface - Part 3 - Conformity Test Specification," Airlines Electronic Engineering Committee, ARINC characteristic, 653P3, 2006.

[9] Aeronautical Radio Inc., "Avionics Application Software Standard Interface - Part 4 - Subset Services," Airlines Electronic Engineering Committee, ARINC characteristic, 653P4, 2012.

[10] Wind River, "Wind River VxWorks 653 Platform," *Wind River*. [En ligne]. Disponible: http://www.windriver.com/products/platforms/safety_critical_arinc_653/. [Consulté le 5 novembre 2012].

[11] Sysgo Embedding Innovations, "PikeOS RTOS Technology," *Sysgo Embedding Innovations*. [En ligne]. Disponible: http://www.sysgo.com/en/products/pikeos-rtos-and-virtualization-concept/rtos-technology/. [Consulté le 29 mai 2012].

[12] Green Hills Software, "Safety Critical Products: Integrity-178B RTOS," *Green Hills Software*. [En ligne]. Disponible: http://www.ghs.com/products/safety_critical/integrity-do-178b.html. [Consulté le 29 mai 2012].

[13] J. Delange, J. Hugues, et P. Dissaux, "Validate implementation correctness using simulation: the TASTE approach," dans *Proceedings of the Embedded Real Time Software and Systems Conference (ERTS), Toulouse, France*, 2012.

[14] P. H. Feiler, D. P. Gluch, et J. J. Hudak, "The Architecture Analysis & Design Language (AADL): An Introduction," Université Carnegie Mellon, Rapport technique CMU/SEI-2006-TN-011, 2006. [En ligne]. Disponible: http://www.sei.cmu.edu/library/abstracts/reports/06tn011.cfm. [Consulté le 29 mai 2012].

[15] GMV IMA Research Group, "SIMA Command Line Tools Application Development and Configuration Guide," *GMV*, version 0.9.2.5, 2009-2010.

[16] GMV, "Home," *GMV*. [En ligne]. Disponible: http://www.gmv.com/en/index.html. [Consulté le 29 mai 2012].

[17] G. Lasnier, B. Zalila, L. Pautet, et J. Hugues, "OCARINA: An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications," dans *Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies, Brest, France*, 2009, pp. 237-250.

[18] Aeronautical Radio Inc., "Multi-purpose Control and Display Unit," Airlines Electronic Engineering Committee, ARINC characteristic, 739A-1, 1998.

[19] CMC Électronique, "CMC Électronique à l'avant-plan," *Esterline*. [En ligne]. Disponible: http://www.esterline.com/avionicssystems/fr-ca/overview.aspx. [Consulté le 20 juin 2012].

[20] École de Technologie Supérieure, "AREXIMAS - ARchitectural EXploration in Integrated Modular Avionics Systems," *École de Technologie Supérieure*. [En ligne]. Disponible: http://areximas.etsmtl.ca/. [Consulté le 26 juin 2012].

[21] T. Carmel-Veilleux, "Adaptation multicoeur d'un noyau de partitionnement robuste vers l'architecture PowerPC," M. Sc. A., École de Technologie Supérieure, Montréal, Québec, Canada, 2011.

[22] P. H. Feiler et D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*, Boston: Addison-Wesley Professional, 2012.

[23] Á. Horváth, D. Varró, et T. Schoofs, "Model-driven development of ARINC 653 configuration tables," dans *Proceedings of the 29th Digital Avionics Systems Conference (DASC), Salt Lake City, Utah*, 2010, pp. 5.A.5-1 - 5.A.5-116.

[24] J. Delange, L. Pautet, A. Plantec, M. Kerboeuf, F. Singhoff, et F. Kordon, "Validate, simulate, and implement ARINC653 systems using the AADL," dans *Proceedings of the ACM SIGAda annual international conference on Ada and related technologies, Tampa Bay, Floride*, 2009, pp. 31-44.

[25] J. Delange, C. Honvault, et J. Windsor, "Model-based Engineering Approach for System Architecture Exploration," dans *Proceedings of the Embedded Real Time Software and Systems Conference (ERTS), Toulouse, France*, 2012.

[26] A. Khoroshilov, A. Petrenko, I. Koverninskiy, et A. Ugnenko, "Integrating AADL based toolchain into existing industrial processes," dans *Proceedings of the 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), Las Vegas, Nevada*, 2011, pp. 367-371.

[27] M. Perrotin, E. Conquet, J. Delange, et T. Tsiodras, "TASTE - An open-source tool-chain for embedded system and software development," dans *Proceedings of the Embedded Real Time Software and Systems Conference (ERTS), Toulouse, France*, 2012.

[28] J. Seronie-Vivien et C. Cantenot, "RTEMS operating system qualification," dans *Proceedings of the DASIA 2005: Data Systems In Aerospace Conference, Édimbourg, Écosse*, 2005.

[29] J. Rufino, J. Craveiro, T. Schoofs, C. Tatibana, et J. Windsor, "AIR Technology: a step towards ARINC 653 in space," dans *Proceedings of the DASIA 2009: Data Systems In Aerospace Conference, Istanbul, Turquie*, 2009.

[30] J. Craveiro, "Integration of generic operating systems in partitioned architectures," M. Sc., Université de Lisbonne, Lisbonne, Portugal, 2009. [En ligne]. Disponible: http://air.di.fc.ul.pt/air-ii/downloads/Craveiro09MSc.pdf. [Consulté le 9 mai 2012].

[31] T. Schoofs, "AIR - Overview," Lisbonne, Portugal, Présentation technique, 2009-2011.

[32] J. Craveiro, J. Rufino, C. Almeida, R. Covelo, et P. Venda, "Embedded Linux in a partitioned architecture for aerospace applications," dans *Proceedings of the 7th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA), Rabat, Maroc*, 2009, pp. 132-138.

[33] A. Crespo, I. Ripoll, M. Masmano, P. Arberet, et J. J. Metge, "XtratuM: An open source hypervisor for TSP embedded systems in aerospace," dans *Proceedings of the DASIA 2009: Data Systems In Aerospace Conference, Istanbul, Turquie*, 2009.

[34] On-Line Applications Research (OAR) Corporation, "Home," *RTEMS Operating System - Real-Time and Real-Free*, 2012. [En ligne]. Disponible: http://www.rtems.com/. [Consulté le 20 juin 2012].

[35] On-Line Applications Research (OAR) Corporation, "RTEMS Google Summer of Code 2012 Projects," *RTEMS Operating System - Real-Time and Real-Free*, 2012. [En ligne]. Disponible: http://www.rtems.com/node/96. [Consulté le 20 juin 2012].

[36] On-Line Applications Research (OAR) Corporation, "License," *RTEMS Operating System - Real-Time and Real-Free*, 2012. [En ligne]. Disponible: http://www.rtems.com/license/LICENSE. [Consulté le 20 juin 2012].

[37] On-Line Applications Research (OAR) Corporation, "RTEMS C User's Guide," *On-Line Applications Research (OAR) Corporation*, Manuel - Édition 4.10.2 pour RTEMS 4.10.2, 2011. [En ligne]. Disponible: http://www.rtems.org/onlinedocs/releases/rtemsdocs-4.10.2/share/rtems/pdf/c_user.pdf. [Consulté le 9 mai 2012].

[38] M. Masmano, Y. Valiente, P. Balbastre, I. Ripoll, et A. Crespo, "LithOS: a ARINC-653 guest operating for XtratuM," dans *12th Real-Time Linux Workshop (RTLWS), Nairobi, Kenya*, 2010. [En ligne]. Disponible:

https://http://www.osadl.org/fileadmin/dam/rtlws/12/Crespo.pdf. [Consulté le 21 juin 2012].

[39] M. Masmano, Y. Valiente, P. Balbastre, I. Ripoll, et A. Crespo, "LithOS: A ARINC-653 guest operating for XtratuM," *Université Polytechnique de Valence*, 2010. [En ligne]. Disponible: http://polimedia.upv.es/visor/?id=c2d897f7-1084-774f-a474-a52dbfd01464. [Consulté le 21 juin 2012].

[40] Groupe sur les systèmes temps-réel de l'institut d'automatisation et d'informatique industrielle de l'Université Polytechnique de Valence, "XtratuM - Legal issues," *Université Polytechnique de Valence*, 2011. [En ligne]. Disponible: http://www.xtratum.org/main/legal. [Consulté le 21 juin 2012].

[41] J. Delange et L. Lec, "POK, an ARINC 653-compliant operating system released under the BSD license," dans *13th Real-Time Linux Workshop (RTLWS), Faculté de Génie Électrique de l'Université Technique Tchèque de Prague*, 2011. [En ligne]. Disponible: https://lwn.net/images/conf/rtlws-2011/proc/Delange_POK.pdf. [Consulté le 21 juin 2012].

[42] Équipe de POK, "POK User Guide," *Équipe de POK*, Version de février 2011, 2011. [En ligne]. Disponible: http://download.tuxfamily.org/pok/snapshots/. [Consulté le 6 février 2012].

[43] Télécom ParisTech, "École d'ingénieurs - Télécom ParisTech," *Télécom ParisTech*. [En ligne]. Disponible: http://www.telecom-paristech.fr. [Consulté le 21 juin 2011].

[44] Laboratoire d'informatique de Paris 6, "Laboratoire d'informatique de Paris 6 - Accueil," *Université Pierre et Marie Curie*. [En ligne]. Disponible: http://www.lip6.fr. [Consulté le 21 juin 2011].

[45] École pour l'informatique et les techniques avancées, "EPITA - École d'ingénieur en informatique," *École pour l'informatique et les techniques avancées*. [En ligne]. Disponible: http://www.epita.fr. [Consulté le 21 juin 2011].

[46] J. Delange, "Discussion électronique: Fixed available memory for QUEUE," *POK Developper Mailing List*, 5-6 février 2012. [En ligne]. Disponible: http://listengine.tuxfamily.org/lists.tuxfamily.org/pok-devel/2012/02/msg00009.html. [Consulté le 6 février 2012].

[47] POK, "About POK licence," *POK, a partitionned operating system*, 2008-2009. [En ligne]. Disponible: http://pok.safety-critical.net/legal. [Consulté le 21 juin 2012].

[48] J. Delange, L. Lec, et P. Ficheux, "Discussion électronique: Question about POK's supported platform," *POK Developper Mailing List*, 19-20 décembre 2011. [En ligne]. Disponible: http://listengine.tuxfamily.org/lists.tuxfamily.org/pok-devel/2011/12/threads.html. [Consulté le 20 décembre 2011].

[49] J. Delange, "Discussion électronique: Question about POK roadmap," *POK Developper Mailing List*, 23-24 février 2012. [En ligne]. Disponible: http://listengine.tuxfamily.org/lists.tuxfamily.org/pok-devel/2012/02/msg00042.html. [Consulté le 24 février 2012].

[50] GMV, "Skysoft portugal – General Information," *GMV*. [En ligne]. Disponible: http://www.epicos.com/epicos/extended/portugal/skysoft/skysoft_home.html. [Consulté le 7 septembre 2012].

[51] CONFIG_PREEMPT_RT community, "Real-Time Linux Wiki," *CONFIG_PREEMPT_RT community*. [En ligne]. Disponible: https://rt.wiki.kernel.org/index.php/Main_Page. [Consulté le 10 septembre 2012].

[52] GMV - Skysoft, "Use of SIMA in the DIANA Project - White Paper," *GMV*, Lisbonne, Portugal, version 0.7, 2010.

[53] SAE AS-2C Architecture Description Language Subcommittee, Embedded Computing Systems Committee, Aerospace Avionics Systems Division, "(R) Architecture Analysis & Design Language (AADL)," SAE Aerospace - An SAE International Group, AS5506A, novembre 2004 - janvier 2009.

[54] SAE AS-2C Architecture Description Language Subcommittee, Embedded Computing Systems Committee, Aerospace Avionics Systems Division, "SAE Architecture Analysis and Design Language (AADL) Annex Volume 1," SAE Aerospace - An SAE International Group, AS5506/1, juin 2006 - avril 2011.

[55] SAE AS-2C Architecture Description Language Subcommittee, Embedded Computing Systems Committee, Aerospace Avionics Systems Division, "SAE Architecture Analysis and Design Language (AADL) Annex Volume 2," SAE Aerospace - An SAE International Group, AS5506/2, janvier 2011.

[56] S. Rubini, F. Singhoff, et J. Hugues, "Modeling and verification of memory architectures with AADL and REAL," dans *6th International Workshop on AADL and UML from the 16th IEEE International Conference on Engineering of Complex Computer Systems*, *Las Vegas, Nevada*, 2011, pp. 338-343.

[57] F. Singhoff, J. Legrand, L. Nana, et L. Marcé, "Cheddar: A flexible Real Time Scheduling Framework," dans *Proceedings of the ACM SigAda Annual International Conference, Atlanta, Georgie*, 2004, pp. 1-8.

[58] O. Sokolsky, I. Lee, et D. Clark, "Schedulability Analysis of AADL models," dans *Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium, Rhodes, Grèce*, 2006.

[59] TOPCASED, "TOPCASED 5.3.0," *TOPCASED - The Open-Source Toolkit for Critical Systems*. [En ligne]. Disponible: http://www.topcased.org/index.php/content/view/40/54/. [Consulté le 3 janvier 2012].

[60] TOPCASED, "Downloads for project ADELE," *TOPCASED - The Open-Source Toolkit for Critical Systems*. [En ligne]. Disponible: http://www.topcased.org/index.php?idd_projet_pere=73. [Consulté le 3 janvier 2012].

[61] J. Hugues, T. Vergnaud, et B. Zalila, "OCARINA, a compiler for the AADL," *École nationale supérieure des télécommunications*, Paris, France, Manuel usager, février 2012.

[62] M. Perrotin, T. Tsiodras, J. Delange, et J. Hugues, "TASTE - The Assert Set of Tools for Engineering," *Agence Spatiale Européenne*, Noordwijk, Pays-Bas, Documentation version 1.1, 27 mars 2012. [En ligne]. Disponible: http://download.tuxfamily.org/taste/snapshots/doc/taste-documentation-current.pdf. [Consulté le 9 mai 2012].

[63] MathWorks, "Simulink Coder - Generate C and C++ code from Simulink and Stateflow models," *The MathWorks Inc.*, 1994-2012. [En ligne]. Disponible: http://www.mathworks.com/products/simulink-coder. [Consulté le 18 juin 2012].

[64] MathWorks, "Stateflow - Design and simulate state charts," *The MathWorks Inc.*, 1994-2012. [En ligne]. Disponible: http://www.mathworks.com/products/stateflow/. [Consulté le 18 juin 2012].

[65] PragmaDev, "PragmaDev - Real Time modeling and testing tools," *PragmaDev*. [En ligne]. Disponible: http://www.pragmadev.com/index.html. [Consulté le 18 juin 2012].

[66] PragmaDev et TASTE, "Real Time Developper Studio (RTDS) tool for verification of properties and automatic code generation," *PragmaDev et l'Agence Spatiale Européenne (ESA)*, Paris, France, Acétates de présentation, 2010. [En ligne]. Disponible: http://www.pragmadev.com/downloads/Taste.pdf. [Consulté le 18 juin 2012].

[67] L. Bao, "Rapport de stage d'ingénieur III: Système d'exploitation temps-réel pour des applications en avionique," *École supérieure d'ingénieurs en génie électrique (ESIGELEC) et École Polytechnique de Montréal*, Montréal, Qc, Canada, 2012.

[68] Voyage Design and Consultants, "About Voyage Linux," *Voyage Linux – x86 Embedded Linux is Green Computing*. [En ligne]. Disponible: http://linux.voyage.hk/. [Consulté le 15 novembre 2011].

[69] QEMU - Open Source Processor Emulator, "About QEMU," *QEMU*. [En ligne]. Disponible: http://wiki.qemu.org/Main_Page. [Consulté le 15 novembre 2011].

[70] Real-Time Linux Wiki, "Cyclictest," *Cyclictest – RT Wiki*. [En ligne]. Disponible: https://rt.wiki.kernel.org/articles/c/y/c/Cyclictest.html [Consulté le 21 octobre 2012].

[71] M. Desnoyers et M. Dagenais, "The LTTng Tracer: A low impact performance and behaviour monitor for GNU/Linux," dans *Proceedings of the Linux Symposium, Ottawa, Ontario, Canada*, vol. 1, 19-22 juillet 2006, pp. 209-224.

ANNEXE 1 – Diagrammes UML du système de MCDU

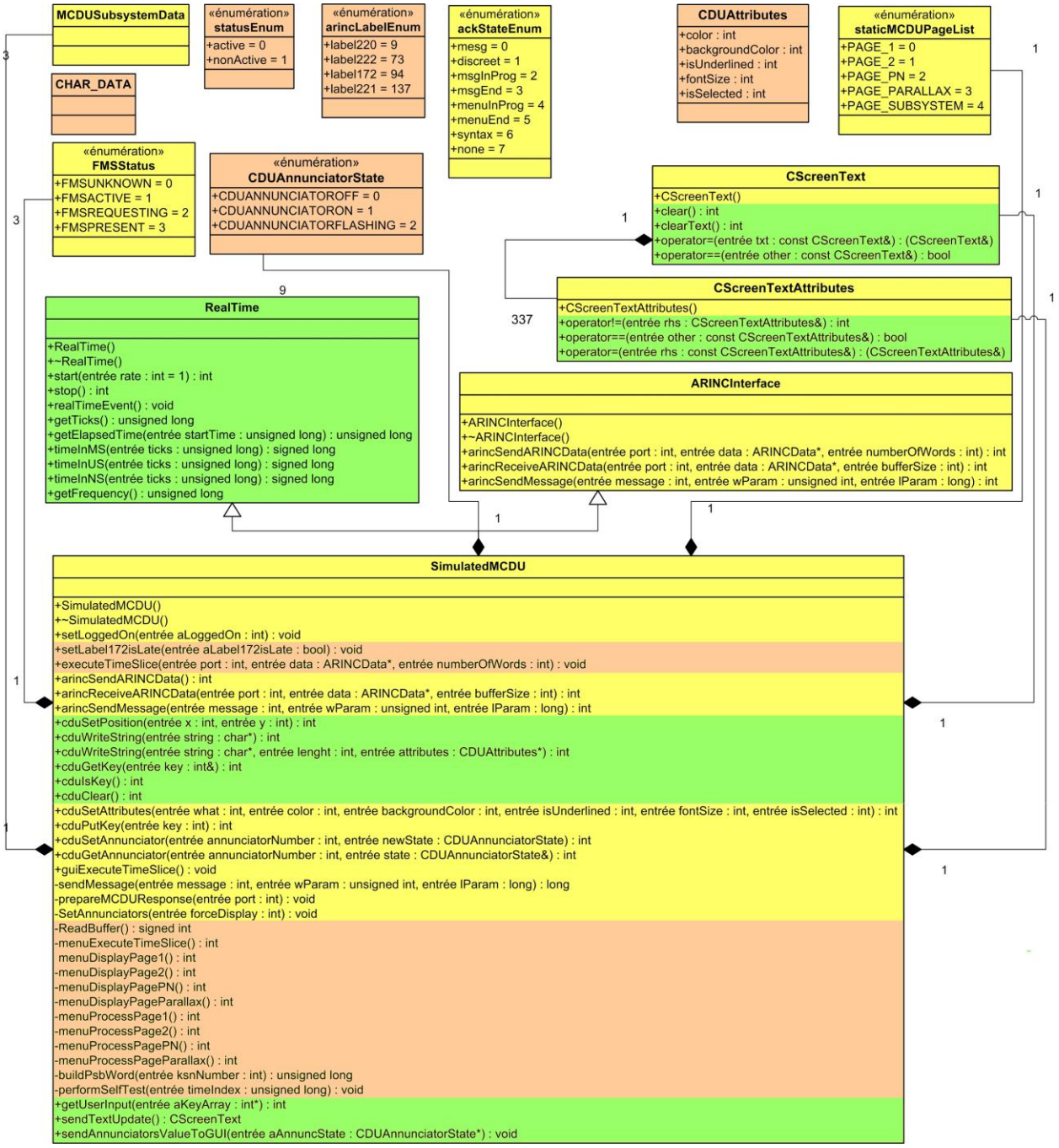

Figure A-1 : Diagramme de classe de l'engin MCDU

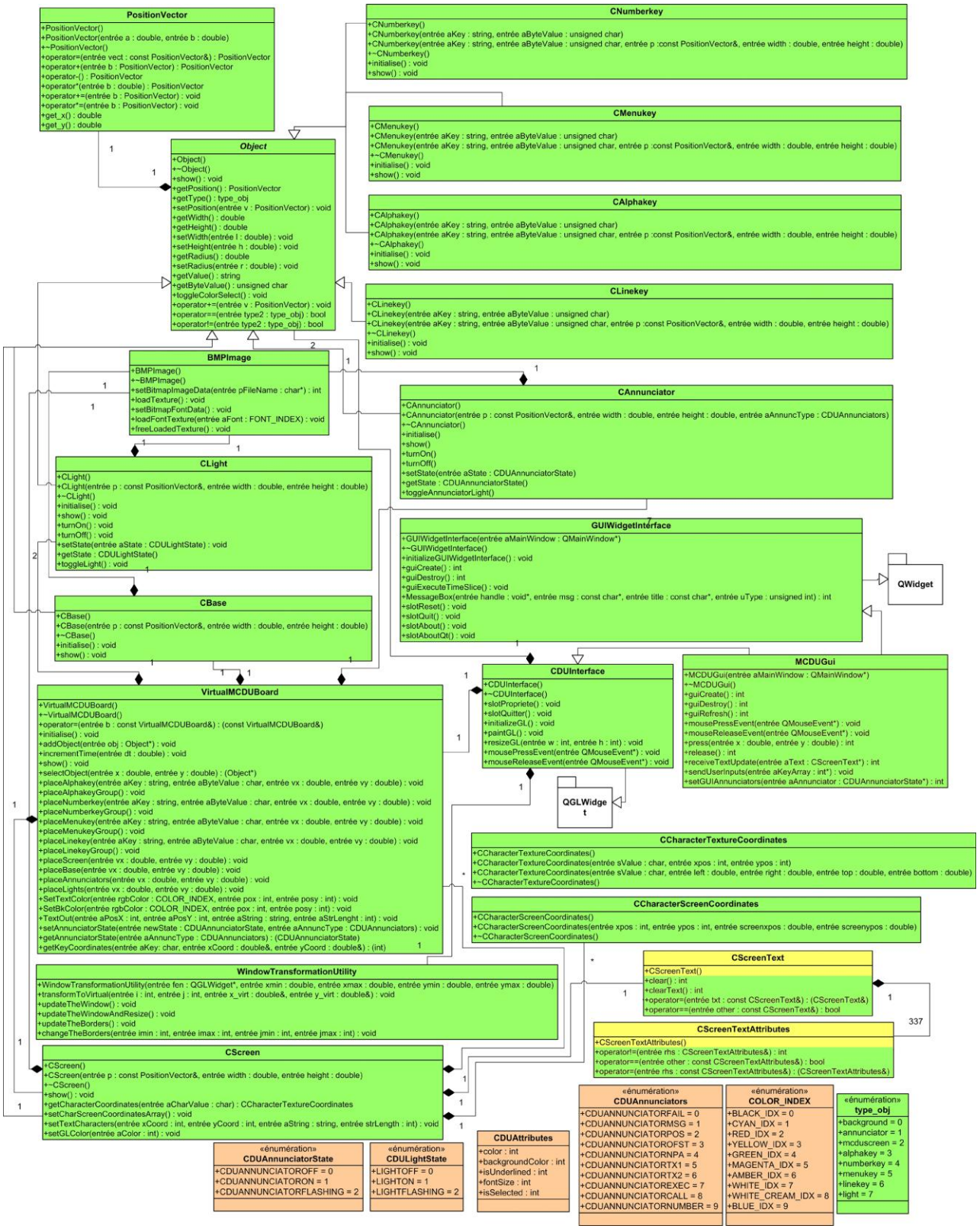

Figure A-2 : Diagramme de classe de l'engin MCDU

ANNEXE 2 – Configuration XML du système de MCDU

```
<?xml version="1.0" encoding="UTF-8"?>
<ARINC_653_Module xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=".Clearwater Version 1.1 .xsd"
ModuleName="Example ARINC 653 XML Instance" ModuleVersion="18-Mar-2003">
  <System_HM_Table>
    <System_State_Entry SystemState="1" Description="Module Init">
        <Error_ID_Level ErrorIdentifier="2"
                    Description="Module Config Error" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="4"
                    Description="partition init error" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="5"
                    Description="segmentation error" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="7"
                    Description="invalid OS call" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="8"
                    Description="divide by 0" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="9"
                    Description="floating point error" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="11"
                    Description="bad opcode" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="12"
                    Description="power failure" ErrorLevel="MODULE"/>
    </System_State_Entry>

    <System_State_Entry SystemState="2" Description="System Function
Execution">
        <Error_ID_Level ErrorIdentifier="5"
                    Description="Segmentation Violation"
ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="6"
                    Description="time duration exceeded"
ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="7"
                    Description="invalid OS call" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="8"
                    Description="divide by 0" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="9"
                    Description="floating point error" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="11"
                    Description="floating point error" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="12"
                    Description="floating point error" ErrorLevel="MODULE"/>
    </System_State_Entry>

    <System_State_Entry SystemState="4" Description="Partition Init">
        <Error_ID_Level ErrorIdentifier="3"
                    Description="partition config error"
ErrorLevel="PARTITION"/>
        <Error_ID_Level ErrorIdentifier="5"
                    Description="segmenation violation"
ErrorLevel="PARTITION"/>
```

```
        <Error_ID_Level ErrorIdentifier="6"
                        Description="time duration exceeded"
ErrorLevel="PARTITION"/>
        <Error_ID_Level ErrorIdentifier="7"
                        Description="invalid OS call" ErrorLevel="PARTITION"/>
        <Error_ID_Level ErrorIdentifier="8"
                        Description="divide by 0" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="9"
                        Description="floating point error" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="11"
                        Description="floating point error" ErrorLevel="MODULE"/>
        <Error_ID_Level ErrorIdentifier="12"
                        Description="floating point error" ErrorLevel="MODULE"/>
    </System_State_Entry>

    <System_State_Entry SystemState="5" Description="Application code is
execution">
      <Error_ID_Level ErrorIdentifier="5"
                      Description="segmentation error"
PartitionRecoveryAction="WARM_RESTART" ErrorCode="MEMORY_VIOLATION"
ErrorLevel="PROCESS"/>
      <Error_ID_Level ErrorIdentifier="6"
                      Description="time duration exceeded"
PartitionRecoveryAction="WARM_RESTART" ErrorCode="DEADLINE_MISSED"
ErrorLevel="PROCESS"/>
      <Error_ID_Level ErrorIdentifier="7"
                      Description="invalid OS call"
PartitionRecoveryAction="WARM_RESTART" ErrorCode="ILLEGAL_REQUEST"
ErrorLevel="PROCESS"/>
      <Error_ID_Level ErrorIdentifier="8"
                      Description="divide by 0"
PartitionRecoveryAction="WARM_RESTART" ErrorCode="NUMERIC_ERROR"
ErrorLevel="PROCESS"/>
      <Error_ID_Level ErrorIdentifier="9"
                      Description="overflow"
PartitionRecoveryAction="WARM_RESTART" ErrorCode="STACK_OVERFLOW"
ErrorLevel="PROCESS"/>
      <Error_ID_Level ErrorIdentifier="10"
                      Description="application error"
PartitionRecoveryAction="WARM_RESTART" ErrorCode="APPLICATION_ERROR"
ErrorLevel="PROCESS"/>
      <Error_ID_Level ErrorIdentifier="11"
                      Description="bad opcode"
PartitionRecoveryAction="WARM_RESTART" ErrorCode="ILLEGAL_REQUEST"
ErrorLevel="PROCESS"/>
      <Error_ID_Level ErrorIdentifier="12"
                      Description="power failure"
PartitionRecoveryAction="WARM_RESTART" ErrorCode="POWER_FAIL"
ErrorLevel="PROCESS"/>
    </System_State_Entry>

    <System_State_Entry SystemState="6" Description="Partition Error Handler">
      <Error_ID_Level ErrorIdentifier="10"
                      Description="application error" ErrorLevel="PARTITION"/>
    </System_State_Entry>
```

```
    <System_State_Entry SystemState="9" Description="mos HM is executing">
      <Error_ID_Level ErrorIdentifier="5"
                      Description="segmentation error"
ErrorLevel="PARTITION"/>
      <Error_ID_Level ErrorIdentifier="6"
                      Description="time duration exceeded"
ErrorLevel="PARTITION"/>
      <Error_ID_Level ErrorIdentifier="7"
                      Description="invalid OS call" ErrorLevel="PARTITION"/>
      <Error_ID_Level ErrorIdentifier="8"
                      Description="divide by 0" ErrorLevel="PARTITION"/>
      <Error_ID_Level ErrorIdentifier="9"
                      Description="overflow" ErrorLevel="PARTITION"/>
      <Error_ID_Level ErrorIdentifier="11"
                    Description="floating point error"
ErrorLevel="PARTITION"/>
      <Error_ID_Level ErrorIdentifier="12"
                    Description="floating point error"
ErrorLevel="PARTITION"/>
    </System_State_Entry>
  </System_HM_Table>

  <Module_HM_Table ModuleCallback="module_HM_callback">
    <System_State_Entry SystemState="1" Description="module init">
      <Error_ID_Action ErrorIdentifier="2" Action="SHUTDOWN"/>
      <Error_ID_Action ErrorIdentifier="4" Action="SHUTDOWN"/>
      <Error_ID_Action ErrorIdentifier="5" Action="SHUTDOWN"/>
      <Error_ID_Action ErrorIdentifier="7" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="8" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="9" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="11" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="12" Action="RESET"/>
    </System_State_Entry>
    <System_State_Entry SystemState="2" Description="system function
execution">
      <Error_ID_Action ErrorIdentifier="5" Action="SHUTDOWN"/>
      <Error_ID_Action ErrorIdentifier="6" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="7" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="8" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="9" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="11" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="12" Action="RESET"/>
    </System_State_Entry>
    <System_State_Entry SystemState="4" Description="partition init">
      <Error_ID_Action ErrorIdentifier="8" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="9" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="11" Action="RESET"/>
      <Error_ID_Action ErrorIdentifier="12" Action="RESET"/>
    </System_State_Entry>
  </Module_HM_Table>

  <Partition PartitionIdentifier="1"
             PartitionName="mcduapp"
          Criticality="LEVEL_A"
```

```
        SystemPartition="true" EntryPoint="Initial">
  <Queuing_Port Name="OUT_ANNUNC"
                MaxMessageSize="54"
      Direction="SOURCE"
      MaxNbMessages="24"/>
  <Queuing_Port Name="OUT_DISPLAY"
                MaxMessageSize="8192"
      Direction="SOURCE"
      MaxNbMessages="20"/>
  <Queuing_Port Name="IN_USERINPUT"
                MaxMessageSize="128"
      Direction="DESTINATION"
      MaxNbMessages="32"/>

  <Queuing_Port Name="IN_ARINC_SEND"
                MaxMessageSize="1024"
      Direction="DESTINATION"
      MaxNbMessages="50"/>
  <Queuing_Port Name="OUT_ARINC_REC"
                MaxMessageSize="1024"
      Direction="SOURCE"
      MaxNbMessages="50"/>
</Partition>

<Partition PartitionIdentifier="2"
           PartitionName="mcduGUICOM"
        Criticality="LEVEL_A"
        SystemPartition="false" EntryPoint="Initial">
  <Queuing_Port Name="IN_ANNUNC"
                MaxMessageSize="54"
      Direction="DESTINATION"
      MaxNbMessages="24"/>
  <Queuing_Port Name="IN_DISPLAY"
                MaxMessageSize="8192"
      Direction="DESTINATION"
      MaxNbMessages="20"/>
  <Queuing_Port Name="OUT_USERINPUT"
                MaxMessageSize="128"
      Direction="SOURCE"
      MaxNbMessages="32"/>
</Partition>

<Partition PartitionIdentifier="3"
           PartitionName="mcduCOM"
        Criticality="LEVEL_A"
        SystemPartition="false" EntryPoint="Initial">
  <Queuing_Port Name="IN_ARINC_REC"
                MaxMessageSize="1024"
      Direction="DESTINATION"
      MaxNbMessages="50"/>
  <Queuing_Port Name="OUT_ARINC_SEND"
                MaxMessageSize="1024"
      Direction="SOURCE"
      MaxNbMessages="50"/>
</Partition>
```

```
  <Partition_Memory PartitionIdentifier="1" PartitionName="mcduapp">
    <Memory_Requirements Type="CODE" SizeBytes="92160" Access="READ_WRITE"/>
    <Memory_Requirements Type="DATA" SizeBytes="92160" Access="READ_WRITE"/>
    <Memory_Requirements Type="INPUT_OUTPUT" SizeBytes="128000"
                         PhysicalAddress="D0000000" Access="READ_WRITE"/>
  </Partition_Memory>

  <Partition_Memory PartitionIdentifier="2" PartitionName="mcduGUICOM">
    <Memory_Requirements Type="CODE" SizeBytes="314572800"
Access="READ_WRITE"/>
    <Memory_Requirements Type="DATA" SizeBytes="314572800"
Access="READ_WRITE"/>
    <Memory_Requirements Type="INPUT_OUTPUT" SizeBytes="128000"
                         PhysicalAddress="E0000000" Access="READ_WRITE"/>
  </Partition_Memory>

  <Partition_Memory PartitionIdentifier="3" PartitionName="mcduCOM">
    <Memory_Requirements Type="CODE" SizeBytes="314572800"
Access="READ_WRITE"/>
    <Memory_Requirements Type="DATA" SizeBytes="314572800"
Access="READ_WRITE"/>
    <Memory_Requirements Type="INPUT_OUTPUT" SizeBytes="128000"
                         PhysicalAddress="E0000000" Access="READ_WRITE"/>
  </Partition_Memory>


  <Module_Schedule MajorFrameSeconds="1.000">
    <Partition_Schedule PartitionIdentifier="1"
                        PartitionName="mcduapp"
            PeriodSeconds="1.000"
            PeriodDurationSeconds="0.300">
      <Window_Schedule WindowIdentifier="101"
                       WindowStartSeconds="0.700"
             WindowDurationSeconds="0.300"
             PartitionPeriodStart="true"/>
    </Partition_Schedule>

    <Partition_Schedule PartitionIdentifier="2"
                        PartitionName="mcduGUICOM"
            PeriodSeconds="1.000"
            PeriodDurationSeconds="0.350">
      <Window_Schedule WindowIdentifier="201"
                       WindowStartSeconds="0.000"
             WindowDurationSeconds="0.350"
             PartitionPeriodStart="true"/>
    </Partition_Schedule>

    <Partition_Schedule PartitionIdentifier="3"
                        PartitionName="mcduCOM"
            PeriodSeconds="1.000"
            PeriodDurationSeconds="0.350">
      <Window_Schedule WindowIdentifier="301"
                       WindowStartSeconds="0.350"
```

```
            WindowDurationSeconds="0.350"
            PartitionPeriodStart="true"/>
    </Partition_Schedule>

  </Module_Schedule>

  <Partition_HM_Table PartitionIdentifier="1"
                      PartitionName="mcduapp"
           PartitionCallback="partition_HM_callback">
    <System_State_Entry SystemState="4" Description="partition init">
      <Error_ID_Action ErrorIdentifier="3" Description="partition config
error" Action="IDLE"/>
      <Error_ID_Action ErrorIdentifier="5" Description="segmentation error"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="6" Description="time duration
exceeded" Action="IDLE"/>
      <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call"
Action="WARM_START"/>
    </System_State_Entry>

    <System_State_Entry SystemState="6" Description="partition error handler">
      <Error_ID_Action ErrorIdentifier="10" Description="application error"
Action="WARM_START"/>
    </System_State_Entry>
    <System_State_Entry SystemState="9" Description="mos hm is executing">
      <Error_ID_Action ErrorIdentifier="5" Description="segmentation error"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="6" Description="time duration
exceeded" Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="8" Description="divide by 0"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="9" Description="overflow"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="11" Description="bad opcode"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="12" Description="power failure"
Action="WARM_START"/>
    </System_State_Entry>
  </Partition_HM_Table>


  <Partition_HM_Table PartitionIdentifier="2"
                      PartitionName="mcduGUICOM"
           PartitionCallback="partition_HM_callback">
    <System_State_Entry SystemState="4" Description="partition init">
      <Error_ID_Action ErrorIdentifier="3" Description="partition config
error" Action="IDLE"/>
      <Error_ID_Action ErrorIdentifier="5" Description="segmentation error"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="6" Description="time duration
exceeded" Action="IDLE"/>
      <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call"
Action="WARM_START"/>
```

```
    </System_State_Entry>
    <System_State_Entry SystemState="6" Description="partition error handler">
      <Error_ID_Action ErrorIdentifier="10" Description="application error"
Action="WARM_START"/>
    </System_State_Entry>
    <System_State_Entry SystemState="9" Description="mos hm is executing">
      <Error_ID_Action ErrorIdentifier="5" Description="segmentation error"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="6" Description="time duration
exceeded" Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="8" Description="divide by 0"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="9" Description="overflow"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="11" Description="bad opcode"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="12" Description="power failure"
Action="WARM_START"/>
    </System_State_Entry>
  </Partition_HM_Table>


  <Partition_HM_Table PartitionIdentifier="3"
                      PartitionName="mcduCOM"
            PartitionCallback="partition_HM_callback">
    <System_State_Entry SystemState="4" Description="partition init">
      <Error_ID_Action ErrorIdentifier="3" Description="partition config
error" Action="IDLE"/>
      <Error_ID_Action ErrorIdentifier="5" Description="segmentation error"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="6" Description="time duration
exceeded" Action="IDLE"/>
      <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call"
Action="WARM_START"/>
    </System_State_Entry>

    <System_State_Entry SystemState="6" Description="partition error handler">
      <Error_ID_Action ErrorIdentifier="10" Description="application error"
Action="WARM_START"/>
    </System_State_Entry>
    <System_State_Entry SystemState="9" Description="mos hm is executing">
      <Error_ID_Action ErrorIdentifier="5" Description="segmentation error"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="6" Description="time duration
exceeded" Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="7" Description="invalid OS call"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="8" Description="divide by 0"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="9" Description="overflow"
Action="WARM_START"/>
      <Error_ID_Action ErrorIdentifier="11" Description="bad opcode"
Action="WARM_START"/>
```

```
      <Error_ID_Action ErrorIdentifier="12" Description="power failure"
Action="WARM_START"/>
    </System_State_Entry>
  </Partition_HM_Table>


  <Connection_Table>
    <Channel ChannelIdentifier="1" ChannelName="Annunciators_Infos">
      <Source>
        <Standard_Partition PartitionIdentifier="1"
                            PartitionName="mcduapp" PortName="OUT_ANNUNC"/>
      </Source>
      <Destination>
        <Standard_Partition PartitionIdentifier="2"
                            PartitionName="mcduGUICOM" PortName="IN_ANNUNC"/>
      </Destination>
    </Channel>
    <Channel ChannelIdentifier="2" ChannelName="Display_Infos">
      <Source>
        <Standard_Partition PartitionIdentifier="1"
                            PartitionName="mcduapp" PortName="OUT_DISPLAY"/>
      </Source>
      <Destination>
        <Standard_Partition PartitionIdentifier="2"
                            PartitionName="mcduGUICOM" PortName="IN_DISPLAY"/>
      </Destination>
    </Channel>
    <Channel ChannelIdentifier="3" ChannelName="User_Input_Infos">
      <Source>
        <Standard_Partition PartitionIdentifier="2"
                            PartitionName="mcduGUICOM" PortName="OUT_USERINPUT"/>
      </Source>
      <Destination>
        <Standard_Partition PartitionIdentifier="1"
                            PartitionName="mcduapp" PortName="IN_USERINPUT"/>
      </Destination>
    </Channel>

    <Channel ChannelIdentifier="4" ChannelName="ARINC_Send">
      <Source>
        <Standard_Partition PartitionIdentifier="3"
                            PartitionName="mcduCOM" PortName="OUT_ARINC_SEND"/>
      </Source>
      <Destination>
        <Standard_Partition PartitionIdentifier="1"
                            PartitionName="mcduapp" PortName="IN_ARINC_SEND"/>
      </Destination>
    </Channel>
    <Channel ChannelIdentifier="5" ChannelName="ARINC_Rec">
      <Source>
        <Standard_Partition PartitionIdentifier="1"
                            PartitionName="mcduapp" PortName="OUT_ARINC_REC"/>
      </Source>
      <Destination>
        <Standard_Partition PartitionIdentifier="3"
```

```
                        PartitionName="mcduCOM" PortName="IN_ARINC_REC"/>
      </Destination>
    </Channel>
  </Connection_Table>

</ARINC_653_Module>
```

ANNEXE 3 – Modèle AADL textuel du système de MCDU

```
--
--                          MCDU System - AADL Model
--


package mcdu

public

with Data_Model;
with ARINC653;

data mydatabyte
properties
   Data_Model::Data_Representation => Integer;
   Data_Model::Number_Representation => Unsigned;
   Source_Data_Size => 1 Bytes;
end mydatabyte;

data CScreenText
properties
   Data_Model::Data_Representation => Array;
   Data_Model::Base_Type => (classifier (mcdu::mydatabyte));
   Data_Model::Dimension => (7076);
end CScreenText;

data annunciatorArray
properties
   Data_Model::Data_Representation => Array;
   Data_Model::Base_Type => (classifier (mcdu::mydatabyte));
   Data_Model::Dimension => (36);
end annunciatorArray;

data userInputsArray
properties
   Data_Model::Data_Representation => Array;
   Data_Model::Base_Type => (classifier (mcdu::mydatabyte));
   Data_Model::Dimension => (128);
end userInputsArray;

data arincMessageTransfer
properties
   Data_Model::Data_Representation => Array;
   Data_Model::Base_Type => (classifier (mcdu::mydatabyte));
   Data_Model::Dimension => (1024);
end arincMessageTransfer;


data integer
properties
  Data_Model::Data_Representation => integer;
```

```
  Source_Data_Size => 8 Bytes;
end integer;


virtual processor partition
end partition;

subprogram com_subprog
features
   inarincrec : in parameter arincMessageTransfer;
   outarincsend : out parameter arincMessageTransfer;
properties
   source_name => "dataexchangewithfmstask";
   source_language => C;
   source_text => ("../../../mcdusourcecode/mcduCOM/mcduCOM.o");
end com_subprog;

subprogram initcom_subprog
properties
   source_name => "init_Communications";
   source_language => C;
   source_text => ("../../../mcdusourcecode/mcduCOM/mcduCOM.o");
end initcom_subprog;

subprogram appstart_subprog
properties
   source_text => ("../../../mcdusourcecode/mcduapp/mcdu.o");
   source_language => C;
   source_name => "mcduengine_start";
end appstart_subprog;

subprogram app_subprog
features
   inarincsend : in parameter arincMessageTransfer;
   inuserinputs : in parameter userInputsArray;
   outarincrec : out parameter arincMessageTransfer;
   outdisplay : out parameter CScreenText;
   outannunc : out parameter annunciatorArray;
properties
   source_text => ("../../../mcdusourcecode/mcduapp/mcdu.o");
   source_language => C;
   source_name => "mcduengineinout";
end app_subprog;



subprogram initguicom_subprog
properties
   source_name => "initMcduGuiCom";
   source_language => C;
   source_text => ("../../../mcdusourcecode/mcduGUI/mcdugui.o");
end initguicom_subprog;

subprogram gui_comm_subprog
features
```

```
   indisplay : in parameter CScreenText;
   inannunc : in parameter annunciatorArray;
   outuserinputs : out parameter userInputsArray;
properties
   source_name => "commwithmcduapp";
   source_language => C;
   source_text => ("../../../mcdusourcecode/mcduGUI/mcdugui.o");
end gui_comm_subprog;

virtual processor implementation partition.com
end partition.com;

virtual processor implementation partition.app
end partition.app;

virtual processor implementation partition.gui
end partition.gui;


processor mcdusys
end mcdusys;

processor implementation mcdusys.impl
subcomponents
   partition_com : virtual processor partition.com;
   partition_app : virtual processor partition.app;
   partition_gui : virtual processor partition.gui;
properties
   ARINC653::Module_Major_Frame => 1000ms;
   ARINC653::Partition_Slots => (350ms, 350ms, 300ms);
   ARINC653::Slots_Allocation => ( reference (partition_gui), reference
(partition_com), reference (partition_app) );
end mcdusys.impl;

thread com_thread
features
   t_out_arinc_send : out event data port arincMessageTransfer
{ARINC653::Timeout => 85 ms;};
   t_in_arinc_rec : in event data port arincMessageTransfer {ARINC653::Timeout
=> 85 ms;};
properties
   Dispatch_Protocol => Periodic;
   Compute_Execution_Time => 0 ms .. 175 ms;
   Period => 500 Ms;
   Source_Stack_Size => 400000 bytes;
   Initialize_Entrypoint  => classifier (mcdu::initcom_subprog);
end com_thread;

thread app_thread
features
   t_in_arinc_send : in event data port arincMessageTransfer
{ARINC653::Timeout => 20 ms;};
   t_in_userinputs : in event data port userInputsArray {ARINC653::Timeout =>
20 ms;};
   t_out_arinc_rec : out event data port arincMessageTransfer
```

```
{ARINC653::Timeout => 20 ms;};
   t_out_display : out event data port CScreenText {ARINC653::Timeout => 20
ms;};
   t_out_annunc : out event data port annunciatorArray {ARINC653::Timeout =>
20 ms;};
properties
   Dispatch_Protocol => Periodic;
   Compute_Execution_Time => 0 ms .. 150 ms;
   Period => 500 Ms;
   Source_Stack_Size => 400000 bytes;
   Initialize_Entrypoint  => classifier (mcdu::appstart_subprog);
end app_thread;


thread gui_comm_thread
features
   t_in_display : in event data port CScreenText {ARINC653::Timeout => 29
ms;};
   t_in_annunc : in event data port annunciatorArray {ARINC653::Timeout => 29
ms;};
   t_out_userinputs : out event data port userInputsArray {ARINC653::Timeout
=> 29 ms;};
properties
   Dispatch_Protocol => Periodic;
   Compute_Execution_Time => 0 ms .. 175 ms;
   Period => 500 Ms;
   Source_Stack_Size => 400000 bytes;
   Initialize_Entrypoint  => classifier (mcdu::initguicom_subprog);
end gui_comm_thread;


thread implementation com_thread.impl
calls
   callsend : { comspg : subprogram com_subprog;};
connections
   parameter comspg.outarincsend -> t_out_arinc_send;
   parameter t_in_arinc_rec -> comspg.inarincrec;
end com_thread.impl;

thread implementation app_thread.impl
calls
   callsend : { appspg : subprogram app_subprog;};
connections
   parameter appspg.outarincrec -> t_out_arinc_rec;
   parameter appspg.outdisplay -> t_out_display;
   parameter appspg.outannunc -> t_out_annunc;
   parameter t_in_arinc_send -> appspg.inarincsend;
   parameter t_in_userinputs -> appspg.inuserinputs;
end app_thread.impl;


thread implementation gui_comm_thread.impl
calls
   callsend : { guicomspg : subprogram gui_comm_subprog;};
connections
```

```
   parameter t_in_display -> guicomspg.indisplay;
   parameter t_in_annunc -> guicomspg.inannunc;
   parameter guicomspg.outuserinputs -> t_out_userinputs;
end gui_comm_thread.impl;


process com_process
features
   out_arinc_send : out event data port arincMessageTransfer
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 50;
       Queue_Processing_Protocol => Fifo;
       };
   in_arinc_rec : in event data port arincMessageTransfer
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 50;
       Queue_Processing_Protocol => Fifo;
       };
end com_process;


process mcduapp_process
features
   in_arinc_send : in event data port arincMessageTransfer
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 50;
       Queue_Processing_Protocol => Fifo;
       };
   out_arinc_rec : out event data port arincMessageTransfer
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 50;
       Queue_Processing_Protocol => Fifo;
       };

   in_userinputs : in event data port userInputsArray
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 32;
       Queue_Processing_Protocol => Fifo;
       };
   out_display : out event data port CScreenText
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 20;
       Queue_Processing_Protocol => Fifo;
       };
   out_annunc : out event data port annunciatorArray
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 24;
       Queue_Processing_Protocol => Fifo;
       };

properties
   Source_Code_Size => 10 KByte;
   Source_Data_Size => 15 KByte;
end mcduapp_process;
```

```
process gui_process
features
   out_userinputs : out event data port userInputsArray
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 32;
       Queue_Processing_Protocol => Fifo;
       };
   in_display : in event data port CScreenText
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 20;
       Queue_Processing_Protocol => Fifo;
       };
   in_annunc : in event data port annunciatorArray
      {Overflow_Handling_Protocol => DropOldest;
       Queue_Size => 24;
       Queue_Processing_Protocol => Fifo;
       };

properties
   Source_Code_Size => 10 KByte;
   Source_Data_Size => 15 KByte;
end gui_process;


process implementation com_process.impl
subcomponents
   thr : thread com_thread.impl;
connections
   port thr.t_out_arinc_send -> out_arinc_send;
   port in_arinc_rec -> thr.t_in_arinc_rec;
end com_process.impl;


process implementation mcduapp_process.impl
subcomponents
   thr : thread app_thread.impl;
connections
   port thr.t_out_arinc_rec -> out_arinc_rec;
   port thr.t_out_display -> out_display;
   port thr.t_out_annunc -> out_annunc;
   port in_arinc_send -> thr.t_in_arinc_send;
   port in_userinputs -> thr.t_in_userinputs;
end mcduapp_process.impl;


process implementation gui_process.impl
subcomponents
   thr1 : thread gui_comm_thread.impl;
connections
   port in_display -> thr1.t_in_display;
   port in_annunc -> thr1.t_in_annunc;
   port thr1.t_out_userinputs -> out_userinputs;
end gui_process.impl;
```

```
system node
end node;

memory partitionmemory
properties
   Word_Count => 128000;
   ARINC653::Memory_Kind => memory_code;
   ARINC653::Access_Type => read_write;
end partitionmemory;

memory main_memory
end main_memory;

memory implementation main_memory.impl
subcomponents
   part1: memory partitionmemory;
   part2: memory partitionmemory;
   part3: memory partitionmemory;
end main_memory.impl;


system implementation node.impl
subcomponents
   sys : processor mcdusys.impl;
   pr1 : process com_process.impl;
   pr2 : process mcduapp_process.impl;
   pr3 : process gui_process.impl;
   mem : memory main_memory.impl;

connections
   port pr1.out_arinc_send -> pr2.in_arinc_send;
   port pr2.out_arinc_rec -> pr1.in_arinc_rec;
   port pr2.out_display -> pr3.in_display;
   port pr2.out_annunc -> pr3.in_annunc;
   port pr3.out_userinputs -> pr2.in_userinputs;

properties
   actual_processor_binding =>
      (reference (sys.partition_gui)) applies to pr3;
   actual_processor_binding =>
      (reference (sys.partition_app)) applies to pr2;
   actual_processor_binding =>
      (reference (sys.partition_com)) applies to pr1;
   actual_memory_binding =>
      (reference (mem.part1)) applies to pr1;
   actual_memory_binding =>
      (reference (mem.part2)) applies to pr2;
   actual_memory_binding =>
      (reference (mem.part3)) applies to pr3;
end node.impl;

end mcdu;
```

ANNEXE 4 – Couverture du générateur de code ciblant SIMA

Entité ARINC 653	Entité AADL	Propriétés	Lue et stockée	Générée
Module	Processeur	ARINC653::Module_Major_Frame	OUI	OUI
		ARINC653::Partition_Slots	OUI	OUI
		ARINC653::Slots_Allocation	OUI	OUI
		ARINC653::HM_Errors	OUI	OUI
		ARINC653::HM_Module_Recovery_Actions	OUI	OUI
		ARINC653::HM_Callback	NON	NON
		ARINC653::Module_Name (Valeur défaut : Example for Projet AVIO509)	NON	OUI
		ARINC653::Module_Id	NON	NON
		ARINC653::Module_Version (Valeur défaut : Date)	NON	OUI
		Scheduling_Protocol	NON	NON
Multiple Partitions	Ensemble de processeur virtuel	ARINC653::HM_Errors	NON	NON
		ARINC653::Multipartitions	NON	NON
		ARINC653::HM_Multipartitions_Errors	NON	NON
		ARINC653::HM_Multipartitions_Actions	NON	NON
Partition	Processeur virtuel	Scheduling_protocol	NON	NON
		ARINC653::Criticality (::DAL) (Valeur défaut : LEVEL_A)	NON	OUI
		ARINC653::HM_Errors	OUI	OUI
		ARINC653::HM_Partition_Recovery_Actions	OUI	OUI
		ARINC653::HM_Callback	NON	NON
		ARINC653::Partition_Identifier	NON	OUI
		ARINC653::System_Overhead_Time	NON	NON
		ARINC653::System_Partition	NON	NON
		ARINC653::Entrypoint (Valeur defaut : *entry_point()*)	NON	OUI
	Processus	Source_Data_Size	OUI	OUI
		Source_Code_Size	OUI	OUI
Processus	Processus léger	ARINC653::HM_Errors	OUI	NON
		ARINC653::HM_Process_Recovery_Actions	OUI	NON
		ARINC653::HM_Callback	NON	NON

		ARINC653::Error_Handling	NON	NON
		Source_Code_Size	NON	NON
		Source_Data_Size	NON	NON
		Source_Heap_Size	NON	NON
		Source_Stack_Size	OUI	OUI
		Initialize_Entrypoint	OUI	OUI
		Compute_Execution_Time	OUI	OUI
		Deadline (Valeur défaut : SOFT)	NON	OUI
		Period	OUI	OUI
		Priority (Valeur défaut : Aléatoire)	NON	OUI
Port de mise en file d'attente	Connexion de type « event data ports » entre des processus AADL	Queue_Size	OUI	OUI
		Queue_Processing_Protocol	OUI	OUI
		ARINC653::Timeout	OUI	OUI
		Source_Name	NON	OUI
Port d'échantillonnage	Connexion de type « data ports » entre des processus AADL	ARINC653::Sampling_Refresh_ Period	OUI	OUI
		Source_Name	NON	OUI
Tampon	Connexion de type « event data ports » entre des processus légers d'un même processus	Queue_Size	OUI	OUI
		Queue_Processing_Protocol (Valeur défaut : FIFO)	NON	OUI
		Source_Name	NON	OUI
		ARINC653::Timeout (::Compute_Deadline)	OUI	OUI
Tableau noir	Composant « data » partagé entre processus légers d'un même processus Connexion de type « data ports » entre processus légers d'un même processus	ARINC653::Timeout	NON	NON
		Source_Name	NON	OUI
Sémaphore[11]	Composant « data » partagé entre processus légers d'un même processus	ARINC653::Timeout	NON	NON
		Concurrency_Control_Protocol (Valeur défaut : FIFO)	NON	OUI
		Source_Name (Valeur défaut : aléatoire)	NON	OUI
Événements	Connexion de type	ARINC653::Timeout	NON	NON

11 Seule la création des sémaphores générés. Les manipulations (verrouiller et déverrouiller) ne le sont pas dans cette version.

	« event ports » entre processus légers d'un même processus	Source_Name	NON	OUI
		Queue_Size = 1	OUI	OUI
		Overflow_Handling_Protocol = DropOldest	OUI	OUI
Requis mémoire	Décrit les requis mémoire d'un processus ARINC 653 et spécifie l'allocation des partitions sur celle-ci	Actual_Memory_Binding	NON	NON
		Byte_Count (Word_Count)	OUI	OUI
		Base_Address	NON	NON
		ARINC653::Access_Type	OUI	OUI
		ARINC653::Memory_Type (::Memory_Kind)	OUI	OUI
Pilote de matériel	Composant matériel AADL (type « device ») où le pilote associé est spécifié via les propriétés. Il est lié à un processeur ou un processeur virtuel AADL.	Device_Driver	NON	NON
		Actual_Processor_Binding	NON	NON

Autres concepts	Entité AADL	Propriétés	Lue et stockée	Générée
Lien vers code source usager	Sous-programme	Source_Name	OUI	OUI
		Source_Language	OUI	OUI
		Source_Text	OUI	OUI
Paramètres d'une fonction de code source usager	Connexion de type « parameter » entre un sous-programme et le processus léger qui le contient	N/A	OUI	OUI
Type de donnée	Donnée	Data_Model::Data_Representation	OUI	OUI
		Data_Model::Base_Type	OUI	OUI
		Data_Model::Dimension	OUI	OUI

Concept de configuration du simulateur	Sous-concept	Propriétés	Issue du modèle	Générée
Module	Général	Configuration A653 (Valeur défaut : a653.xml)	NON	OUI
		Script de démarrage (Répertoire d'installation par défaut : $SIMA_ROOT/samples/)	NON	OUI
		Nombre de partitions	OUI	OUI

		Granularité (Valeur défaut : 100 µs)	NON	OUI
	Sortie	Type (Valeur défaut : FIFO)	NON	OUI
		Répertoire	NON	OUI
Partition	Générale	Identificateur de partition	NON	OUI
		Nom de partition	OUI	OUI
		Clé de mémoire partagée *(Incrémenté par dizaine de milliers à partir de la valeur de base aléatoire : 50010)*	NON	OUI
		Script de démarrage (Répertoire d'installation par défaut : $SIMA_ROOT/samples/)	NON	OUI
		Visibilité de la partition (Valeur défaut : Visible)	NON	OUI
	Sortie	Type (Valeur défaut : FIFO)	NON	OUI
		Répertoire	NON	OUI
	Transport des messages	Période allouée en début de fenêtre d'exécution (Valeur défaut : 10 ms)	NON	OUI
		Période allouée en fin de fenêtre d'exécution (Valeur défaut : 10 ms)	NON	OUI
		Priorité (Valeur défaut : 1)	NON	OUI
		Fenêtres concernées par ces propriétés (Valeur défaut : Toute)	NON	OUI
	Port de destination	Nom	OUI	OUI
		Type (Valeur défaut : UDP)	NON	OUI
		Adresse IP (Valeur défaut : 127.0.0.1)	NON	OUI
		Numéro de port *(Incrémenté par dizaine à partir de la valeur de base aléatoire : 12351)*	NON	OUI
Pseudo partition	Port de destination	Nom	NON	NON
		Type (Valeur défaut : UDP)	NON	NON
		Adresse IP (Valeur défaut : 127.0.0.1)	NON	NON
		Numéro de port (Valeur défaut : aléatoire)	NON	NON