Modélisation de systèmes avioniques en AADL

par

Rim BEN DHAOU

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN GÉNIE DES TECHNOLOGIES DE
L'INFORMATION
M. Sc. A.

MONTRÉAL, LE 14 FÉVRIER 2018

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Roger Champagne, directeur de mémoire
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Yann-Gaël Guéhéneuc, codirecteur
Département de génie informatique et génie logiciel à l'université de Concordia

M. Christopher Fuhrman, président du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

Mme Ghizlane El Boussaidi, membre du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 19 JANVIER 2018

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

A travers ce travail, je tiens à remercier mon directeur de recherche le Professeur Roger CHAMPAGNE pour son support, ses conseils et sa disponibilité tout au long de mon mémoire. Son implication active et ses suggestions pertinentes m'ont permis d'avancer dans mon projet de mémoire. Je remercie également mon co-directeur Mr. Yan Gaël pour sa collaboration et ses commentaires.

Ensuite je remercie, tous nos partenaires subventionnaires et industriels, le Consortium de recherche et d'innovation en aérospatiale au Québec (CRIAQ), le Conseil de recherche en sciences naturelles et en génie (CRSNG), Esterline CMC Electronics et Solutions Isonéo pour leur soutien financier et humain.

Je voudrais remercier aussi mes collègues du laboratoire en architecture de systèmes informatiques (LASI) de l'École de technologie supérieure (ETS) pour leurs encouragements et suggestions enrichissantes.

Je voudrais adresser mon affectueuse reconnaissance à ma famille: mes chers parents qui mon permis d'arriver à ce niveau, ma sœur et son mari pour leur patience et leur encouragement et enfin mon frère pour son humour tout le temps.

MODÉLISATION DE SYSTÈMES AVIONIQUES EN AADL

Rim BEN DHAOU

RÉSUMÉ

Dans l'industrie aéronautique, les équipements et les systèmes avioniques sont généralement conçus et développés par des organisations différentes. L'intégration des équipements et systèmes est effectuée à l'aide des documents de contrôle d'interface (*Interface Control Document,* ICD) qui doivent fournir une information complète et claire sur les interfaces de ces équipements et systèmes. Le processus de rédaction et de gestion des ICD est souvent coûteux et délicat puisque l'omission ou l'ambiguïté d'un petit détail peut entrainer l'échec d'une intégration ou une panne potentielle pour des systèmes critiques.

Dans ce contexte, notre travail a comme objectif d'utiliser le langage de modélisation AADL pour modéliser des systèmes avioniques modulaires intégrés (*Integrated Modular Avionics*, IMA) en mettant l'emphase sur les informations reliées aux interfaces afin d'extraire les ICD à partir de ces modèles. Nous expérimentons le langage AADL et son écosystème d'outils libres à travers une étude de cas pour vérifier s'il permet de modéliser tous les aspects et les informations pertinentes à inclure dans un ICD. Nous évaluons également s'il nous permet de valider les modèles obtenus et de vérifier leur conformité aux normes et standards avioniques pertinents.

Notre étude de cas consiste en la modélisation d'un *Air Data Inertial Reference Unit* (ADIRU) et son interaction avec d'autres équipements. Nous modélisons quelques fonctionnalités de l'ADIRU comme preuve de concept afin de démontrer la faisabilité de notre approche. Nous validons ensuite si notre modèle répond aux exigences des normes ARINC 653 et ARINC 664 qui sont les deux principales normes reliées aux systèmes IMA.

AADL présente des limites dans la modélisation de la structure des données et des messages échangées entre les applications. Nous proposons donc de le complémenter avec l'usage du langage DFDL (*Data Format Description Language)* qui nous permet de décrire les messages et leurs structures spécifiques.

Ainsi, nous montrons qu'AADL, quand complémenté par DFDL, nous permet de présenter les informations reliées aux interfaces et d'extraire les ICD. Nous recommandons comme travaux futurs de mener une validation expérimentale avec de vrais intégrateurs pour comparer notre approche avec leurs outils existants en termes de réduction des coûts de production des systèmes avioniques.

Mots-clés : modélisation, systèmes avioniques, AADL, DFDL, interface contrôle documents

MODÉLISATION DE SYSTÈMES AVIONIQUES EN AADL

Rim BEN DHAOU

ABSTRACT

In the aeronautical industry, avionics equipment and systems are generally designed and developed by different organizations. Their integration is performed using Interface Control Documents (ICDs), which must provide complete and clear information about the subsystem interfaces. The ICD writing process and usage is often expensive and delicate, because the omission or ambiguity of a small detail can lead to the failure of the integration or to a potential failure of the critical systems.

In this context, our work aims to use the AADL modeling language to model Integrated Modular Avionics (IMA) subsystems with an emphasis on the interface-related information to extract ICDs from these models. We experiment with AADL and its ecosystem of free tools through a case study to check if it allows modeling all aspects and relevant information to be included in an ICD. We also evaluate whether it allows us to validate the obtained models and to verify their compliance with relevant avionics standards.

Our case study consists of the modeling of an Air Data Inertial Reference Unit (ADIRU) and its interaction with other equipment. We model some features of the ADIRU as proof of concept to demonstrate the feasibility of our approach. We then validate whether our model meets the requirements of the ARINC 653 and ARINC 664 standards, which are the two main standards related to IMA systems.

AADL presents limitations in modeling the structure of data and messages exchanged between applications. We therefore propose to complement it with the use of the Data Format Description Language (DFDL), which was used to describe the messages and their specific structures.

Thus, we show that AADL, when complemented by DFDL, allows us to present information related to interfaces and to extract ICDs. We recommend as future work to carry out an

experimental validation with real integrators to compare our approach with their existing tools in terms of reduction of production costs of avionics systems.

Keywords: modeling language, avionics systems, AADL, DFDL, Interface Control Documents

TABLE DES MATIÈRES

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AADL	Architecture Analysis and Design Language
ACL	Access Control List
ADIRS	Air Data Inertial Reference System
ADIRU	Air Data Inertial Reference Unit
ADR	Air Data Reference
AFDX	Avionics Full DupleX switched Ethernet
APEX	APplication EXecutive
ARINC	Aeronautical Radio INCorporated
BAG	Bandwidth Allocation Gap
BCD	Binary Coded Decimal
CPM	Core Processing Module
CPU	Central Processing Unit
CRIAQ	Consortium de Recherche et d'Innovation en Aérospatiale au Québec
ES	End System
FMS	Flight Management System
GWM	GateWay Module
ICD	Interface Control Document
ICDM	Interface Control Document Management
ILS	Instrument Landing System
IMA	Integrated Modular Avionics
IOM	Input Output Module
IR	Inertial Reference
LRM	Line Replaceable Module
PCI	Peripheral Component Interconnect
RDC	Remote Data Concentrator
SAE	Society of Automotive Engineers
VL	Virtual Link
XML	eXtensible Markup Language

INTRODUCTION

Les systèmes avioniques sont composés de différents équipements, composés eux-mêmes de matériel et de logiciel, qui interagissent ensemble pour accomplir certaines fonctionnalités. L'interaction ou l'échange d'informations entre ces équipements est assuré par des interfaces physiques/logiques. Pour avoir une bonne intégration et coopération entre les équipements avioniques, il faut que les interfaces soient décrites d'une manière claire, précise, complète et non ambiguë. Dans le milieu de l'aéronautique, on capture souvent les informations liées aux interfaces dans des documents de contrôle d'interface (*Interface Control Document,* ICD). Les équipements avioniques sont conçus et fabriqués par différents fournisseurs, ce qui rend ces documents plus complexes dans l'absence d'un langage commun pour la définition des ICD et d'une spécification standard de ce qu'il faut préciser dans un ICD ou de son organisation (Louadah, H. *et al*., 2014).

Le traitement manuel des ICD hétérogènes est compliqué puisqu'il n'y a pas de solution standard, commune et automatisée pour la gestion de ces documents. Ceci peut engendrer des problèmes dans l'intégration des équipements des systèmes avioniques, considérés comme critiques. De plus, le processus de production et de traitement des ICD n'est ni standardisé ni automatisé, ce qui entraîne des risques d'erreurs et une lenteur significative dans la conception, la fabrication et l'intégration des équipements avioniques.

Dans les systèmes avioniques, il y a essentiellement deux architectures couramment utilisées: l'architecture fédérée, où chaque fonction avionique possède ses propres ressources et sa plateforme d'exécution, et l'architecture modulaire intégrée (*Integrated Modular Avionics,* IMA) qui est basée sur un environnement partitionné où plusieurs fonctions avioniques sont exécutées par une plateforme d'exécution partagée (Watkins, C. B., & Walter, R., 2007). Chacune de ces architectures a des besoins différents et spécifiques en termes d'ICD.

Notre projet s'inscrit dans le cadre du projet CRIAQ AVIO-506 ayant comme objectif de proposer une méthodologie, une démarche ou des outils pour le développement et la gestion des ICD pour l'intégration des systèmes avioniques.

Le projet AVIO-506 implique quatre partenaires subventionnaires/industriels: le Consortium de recherche et d'innovation en aérospatiale au Québec (CRIAQ), le Conseil de recherche en sciences naturelles et en génie (CRSNG), Esterline CMC Electronics et Solutions Isonéo. Outre l'ÉTS, des chercheurs de deux autres universités sont aussi impliqués: l'École Polytechnique de Montréal et l'Université Carleton.

Le projet AVIO-506 a commencé avec une autre équipe de recherche qui s'occupe d'étudier une solution logicielle commerciale, nommée CITRUS, dans le but d'analyser et d'évaluer son utilisation dans la gestion des interfaces et des ICD. Cette solution est éditée par Solutions Isonéo, l'un des partenaires industriels, pour supporter le cycle de développement des systèmes avioniques de la compagnie Esterline CMC Electronics.

La motivation de notre projet est de trouver une suite d'outils libres qui permet de gérer les interfaces et les ICD au lieu d'avoir recours à une solution propriétaire et commerciale. De cette motivation, notre objectif est d'explorer un langage libre conçu spécifiquement pour la modélisation des systèmes embarqués temps réel AADL (*Architecture Analysis and Design Language*), afin d'évaluer ses capacités et son utilité pour raisonner sur les ICD. L'idée est donc de profiter d'une suite d'outils existants et libres autour de ce langage pour capturer des informations d'interfaces de systèmes avioniques.

AADL est un langage de modélisation architecturale orienté composants qui décrit les composants logiciels et matériels d'un système. Ce langage est dédié aux systèmes embarqués temps réel, ce qui inclut les systèmes avioniques.

L'intérêt de notre projet est porté vers AADL comme élément central d'une infrastructure de modélisation des systèmes avioniques. Nous explorons le potentiel et les limites d'AADL pour la modélisation et l'extraction des ICD. Notre travail de recherche consiste donc à tenter de répondre à ces questions:

- Est-ce que les outils libres existants dans l'écosystème AADL nous permettent de modéliser les aspects pertinents des interfaces de systèmes avioniques à inclure dans des ICD?

- Est-ce qu'un écosystème d'outils libres basé sur AADL permet d'effectuer certaines formes de validation lors de l'intégration des équipements avioniques, par exemple la vérification de la compatibilité des interfaces des différents composants?

Pour tenter de répondre à ces questions, nous avons fixé les objectifs suivants:

1- Déterminer quelles sont les informations d'interfaces nécessaires et pertinentes que nous devons extraire pour chacun des deux types principaux d'architecture en avionique, soit fédérée et modulaire.

2- Modéliser avec AADL un cas d'équipement avionique dans un contexte modulaire (IMA) et analyser les capacités et les limites d'AADL quant à la modélisation des interfaces et valider ainsi la conformité des modèles obtenus aux standards avioniques (principalement ceux d'ARINC).

Pour atteindre nos objectifs, nous avons suivi la méthodologie suivante:

- Faire une étude bibliographique sur AADL et son écosystème et sur les systèmes avioniques afin de circonscrire les différentes architectures, normes et standards.

- Expérimenter, par une étude de cas, la modélisation AADL d'un équipement avionique en prenant le cas d'un *Air Data Inertial Reference Unit* (ADIRU). Ce choix est motivé par le fait que certaines fonctionnalités de l'ADIRU ont été modélisées en AADL par d'autres chercheurs dans le but de générer le code source (Hugues, J. & Delange, J., 2015).

- Modéliser spécifiquement les données échangées à travers les interfaces de l'équipement.

- Extraire les ICD à partir des modèles obtenus.

Notre rapport de mémoire est structuré comme suit. Le premier chapitre est un survol de littérature où les systèmes avioniques et leurs architectures sont définis. Les principaux standards portant sur ces systèmes sont ainsi présentés en mettant l'accent sur ceux qui

régissent les systèmes IMA, contexte de notre mémoire. Nous introduisons par la suite le langage AADL en présentant ses composants et ses principales caractéristiques. Nous présentons des travaux de recherche pertinents visant à traiter la problématique de la gestion des ICD dans le but de comprendre le principe et l'organisation des ICD selon différents chercheurs et quels sont les outils utilisés pour générer ces ICD.

Le deuxième chapitre présente l'étude de cas et les expérimentations de modélisation menées durant notre projet. Au début de ce chapitre l'étape de collecte d'information est détaillée dans les deux contextes, fédéré et IMA, dans le but de souligner la différence entre les interfaces dans chacun de ces deux contextes par des exemples d'équipements avioniques. Cette étape nous prépare à l'étape de modélisation, où nous présentons deux tentatives de modélisation sur lesquelles nous nous sommes basées. Ensuite nous proposons notre modèle AADL qui vise à modéliser quelques fonctionnalités d'un système IMA (ADIRU) en le validant avec une suite de règles de validation pertinentes. À la fin de ce chapitre, nous évaluons l'expérience de modélisation avec AADL tout en nous concentrant sur les informations relatives aux interfaces.

Le troisième chapitre est consacré à la modélisation des structures de données échangées qui représentent des informations intéressantes dans un ICD. Nous décrivons d'abord la structure spécifique des données qui sont communiquées dans le contexte IMA et nous présentons ensuite le langage DFDL (*Data Format Description Language*) et l'outil de modélisation utilisés. Les principaux résultats sont ainsi présentés. Nous résumons ce chapitre avec un rappel de toute la démarche de notre étude de cas qui nous a mené à ces résultats.

Nous clôturons ce mémoire par une synthèse de notre travail de recherche et des recommandations pour de futurs travaux.

CHAPITRE 1

Revue de la littérature

Ce chapitre résume notre revue de la littérature, en relation avec le sujet abordé. Une introduction aux systèmes avioniques est d'abord présentée, où certaines notions sur les architectures et les standards pertinents sont décrites. Ensuite, le langage AADL est présenté en soulignant son intérêt pour notre projet. Afin de comprendre ce qu'est un ICD et ce qu'il doit contenir, nous citons quelques travaux de recherche axés sur les ICD ou ayant parmi leurs objectifs la gestion des ICD.

1.1 Les architectures des systèmes avioniques

Un système avionique, d'un point de vue fonctionnel, est un ensemble de sous-systèmes assurant diverses fonctionnalités, comme la navigation, le pilotage, l'atterrissage, la communication (entre avions, entre différents éléments de l'avion ou entre l'avion et la terre), le contrôle de vol, la connaissance de l'environnement, etc.

D'un point de vue architectural, on retrouve principalement deux architectures dans les systèmes avioniques: l'architecture fédérée et l'architecture modulaire intégrée (IMA), cette dernière étant relativement récente par rapport à la première. L'IMA, considérée comme successeur des architectures fédérées, a été adoptée pour faire face aux divers défis de l'architecture classique fédérée (Alena R. *et al.*, 2007). Ces problèmes sont décrits dans les paragraphes suivants.

L'architecture classique des systèmes avioniques a été conçue en définissant des fonctions qui sont implémentées comme des unités fonctionnelles fédérées où chaque unité fonctionnelle est une application qui possède ses propres ressources ou plateforme matérielle, et qui accomplit des fonctions bien déterminées. Il s'agit dans ce cas d'avoir un calculateur par unité fonctionnelle. La communication entre ces calculateurs est effectuée par des canaux dédiés, comme présenté sur la figure 1.1. De même, les interfaces qui assurent l'échange d'information entre ces applications sont conçues selon les besoins de communication entre

les applications. Dans cette figure, chaque interconnexion représente un bus de données possédant un seul équipement émetteur et un ou plusieurs équipements récepteurs. Par exemple, l'équipement EADI #1 reçoit des informations de neuf autres équipements. Il faut donc neuf bus de données pour assurer la communication.

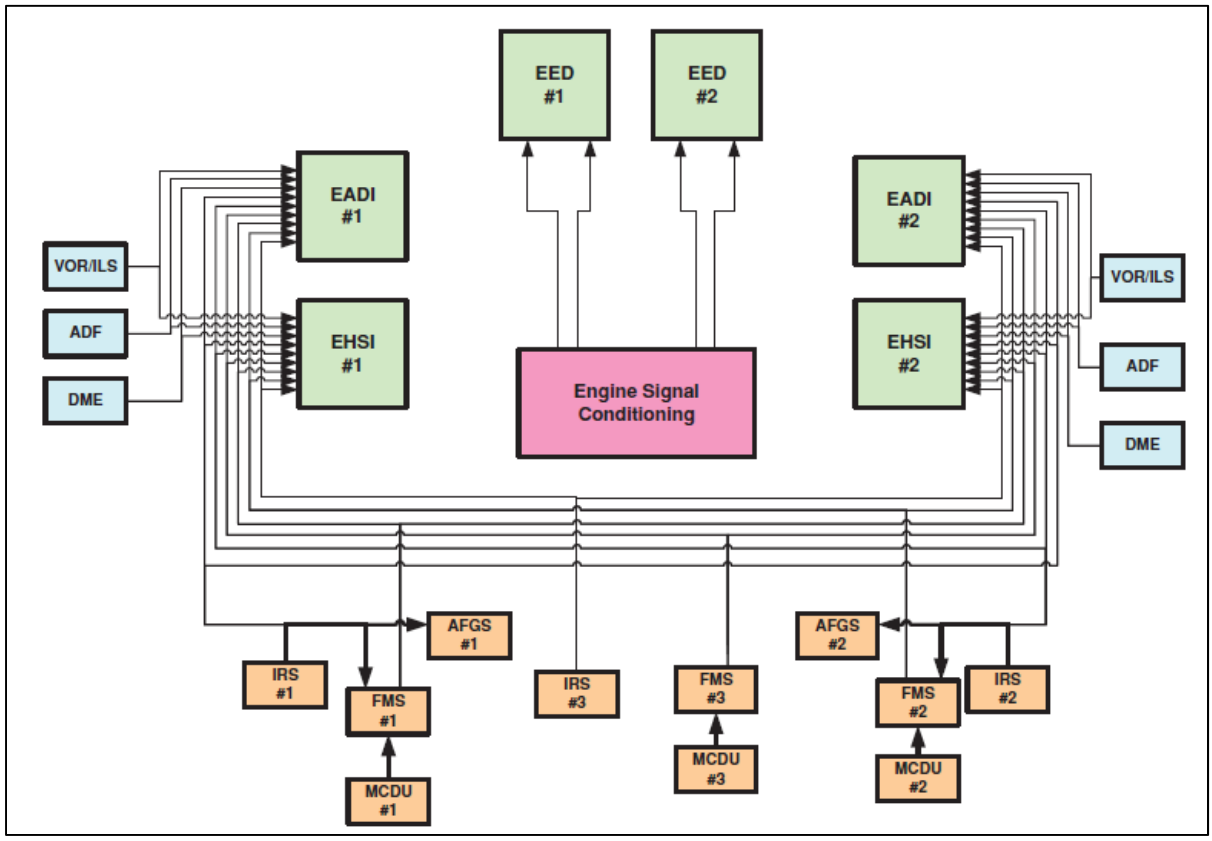


Figure 1.1 Exemple d'architecture fédérée. Figure extraite de (Moir I. *et al*., 2013)

Les fonctionnalités demandées en avionique sont en pleine croissance, il s'ensuit un accroissement exponentiel de la complexité des systèmes fédérés à cause de l'augmentation des nombres de calculateurs et du câblage, entraînant aussi des problèmes de poids, de dimension, de consommation et de coût (Alena R. *et al*., 2007). De plus, ces systèmes ne sont pas assez flexibles pour permettre des modifications ou extensions. Grâce à l'évolution de la puissance des processeurs, la solution apportée par l'IMA est fondée sur le principe de l'exécution de plusieurs fonctions avioniques sur le même calculateur. Une architecture IMA est donc constituée de deux éléments essentiels: des calculateurs génériques et configurables pour l'exécution de plusieurs applications et un réseau de communication partagé assurant

l'échange de données entre les calculateurs. Ces calculateurs sont des unités d'exécution standardisées ayant leurs propres ressources (processeur, mémoire, …), appelés *Line Replaceable Module* (LRM) et regroupés généralement sur des bâtis (*racks*). La figure 1.2 présente une étagère contenant des LRM (A, B, C, ...) et l'architecture physique d'un LRM (à droite de la figure). En IMA, il y a trois types de LRM :

- le *Core Processing Module* (CPM) qui est un module de calculs chargé de l'exécution des applications avioniques;
- l'*Inpout Output Module* (IOM) qui permet la communication entre les sous-systèmes hétérogènes (IMA et non IMA);
- le *GateWay Module* (GWM) qui permet la communication entre les différentes étagères de LRM.

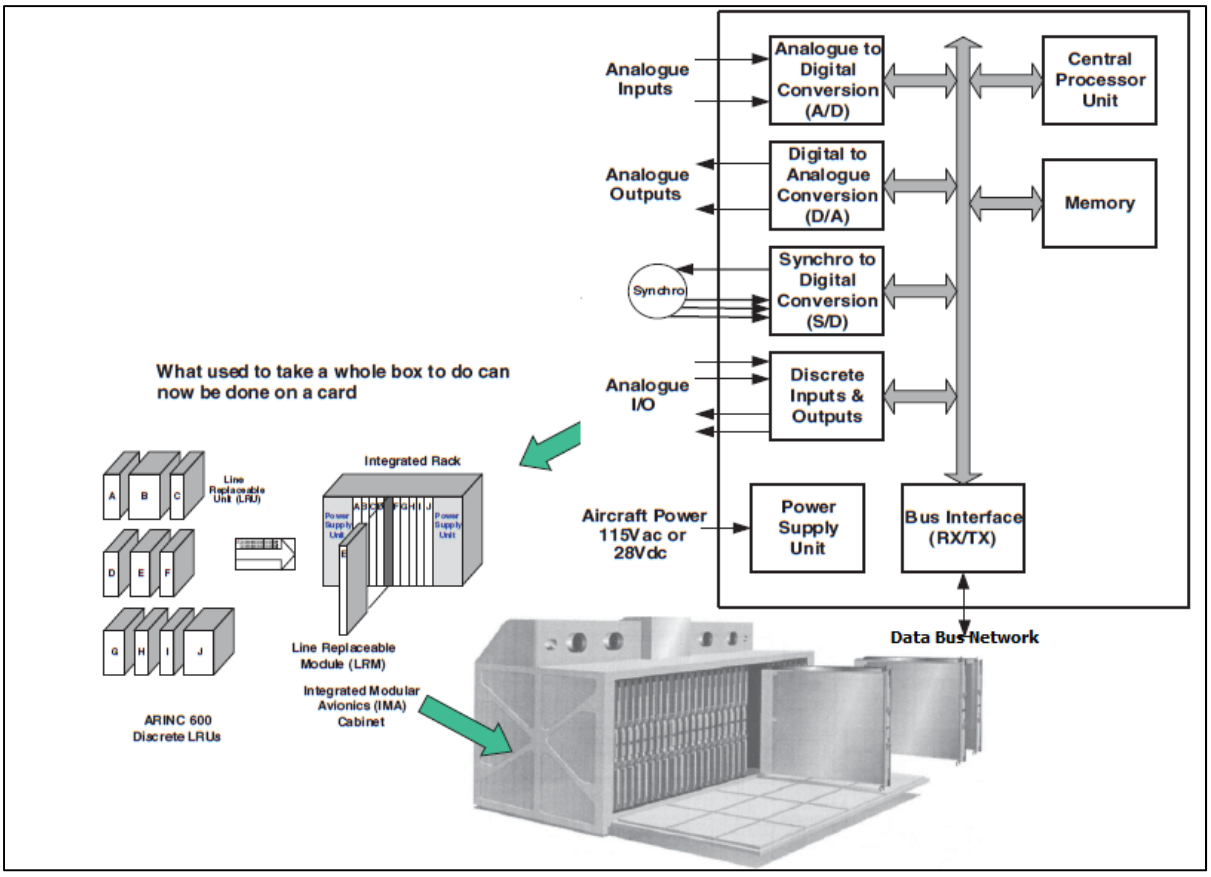


Figure 1.2 Architecture Modulaire Intégrée. Figure extraite de (Moir I. *et al*., 2013)

1.2 Standards avioniques pertinents

Aeronautical Radio, Incorporated (ARINC) est une société détenue par les principaux constructeurs et compagnies aéronautiques. Elle définit et publie les principaux standards de communications des aéronefs portant sur les réseaux, les bus et les protocoles utilisés dans les systèmes avioniques. Dans cette section nous présentons trois standards pertinents dans notre projet: un standard lié à l'architecture fédérée (ARINC 429) et deux standards liés à l'architecture IMA (ARINC 653 et ARINC 664). Nous nous concentrons sur ces deux derniers standards puisque notre recherche est focalisée sur l'architecture IMA.

1.2.1 ARINC 429

Ce standard décrit le bus de données qui permet d'acheminer des données entre calculateurs. La topologie de ces bus est très simple: les calculateurs sont connectés point à point par une liaison directe unidirectionnelle. Ces types de bus sont utilisés dans le contexte fédéré où les communications sont dédiées. Pour chaque bus ARINC 429, le standard exige qu'il y ait un seul émetteur et un ou plusieurs récepteurs comme présenté dans la figure 1.3. Les données échangées sur ce bus sont codées sur 32 bits et ont une structure bien spécifique.

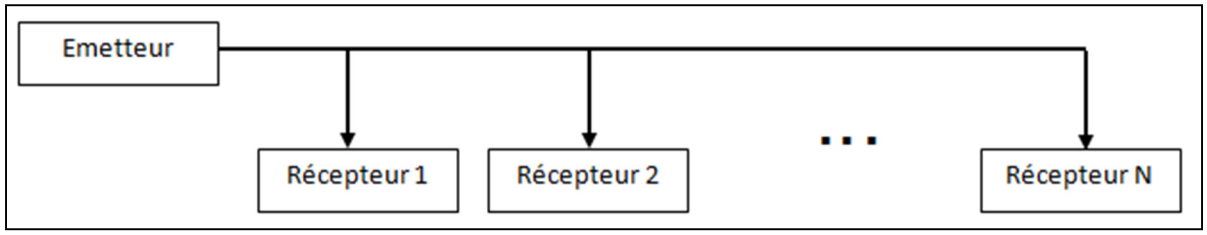


Figure 1.3 Topologie du bus ARINC 429

1.2.2 ARINC 653

Dans une architecture IMA, plusieurs applications logicielles sont hébergées dans une seule plateforme matérielle intégrée, appelée aussi module (figure 1.4), où peuvent se trouver un ou plusieurs microprocesseurs (Samolej, S., 2011). Chaque application est décomposée en plusieurs partitions par mesure de protection et de séparation fonctionnelle entre ces applications (Ananda C.M. *et al*., 2013). Chaque partition est composée d'un ou plusieurs

processus et chaque processus est de même composé d'une ou plusieurs tâches. Le standard ARINC 653 est adopté par l'IMA pour la gestion du partitionnement spatial et temporel des ressources favorisant ainsi une bonne portabilité des applications avioniques.

Le partitionnement spatial vise à interdire tout partage de ressources physiques entre partitions. Il faut s'assurer que chaque partition a ses propres ressources (chaque ressource partageable est allouée à une seule partition) et qu'aucune partition n'affecte ou ne change les données allouées à une autre. Des exemples de ressources sont: le processeur, la mémoire, les bus de données, *etc*.

Le partitionnement temporel vise à accorder pour chaque partition un accès exclusif au processeur pendant un laps du temps. Un ordonnanceur gère le plan d'exécution entre les partitions et assigne des tranches de temps pour chacune. L'ordonnancement est géré par une fenêtre ou un bloc d'activation majeur (*Major Frame Period*) ayant une durée fixe et cyclique. Une fenêtre d'activation majeure est prédéfinie pour chaque module durant laquelle un ensemble de partitions seront exécutées en allouant pour chacune d'elles un laps de temps.

L'objectif de ce standard est de définir une interface exécutive d'application (*APplication EXecutive*, APEX) entre le noyau logiciel du module (*Core Software*) et l'application logicielle (ARINC 653 Part 0). L'APEX fournit une interface commune pour l'application logicielle en offrant des requêtes et des services de gestion normalisés et groupés en sous-catégories: gestion de partition, gestion de processus, gestion du temps, communication inter et intra-partition, gestion de la mémoire, monitorage de la santé du système ou gestion des pannes.

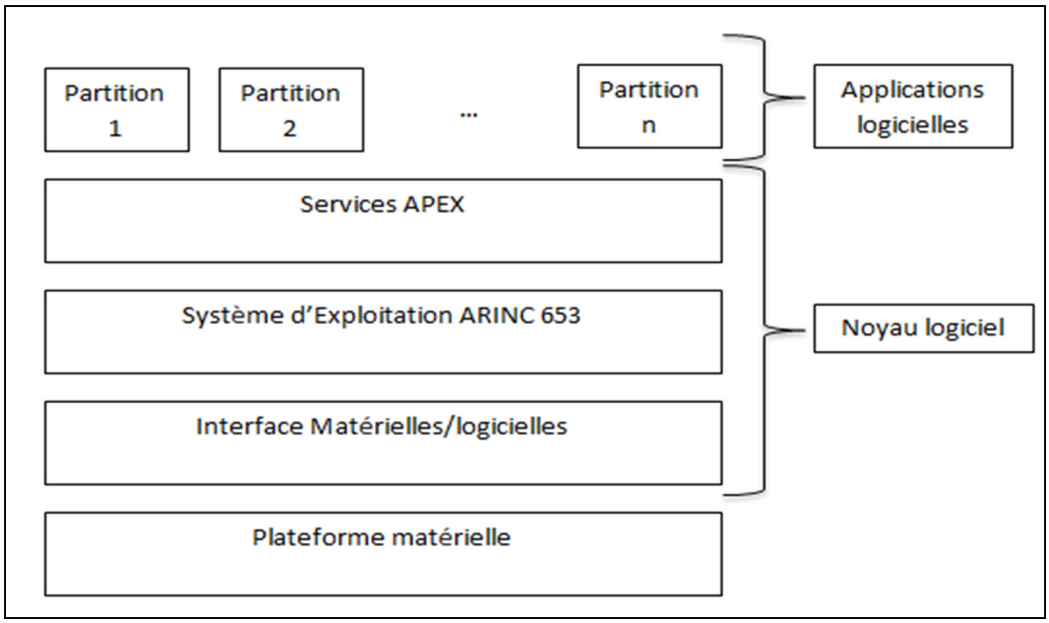


Figure 1.4 Exemple d'architecture d'un module avionique (AEEC., 2010)

La communication inter-partition est assurée par des ports de files d'attente (*Queuing ports*) et des ports d'échantillonnages (*Sampling ports*). Des moyens de synchronisation comme les sémaphores, les événements et les messages assurent la communication intra-partition (Savard J., 2012).

1.2.3 ARINC 664

Avionics Full DupleX switched ethernet (AFDX), normalisé dans la partie 7 du standard ARINC 664, est un réseau Ethernet commuté et adapté aux besoins de communication entre les modules avioniques. Il est introduit dans l'architecture IMA afin d'assurer un échange de données redondant et à haut débit entre les composants des systèmes avioniques. Il permet de remplacer les liaisons point à point des systèmes fédérés par des liaisons virtuelles (*Virtual Links*, VL). La redondance est exigée dans le contexte de systèmes critiques pour réduire la perte de données échangées. Le principe de redondance est implémenté physiquement par une double (ou multiple) transmission des messages sur des liens différents vers leurs destinations.

La topologie du réseau AFDX est constituée des liens virtuels (les VL), des équipements terminaux appelés *End Systems* (ES) et des commutateurs (*Switches*).

Les VL sont des liens logiques qui forment un chemin unidirectionnel (un seul émetteur et un ou plusieurs récepteurs) pour échanger des données entre les ES. Des spécifications AFDX sont associées aux VL pour répondre aux contraintes temps réels de la transmission de données. Selon ces spécifications, un VL est caractérisé par:

- la taille maximum et la taille minimum d'une trame (S_{max}, S_{min});
- le temps minimum entre deux émissions de trames consécutives appelé BAG (*Bandwidth Allocation Gap*). Le BAG est toujours **1 ms * 2^k**, avec k un entier de 0 à 7 : le BAG peut-être 1ms, 2ms, … 128 ms;
- un identifiant unique ou nom du VL;
- un émetteur unique;
- un ou plusieurs récepteurs;
- un chemin bien défini sur le réseau.

Les paramètres de taille maximale de la trame et le BAG permettent d'évaluer statiquement la bande passante maximale autorisée pour un VL.

Les commutateurs, définis sous la norme 802.1D, servent à vérifier, filtrer et acheminer les trames sur le réseau AFDX selon des tables de routages préconfigurées dans chaque commutateur. Le filtrage de trafic est géré par des listes de contrôle d'accès (ACL) qui assurent le filtrage en fonction des adresses Mac ou Ethernet.

La figure 1.5 présente un exemple simplifié d'un réseau AFDX.

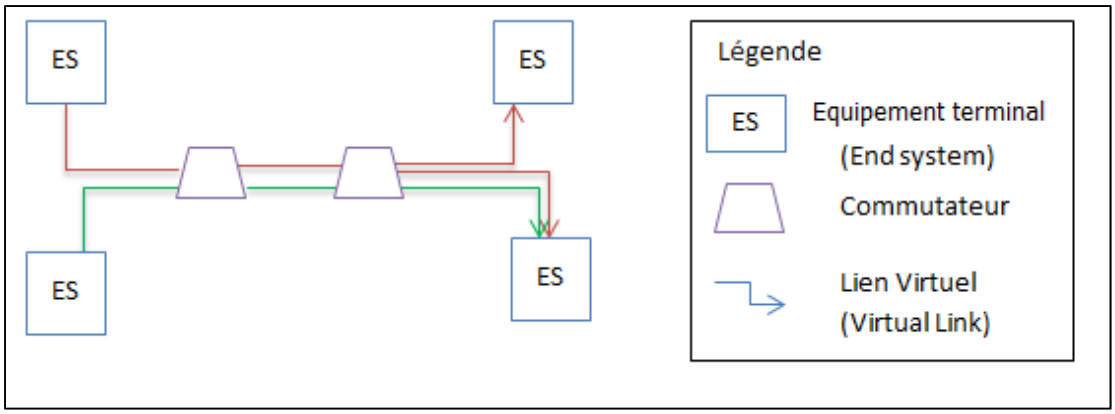


Figure 1.5 Exemple de réseau AFDX avec deux VL
(chaque couleur de lien virtuel indique un VL différent)

1.3 AADL

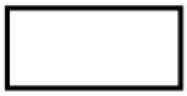
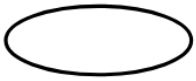
Architecture Analysis and Design Language (AADL) est un langage de description d'architecture standardisé par SAE (*Society of Automotive Engineers*). Ce langage a été conçu originellement pour les systèmes avioniques puis a été généralisé/adapté pour couvrir tous les systèmes embarqués temps réels (Feiler, P. H., Lewis, B., & Vestal, S., 2004). C'est un langage basé sur les composants (*component-based*) qui décrit à la fois les composants matériels et logiciels du système (Feiler, P. H., & Gluch, D. P., 2012). Il offre un ensemble de types de composants essentiels pour la modélisation des systèmes embarqués complexes. En AADL, trois catégories principales de composants sont disponibles:

- les composants logiciels, qui présentent les éléments applicatifs, comme les données, les programmes, les processus, les *threads* (fils d'exécution);
- les composants matériels, qui décrivent les éléments de la plate-forme d'exécution, comme les processeurs, les processeurs virtuels, les bus, les mémoires;
- les composants hybrides, ou composites, qui sont une combinaison de matériel et de logiciel, comme le système.

1.3.1 Composants AADL

Le tableau suivant présente les principaux composants AADL, leur définition et leur représentation graphique (Feiler, P. H., & Gluch, D. P., 2012).

Tableau 1.1 Principaux composants d'AADL

Composant	Définition	Représentation Graphique
Donnée (**`data`**)	Décrit les structures de données échangées entre les composants	
Sous-programme (**`sub program`**)	Représente un fragment de code exécutable	
Fil d'exécution (**`thread`**)	Définit les éléments logiciels actifs	
Processus (**`process`**)	Représente un espace mémoire dans lequel s'exécutent les threads	
Processeur (**`processor`**)	Représente un processeur physique pour l'exécution des threads	
Mémoire (**`Memory`**)	Définit un dispositif de stockage	
`Bus`	Modélise le moyen de communication entre les composants comme un réseau, un bus de communication, un simple fil, internet, ...	
`Device`	Décrit des éléments dont la structure interne est inconnue. Il suffit de préciser son interface et ses caractéristiques externes.	
Système (**`system`**)	Permet de structurer hiérarchiquement une architecture et de présenter ses différents composants (logiciels ou matériels)	

Certains de ces composants possèdent une version virtuelle (c'est-à-dire logique et non pas physique) comme le bus virtuel (**`virtual bus`**) qui représente une interconnexion logique, le processeur virtuel (**`virtual processor`**) qui représente un processeur logique, le processus virtuel (**`virtual process`**) qui représente une partition logicielle, *etc*.

Pour un composant, AADL définit son type et son implémentation:

- le type présente son interface (c'est-à-dire sa description externe ou comment il est observé de l'extérieur). Il décrit les interactions du composant avec son environnement et définit les interfaces requises et fournies par ce composant comme les ports d'entrée/sorties et les services fournis ou requis;
- l'implémentation décrit sa structure interne (c'est-à-dire ses sous-composants et les connexions qui les lient). Elle décrit ainsi le fonctionnement interne de ce composant, ses propriétés et son comportement.

AADL laisse une grande liberté pour le choix du niveau d'abstraction et des détails des composants. La description architecturale d'un système peut être effectuée avec différents niveaux de détails tout en gardant la possibilité de la raffiner plus tard. Le concepteur peut ainsi décrire un composant d'une manière plus abstraite et différer sa description détaillée à plus tard dans le cycle de développement du système. Cette capacité est très intéressante parce qu'elle permet d'obtenir rapidement des modèles abstraits dont les détails peuvent être ajoutés ultérieurement d'une manière incrémentale.

1.3.2 Les éléments d'interfaces

Les composants AADL possèdent des ports de communications qui représentent les points d'entrées ou de sorties d'un composant et qui peuvent transférer du contrôle ou des données. Pour chaque port on peut spécifier sa direction (**`in`**, **`out`** ou **`in out`**), son type (**`event`**, **`data`** ou **`event data`**) et son identifiant. Le port d'évènement (**`event port`**) permet la réception d'un signal qui déclenche un évènement à sa réception comme l'exécution d'un thread. Par contre, le port de donnée (**`data port`**) permet la réception d'un signal qui ne

déclenche aucun événement lors de sa réception mais il transporte des données. Le port évènement/donnée (`event data`) est une combinaison des deux ports, il permet de transmettre des données tout en déclenchant des évènements. Ce type de port est typiquement utilisé pour la modélisation des messages.

Les interfaces d'un composant AADL sont déclarées dans la section `features` de ce composant. Cette section définit les ports d'entrées/sorties, les paramètres du sous-programme et les services d'accès à des sous-composants où on distingue deux types de services : requis (`requires`) ou fournis (`provides`).

1.3.3 Les flux

AADL modélise les relations entre les composants à travers les connexions entre ports. Les flux sont utilisés en AADL pour représenter une abstraction de la traversée bout-en-bout des données à travers le système.

1.3.4 Les propriétés

Pour les caractéristiques non structurelles, comme le temps d'exécution, la période d'une tâche, la taille d'une mémoire, ..., AADL propose des propriétés qui peuvent être associées à un ou plusieurs éléments (composant, port, …). AADL possède des propriétés prédéfinies pour des catégories de composants et offre la possibilité d'ajouter de nouvelles propriétés. Le concepteur peut ainsi définir des propriétés spécifiques qui se rapportent à (ou en rapport avec) son domaine de préoccupation et les associer aux composants de son modèle (Evensen, K., & Weiss, K., 2010).

1.3.5 Les annexes

L'annexe est un mécanisme d'extension fourni par AADL pour ajouter des informations supplémentaires. Les informations complémentaires de l'annexe peuvent être exprimées dans un autre langage comme Z, *Object Constraint Language* (OCL) ou autre, indépendante de la syntaxe standard d'AADL. Certaines annexes sont standardisées, comme l'annexe pour la

modélisation du comportement logiciel et l'annexe ARINC 653 pour la modélisation des systèmes conformes à l'architecture IMA.

1.4 Les interfaces et les ICD

Dans cette section nous résumons les définitions des interfaces et des ICD, leurs objectifs et leurs défis.

1.4.1 Généralités et définitions

Dans notre contexte, une interface est considérée comme une frontière permettant l'interaction et l'échange de données entre des composants ou systèmes différents. La gestion des interfaces est très importante dans l'intégration de systèmes complexes. Elle permet d'éviter les problèmes d'assemblage et de coopération entre sous-systèmes. Une documentation détaillée, claire et précise des interfaces est donc un élément clé dans la conception et le développement des systèmes critiques. Idéalement, la description de ces interfaces doit contenir toutes les informations nécessaires pour assurer et vérifier l'intégrité du système. Dans notre travail, nous supposons que ces informations sont capturées dans un ou plusieurs ICD.

Dans la littérature, nous n'avons pas trouvé de définition claire et complète de ce qu'un ICD doit contenir et de comment il doit être organisé. De plus, dans le domaine des équipements avioniques, il est difficile de trouver un exemple concret d'ICD pour se faire une idée plus claire sur ces documents, parce que ces ICD contiennent souvent de la propriété intellectuelle hautement confidentielle que les entreprises ne veulent pas partager.

Cependant, il y a des tentatives de recherches menées dans l'axe de gestion des ICD, que ce soit dans le domaine des systèmes avioniques ou autres. Rahmani et Thomson ont proposé une méthode systématique pour identifier et décomposer les interfaces dans le but de générer un ICD structuré (Rahmani, K., *et* Thomson, V., 2009). Selon ces chercheurs, l'objectif d'un ICD est de formaliser la description de la connectivité entre deux ou plusieurs sous-systèmes. Pour cela il faut bien connaitre les frontières des sous-systèmes pour pouvoir identifier les

interfaces. De plus, ils jugent que les ICD ne doivent pas nécessairement détailler les interfaces et les données échangées mais plutôt contenir les paramètres, les caractéristiques et les configurations nécessaires pour assurer l'intégrité du système. L'approche proposée consiste à décomposer le système en sous-systèmes selon leurs fonctionnalités et, de la même manière, décomposer chaque sous-système en sous-sous-systèmes. Il s'agit d'une décomposition hiérarchique du système en sous-systèmes dont les frontières physiques et fonctionnelles sont claires. Chaque interface est par la suite décrite en quarte types de description (spatiale: orientation entre deux éléments, énergie transférée entre deux éléments, informations et matériels échangés entre deux éléments). Selon cette approche, les auteurs proposent un modèle standard pour les ICD. Les vérifications et les validations des interfaces peuvent être effectuées par des règles définies manuellement. Cette approche est surtout applicable sur les systèmes matériels plutôt que logiciels.

Pajares *et al*. considèrent que les ICD sont aussi une source d'informations tout comme les exigences fonctionnelles (Pajares, M. *et al*., 2010). Les ICD doivent de même inclure des informations sur les exigences, comme par exemple la capacité maximale d'un bus ou la limite de charge du traitement (*processing load*). Selon ces chercheurs, un ICD doit intervenir dans toutes les phases du cycle de développement: il est utile pour la définition des exigences et pertinent pour la définition des tests d'intégration et des tests logiciels. Un méta-modèle est proposé comme solution. Cette approche gère les ICD en intégration avec un outil commercial (IBM) pour la gestion des exigences.

D'autres travaux sont basés sur des approches de modélisation pour extraire les ICD. Une recherche a été menée par Sergent *et al*. pour supporter la génération des ICD en utilisant le langage SysML (Le Sergent, T., & Le Guennec, A., 2014). Leur approche exige que le modèle conçu en SysML contienne l'architecture topologique du système à modéliser (fonctionnelle, logique, physique) et les informations sur les données échangées entre les sous-composants du système. Ils ont proposé de gérer les données échangées par des tables pour faciliter l'édition, la mise à jour et la manipulation des propriétés fonctionnelles ou logiques de ces données. Les tables peuvent être importées ou exportées par des fichiers .csv.

Les informations relatives aux interfaces sont associées au modèle en profitant du mécanisme d'annotation. Ces dernières sont liées aux connecteurs et aux ports du modèle SysML.

1.4.2 Standard AS5658

AS5658 - *Platform/Subsystem Common Interface Control Document Format* est un standard qui définit un format commun pour les documents d'interfaces des plateformes/sous-systèmes. Ce standard est proposé par la *Society of Automotive Engineers (SAE) Aerospace* suite à une demande de définition d'un standard comparable au standard AS5609 (*Aircraft/Store Common Interface Control Document Format*) proposé aussi par la SAE et adopté par l'industrie avionique militaire.

L'objectif du standard AS5658 est d'assister et harmoniser la standardisation d'interfaces des sous-systèmes et de fournir un format commun de document pour le développement et la comparaison des ICD (SAE International, 2009). Ce standard définit le format éditorial et les politiques nécessaires pour la publication d'un ICD. Il propose un format structuré de document pour définir et documenter les interfaces physiques, logiques et environnementales. Il permet une identification facile des informations techniques dans des sections similaires du document.

Le format du document proposé est divisé en deux volumes : le volume 1, qui couvre les interfaces matérielles (comme les interfaces électriques, mécaniques, aérodynamiques, ...) et le volume 2, qui couvre les interfaces logiques.

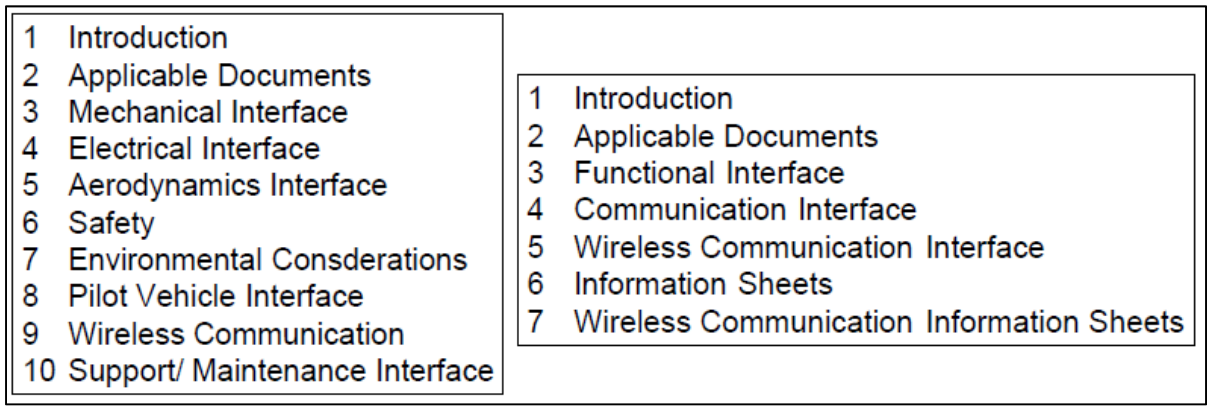


Figure 1.6 Sections du volume 1 (à gauche) et du volume 2 (à droite) du standard AS5658 (SAE International, 2009)

Dans chaque volume, le standard identifie les sections (figure 1.6) et les sous-sections à détailler. Par exemple, pour le volume 2, l'auteur doit décrire:

- les interfaces fonctionnelles: les exigences fonctionnelles de chaque donnée ou bit spécifié, la définition des paramètres, les états et les modes du sous-système, la description des évènements et les séquences des évènements. En plus le standard recommande la présentation des schémas générés par des outils de description de la nomenclature ou de l'architecture du système;
- les interfaces de communication: les protocoles implémentés et les structures et formats des données;
- les interfaces de communication sans fil: les protocoles implémentés ou toute restriction ou option utilisée dans la communication sans fil. On doit aussi y définir les représentations formelles et structurelles des messages ou données transmises à travers la communication sans fil.

Toutes les sections décrites dans le standard doivent être conservées même si elles ne sont pas applicables au système en question. Si par exemple un système ne possède pas d'interfaces physiques, le volume 2 doit exister mais il faut mentionner qu'il n'est pas applicable.

Ce standard vise à générer un seul ICD d'un sous-système pour différents manufacturiers. Il standardise le format de l'ICD dans le but de faciliter l'intégration du sous-système dans des nouvelles plateformes ou dans des plateformes existantes. Le standard explique aussi la position de l'ICD dans le contexte de l'ensemble des documentations d'un projet comme présenté sur la figure 1.7.

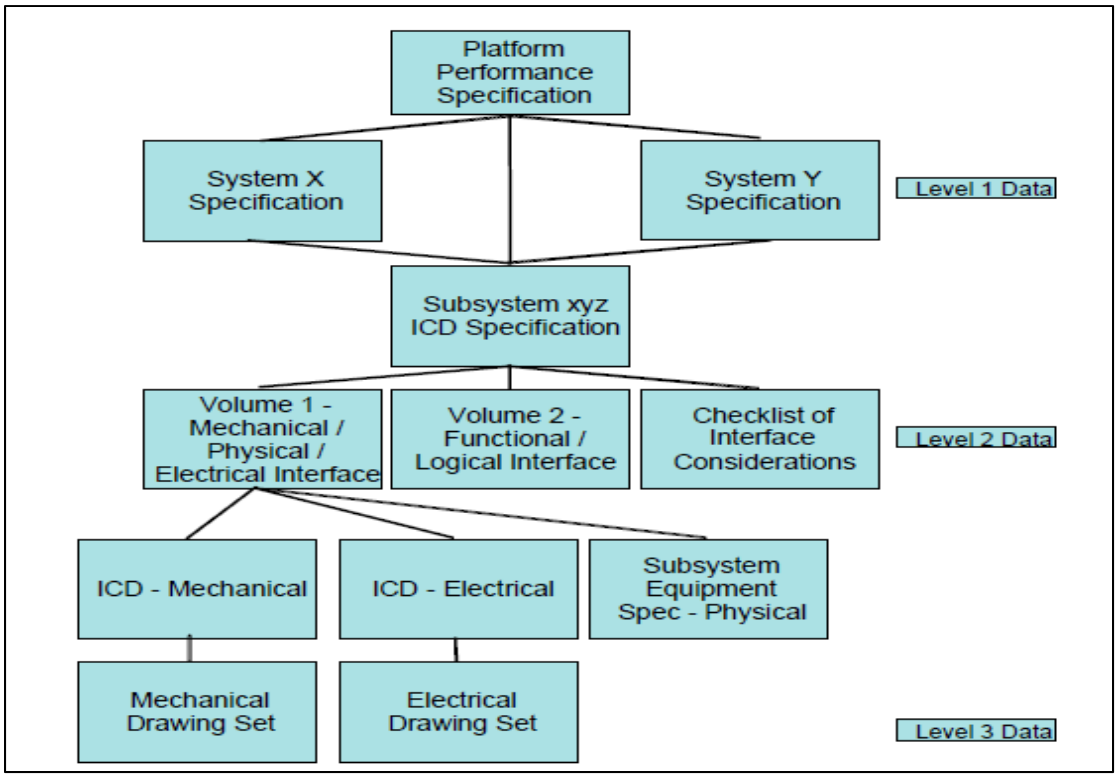


Figure 1.7 Position de l'ICD dans le contexte de l'ensemble des documentations (SAE International, 2009)

Dans notre projet, nous n'avons pas utilisé ce standard parce qu'il est très exhaustif, contient beaucoup de détails et nos partenaires industriels ne l'utilisent pas.

1.5 Conclusion

Dans cette section, une description des différentes architectures et standards normalisant les systèmes avioniques a été réalisée. Nous avons décrit les principaux composants et caractéristiques du langage AADL. Nous avons revu quelques travaux touchant le sujet de la gestion des ICD afin de comprendre ce qu'est un ICD et tenter de comprendre ce qu'il doit contenir. Chacun des travaux cités propose une approche ou une méthode pour identifier les interfaces et structurer les ICD. Un de ces travaux propose une décomposition hiérarchique du système en sous-systèmes selon leurs fonctionnalités pour identifier les interfaces et les décomposer en quatre types différents selon leurs caractéristiques. La décomposition des interfaces en quatre types est plus applicable à des systèmes matériels que logiciels. Mais la

décomposition du système en sous-systèmes selon leurs fonctionnalités reste une idée à considérer dans le cas de systèmes logiciels avioniques. Le deuxième travail de recherche cité (Pajares M. *et al*., 2010) insiste sur le fait que les ICD doivent contenir des informations sur les exigences et doivent intervenir dans tous le cycle de développement du logiciel. Mais la solution proposée est intégrée avec un outil commercial et notre intérêt est fixé dès le début sur les outils libres. Sergent *et al*. ont adopté une méthodologie de modélisation des systèmes pour en extraire les ICD mais utilisent SysML comme langage de modélisation. Nous utilisons dans notre mémoire le langage AADL, conçu initialement pour les systèmes avioniques et ayant une suite d'outils libres. À la fin de ce chapitre, un standard pour les ICD des systèmes avioniques a été présenté, qui est le standard AS5658. Nous n'utilisons pas ce standard mais nous en inspirons car il est intéressant mais trop complexe à mettre en œuvre.

CHAPITRE 2

Étude de cas

Dans ce chapitre, nous présentons à travers une étude de cas la modélisation d'un système IMA avec le langage AADL. D'abord, l'étape de collecte d'information est présentée dans les deux contextes, fédéré et IMA, afin de comprendre leurs différences en termes d'interfaces. Ensuite, l'étape de modélisation est effectuée en se basant essentiellement sur deux travaux de recherches. Les modèles proposés dans ces travaux sont étudiés pour valider leur degré de couverture des aspects des standards ARINC 653 et ARINC 664. Nous proposons par la suite notre propre modèle en détaillant les principaux composants et structures et en effectuant les validations nécessaires. Nous terminons ce chapitre par une évaluation des capacités d'AADL dans la modélisation des systèmes IMA en nous concentrant spécifiquement sur sa capacité à capturer et modéliser des informations relatives aux interfaces.

2.1 Collecte des informations relatives aux interfaces

Dans cette section nous présentons la démarche suivie pour collecter les informations relatives aux interfaces et les ressources/documents auxquels nous nous sommes référés dans les deux contextes (IMA et fédéré). Nous illustrons le processus de recherche d'information à travers deux exemples d'équipements avioniques: *l'Instrument Landing System* (ILS) dans le contexte fédéré et l'ADIRU dans le contexte IMA. En effet, les informations collectées dans cette étape sont utiles par la suite pour l'étape de modélisation.

Dans cette étape, nous nous intéressons seulement à l'étude de cas d'un système IMA. Le contexte fédéré n'est pas couvert dans notre étude parce qu'une autre équipe du projet CRIAQ AVIO-506 s'occupe de la modélisation des systèmes fédérés. Cependant, il est utile de couvrir la collecte d'informations dans les deux contextes, d'une part pour expliquer et comprendre la différence entre l'échange de données dans chacun de ces deux contextes et d'autre part parce que les documents qui couvrent le contexte fédéré sont plus accessibles,

comme par exemple la série 700 des documents ARINC. Ces documents nous ont permis de comprendre l'interaction et la communication entre les différents équipements même s'ils vont être modélisés dans un contexte IMA.

2.1.1 Contexte fédéré

Dans le cas d'une architecture fédérée, les systèmes avioniques sont constitués de micro-ordinateurs (ou calculateurs) permettant de remplir différentes fonctionnalités. D'un point de vue matériel, ces micro-ordinateurs sont des boîtiers physiques ayant des entrées et des sorties qui assurent l'échange de données avec d'autres boitiers similaires.

Les interfaces dans une architecture fédérée sont des interfaces physiques qui présentent les points de liaison physique entre les divers sous-systèmes. Une description de ces interfaces doit préciser les interconnexions et les flux de données entre les boites (Louadah H., 2016). La communication et l'échange de données sont effectués à travers des bus physiques (comme le bus ARINC 429). Nous illustrons la procédure de collecte d'information dans un contexte fédéré par un exemple d'équipement avionique simple mais représentatif qui est l'exemple de l'ILS.

Un ILS est un système qui traite les signaux radio reçus via des antennes et fournit des données au pilote et au système de contrôle automatique de vol pour maintenir la trajectoire idéale de descente de l'aéronef lors de l'atterrissage. Les informations fournies par l'ILS sont l'écart latéral par rapport à l'axe de la piste et l'écart vertical ou plan de descente (*glide slope)*. L'annexe I de ce mémoire contient plus de détails sur cet équipement.

La spécification de l'ILS est décrite dans le standard ARINC 710, qui définit les fonctionnalités désirées et les exigences à respecter pour le développement et l'installation de ce dispositif. Ce document et le standard ARINC 429 ont été nos principales sources d'information sur lesquelles nous nous sommes basés pour l'extraction des ICD pour les systèmes fédérés.

Dans l'objectif de comprendre toutes les interconnexions et les équipements qui interagissent avec l'ILS et ses entrées/sorties, la figure 2.1 illustre sommairement les connexions de l'ILS

avec les autres composants. Sur cette figure, les connexions sont énumérées par leur numéro de broches (MPxy). Les ports ARINC 429 (encadrés pointillés) sont ceux qui nous intéressent. Les autres types d'interfaces sont hors de portée de notre mémoire.

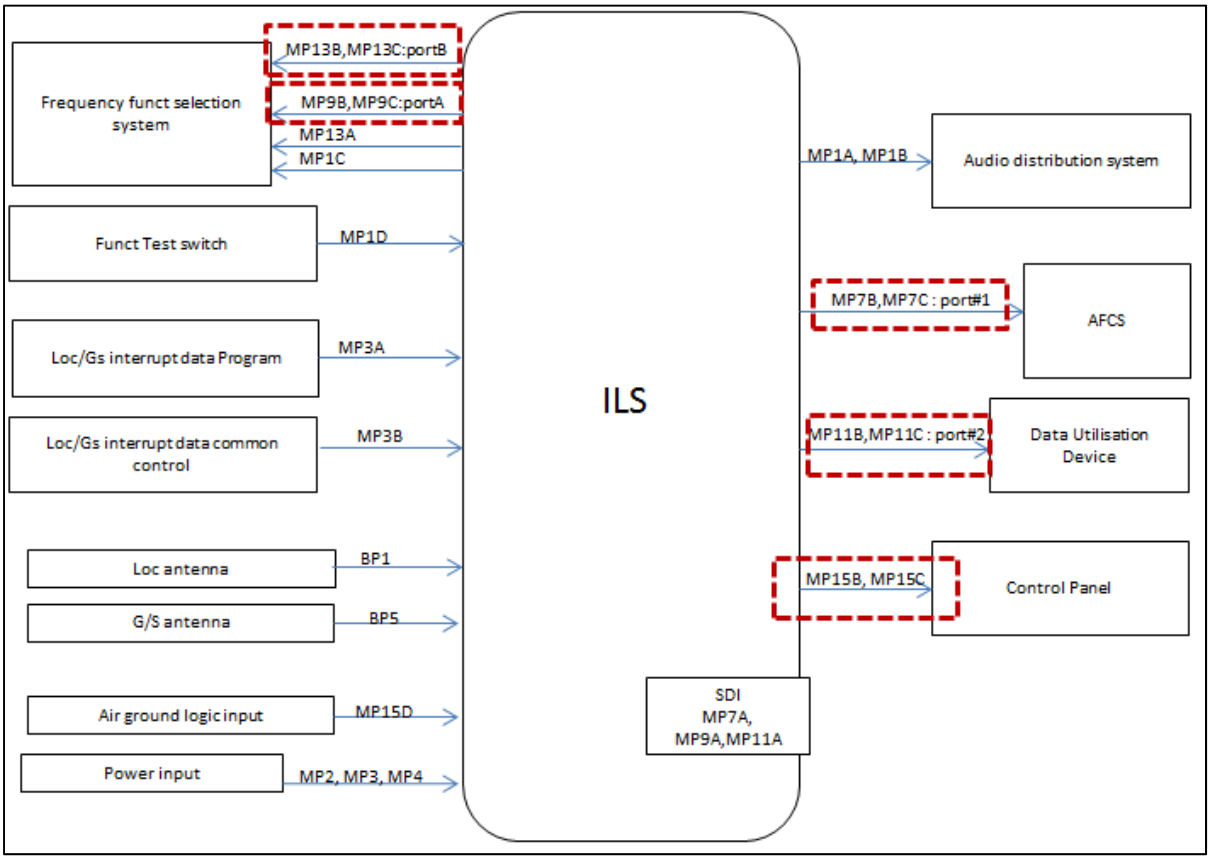


Figure 2.1 Schéma des interconnexions de l'ILS avec les autres sous-systèmes

L'annexe 1 du standard ARINC 710 (AEEC, 1997) décrit la plaquette de branchement et fournit des informations sur le nombre de broches, le câblage et le positionnement des connecteurs dans le boîtier. L'annexe (*attachement*) 2 du même standard décrit les interconnexions et l'échange de données entre l'ILS et d'autres composants. Les informations comme le nombre de ports, leurs caractéristiques physiques et électriques et les données échangées à travers ces ports sont intéressantes dans la description des ICD. Les deux annexes résument les interactions de l'ILS avec son environnement et suscite un bon point de départ pour l'extraction d'informations. D'autres informations utiles sont présentes sous forme de notes, de tableaux ou texte dans d'autres sections du standard ARINC 710. Ce

document n'est pas toujours suffisant car il y manque parfois des informations, par exemple vers quelle destination les sorties seront acheminées. Le standard ARINC 429 contient des informations complémentaires et pertinentes puisque l'ILS échange des mots ARINC 429. D'une manière générale, ce standard fournit des informations sur les flux de données entre les équipements qui coopèrent ensemble via le bus ARINC 429. De plus, ce standard décrit la structure et le format des mots de 32 bits qui circulent sur ce bus. La figure 2.2, extraite du document de spécification ARINC 429 P1-18 (2012), définit la structure générale d'un mot ARINC 429 pour un codage *Binary Coded Decimal* (BCD). Le standard décrit en détail les différents champs du mot et donne des exemples et des explications sur le codage et décodage des données incluses dans ce mot.

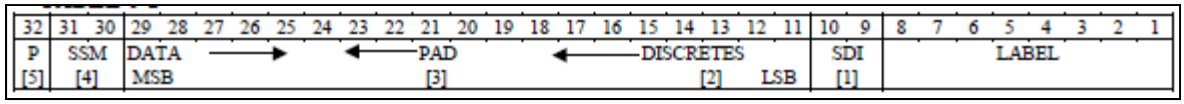


Figure 2.2 Structure générale d'un mot ARINC 429 pour le codage BCD. Figure extraite de (AEEC, 2012)

À ce stade, nous avons assimilé les exigences fonctionnelles et les principales fonctionnalités réalisées par l'ILS. Les différentes connexions sont décrites dans un schéma sommaire d'interconnexions visant à présenter globalement l'ILS et ses interfaces de communication avec l'extérieur. Nous avons ainsi collecté les informations suffisantes pour l'étape de modélisation. Une autre équipe du projet AVIO-506 a modélisé cet équipement, puisque la modélisation des équipements fédérés est hors de portée de notre mémoire. Notre intérêt se concentre sur l'architecture IMA, mais nous allons souligner les différences entre ces deux architectures en termes d'interfaces.

2.1.2 Contexte IMA

Contrairement à l'architecture fédérée, l'architecture IMA est basée sur des modules matériels génériques, regroupant des ressources partagées par plusieurs applications, réparties dans une ou plusieurs partitions. Ces modules standardisés sont destinés à héberger des applications différentes et les liaisons assurant la communication entre les partitions sont virtuelles.

Dans une architecture IMA, les interfaces physiques ne sont pas pertinentes, seulement les connexions virtuelles et les données d'entrées/sorties sont considérées (Louadah H., 2016). Plusieurs documents sont utiles pour analyser et comprendre l'aspect fonctionnel et les spécifications des interfaces:

- Tout document de spécification de la série ARINC 700, malgré qu'il soit décrit dans le cadre du contexte fédéré, reste une source fiable pour comprendre les exigences et l'aspect fonctionnel d'un équipement avionique. Les broches (*pins*), le câblage et les interfaces physiques reliant l'équipement avec les autres ne sont pas utiles en IMA puisque dans ce contexte, il s'agit d'applications logicielles partitionnées. Les communications entre les partitions sont assurées par des liaisons virtuelles à travers un réseau.
- Le document ARINC 653 décrit l'architecture logicielle et matérielle d'un module ARINC 653.
- Le standard ARINC 664-P7, qui normalise le réseau AFDX, décrit en détail la structure des trames AFDX. Il spécifie les propriétés et les caractéristiques nécessaires pour les liaisons virtuelles et le transfert de données sur ce réseau. Ce standard sera ainsi notre référence de base dans les prochaines étapes de modélisation du réseau AFDX.

Dans notre étude de cas, nous avons choisi de modéliser l'ADIRU (*Air Data Inertial Reference Unit*) dans un contexte IMA. L'ADIRU est un composant du système de référence inertielle anémobarométrique (*Air Data Inertial Reference System,* ADIRS) (AEEC, 2001). Il fournit des informations sur l'air (pression, vitesse, température, altitude, …) et sur les référents inertiels (l'attitude, l'accélération, ...). Il fournit deux fonctionnalités essentielles, souvent considérées comme deux sous-systèmes: l'IR (*Inertial Reference*) et l'ADR (*Air Data Reference*). L'annexe I présente de plus amples informations sur ce composant.

Tout comme dans le cas d'une architecture fédérée, nous avons créé le schéma d'interconnexion de l'ADIRU avec les autres sous-systèmes (présenté dans l'annexe II). Des

informations comme la désignation des broches et des connecteurs physiques sont ignorées dans ce contexte. Les spécifications des interfaces doivent décrire:

- les applications ou les partitions logicielles (leur contraintes temporelles d'exécution, les ressources allouées);
- les ports (leur type: *Sampling/Queuing*, leurs caractéristiques);
- les messages échangés (structure, taille, fréquence, …);
- les liaisons virtuelles: leurs caractéristiques, contraintes et paramètres de configuration;
- le réseau AFDX.

Dans cette section, la procédure de collecte d'information a été présentée dans les deux contextes, soit fédéré et IMA. Les informations pertinentes et les sources d'informations utiles pour la description des interfaces sont spécifiées. Cette étape nous prépare à l'étape de modélisation qui fait l'objet de la prochaine section.

2.2 Modèle Hugues et Delange[1] (annexe AADL ARINC 653)

Hugues et Delange (Hugues, J. & Delange, J., 2015) ont proposé un modèle pour l'ADIRU en utilisant le langage de modélisation AADL. Leur but était d'explorer les capacités de ce langage pour la modélisation et la validation des architectures avioniques dans le contexte de la sureté fonctionnelle et la sécurité des systèmes critiques. Ils ont modélisé l'ADIRU du Boeing 777-2H6ER pour reproduire le cas de panne de l'avion qui s'est écrasé pendant le vol 124 de la *Malaysia Airlines* en 2005. Ils ont simulé la panne des accéléromètres de l'avion pour pouvoir valider la sureté des fonctions de l'ADIRU. Leur recherche se concentre sur la sûreté du système et la tolérance aux pannes. Leur modèle ne couvre pas toutes les fonctionnalités de l'ADIRU et se limite aux fonctionnalités reliées aux accéléromètres qui ont été la cause de l'accident. Ces travaux nous ont fourni un modèle duquel nous nous sommes inspirés pour modéliser les concepts ARINC 653.

[1] https://github.com/OpenAADL/AADLib/tree/master/examples/adiru

2.2.1 Modèle

Les travaux menés par les deux chercheurs se basent sur l'annexe ARINC 653 d'AADL pour modéliser les aspects de l'architecture IMA. Cette annexe définit les patrons de modélisation en AADL et les propriétés spécifiques pour satisfaire aux exigences d'une plateforme IMA:

- Un module ARINC 653 est représenté en AADL par le composant **`processor`**, qui représente le processeur physique. Ce dernier doit contenir les processeurs virtuels qui représentent les partitions (figure 2.3). Le processeur décrit la séparation temporelle entre les partitions logicielles en assignant la durée du *time slot* pour chaque partition. Ce critère est décrit par la propriété AADL **`Module_Schedule.`**

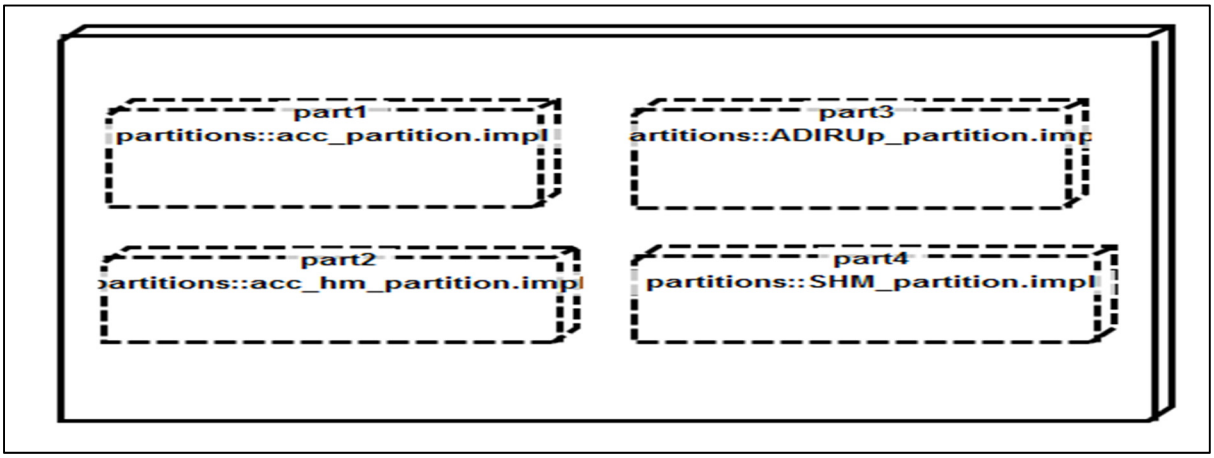


Figure 2.3 Processeur physique contenant des processeurs virtuels représentant le 'runtime execution' des partitions (Hugues, J. & Delange, J., 2015)

- Une partition ARINC 653 possède deux caractéristiques principales:
  1- L'application logicielle: elle est considérée comme étant un processus ARINC 653 qui contient des fils d'exécution et est représentée en AADL par le composant **`process`**.
  2- Le *runtime execution*: elle est similaire à une ressource dans laquelle la partition logicielle est exécutée. Le *runtime execution* d'une partition est représenté par le composant **`virtual processor`** d'AADL contenu dans le module ARINC 653 (qui est le composant **`processor`** en AADL).

L'application logicielle est reliée à la *runtime execution* par l'association **`Actual_Processor_Binding`**. Un processus (ou application logicielle) doit être associé à un segment de mémoire par l'association **`Actual_Memory_Binding`** pour assurer la séparation spatiale entre les applications. En AADL, il faut donc que le composant mémoire soit segmenté en des sous-composants mémoires et que chaque segment de mémoire soit alloué à un maximum d'un seul processus. La figure 2.4 présente tous les composants (logiciels et matériels) utilisés dans la modélisation et schématise globalement comment le modèle est conçu par ces deux chercheurs. Chaque processus qui représente une application logicielle est associé à une mémoire et à une *runtime execution* (ou à des processeurs virtuels). Les accéléromètres sont reliés au processus responsable du traitement des données reçues. Il en résulte des communications entre les différents processus, qui ne sont pas présentes dans ce schéma pour le rendre plus lisible. Ces communications sont décrites par la clause **`connections`** de AADL, où chaque connexion présente les ports d'entrées/sorties à travers lesquels les données sont acheminées.

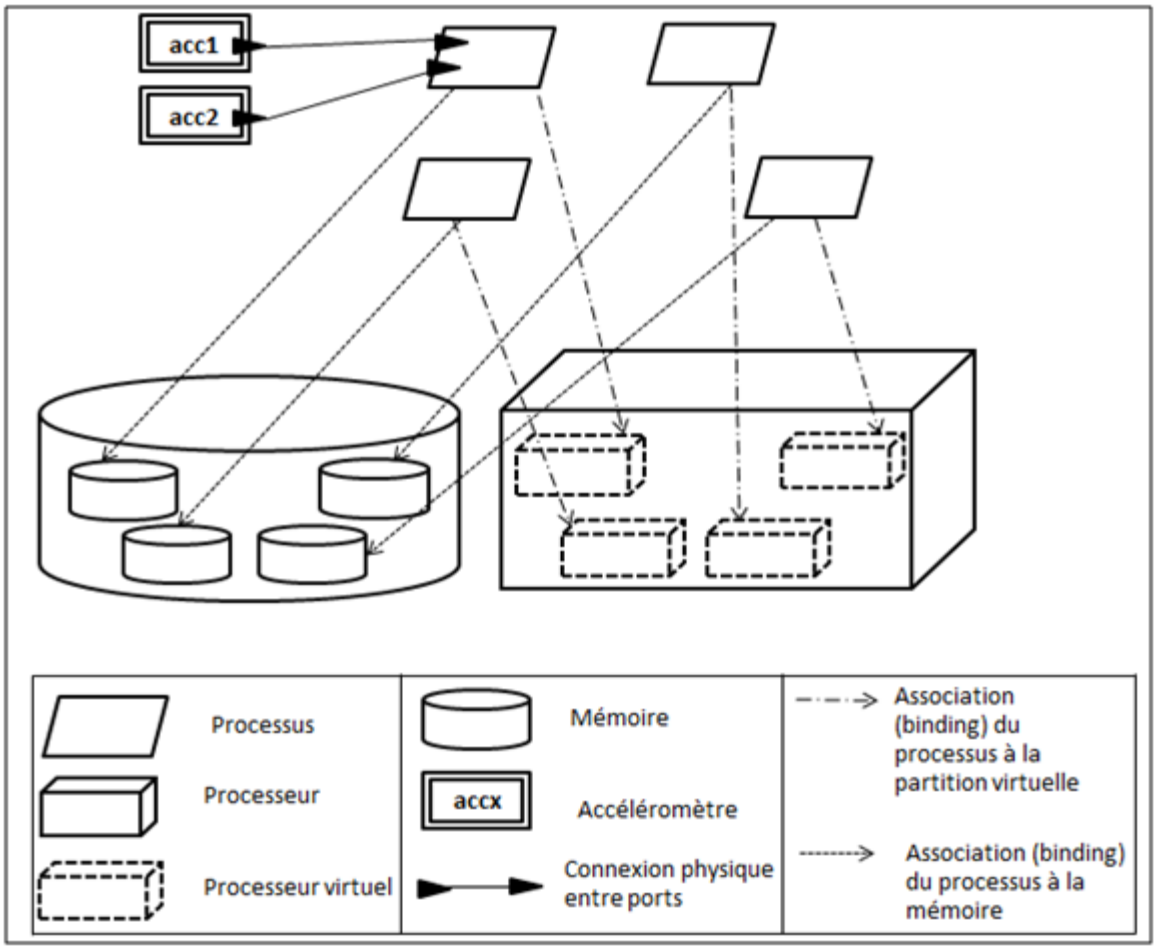


Figure 2.4 Schéma global et simplifié du modèle proposé par Hugues et Delange

2.2.2 Validation RESOLUTE de la partie ARINC 653

Pour valider la conformité du modèle AADL aux exigences de l'architecture IMA, le langage de validation RESOLUTE a été utilisé par les deux chercheurs. La communauté AADL le propose sous forme d'extension spécifique par le mécanisme d'annexe AADL pour analyser le modèle, naviguer ses composants et extraire ses propriétés avec un langage de requête convivial et facile à utiliser (Gacek A. *et al*., 2014). RESOLUTE permet à l'utilisateur d'écrire des règles d'analyse et de validation pour répondre à des exigences spécifiques sans avoir à développer un engin de règles ou comprendre le méta-modèle d'AADL (Hugues, J. & Delange, J., 2015). En analysant le modèle, RESOLUTE produit une représentation graphique et hiérarchique des résultats de validation.

Hugues et Delange ont défini une bibliothèque de règles pour valider la conformité de leur modèle avec l'annexe ARINC 653. Les principales règles ajoutées et qui nous intéressent dans notre étude sont:

- chaque processus doit être lié à une mémoire et à un processeur virtuel;
- chaque processeur doit avoir les propriétés du *Health monitoring* pour la sureté fonctionnelle et de *Scheduling* pour le partitionnement temporel (`ARINC653::HM_Error_ID_Levels, Module_Schedule, Module_Major_Frame`);
- chaque processeur virtuel doit être inclus dans un processeur physique.

Les chercheurs ont aussi comme objectif de générer le code source des partitions à partir du modèle. Conséquemment, les règles qu'ils ont spécifiées sont adaptées plus au besoin de la génération de code, c'est-à-dire valider le modèle pour que la génération de code se fasse correctement. La figure 2.5 est le résultat de la validation de leur modèle par rapport aux exigences de l'annexe ARINC 653. Ce résultat est affiché d'une manière hiérarchique du plus haut niveau (celui du modèle) vers le plus bas niveau de vérification (celui des composants). Dans ce cas, la vérification est groupée par catégorie de composants comme les processus, les processeurs et les processeurs virtuels. Comme présenté, toutes les règles définies sont vérifiées.

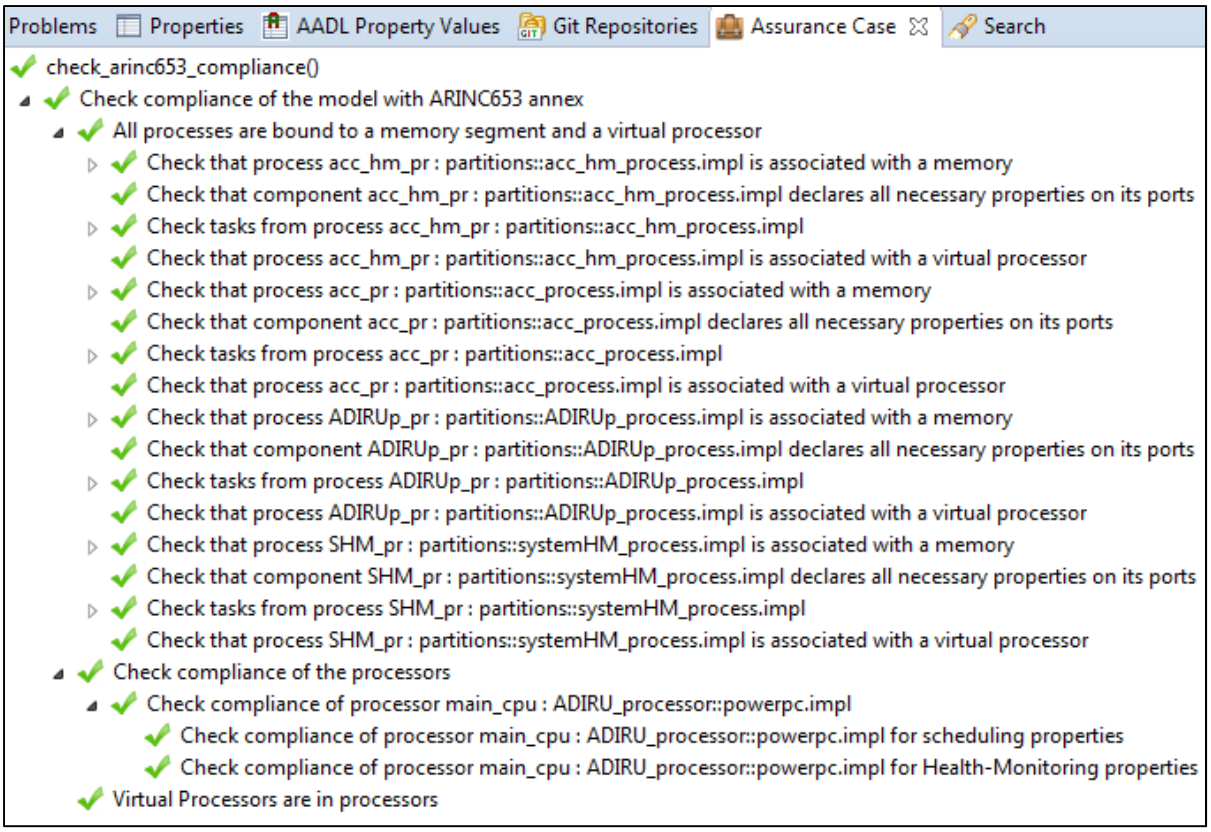


Figure 2.5 Validation de la conformité du modèle de Hugues et Delange aux exigences de l'annexe ARINC 653

Si une de ces règles n'est pas vérifiée, la validation RESOLUTE affiche l'échec de la validation en soulignant le cas qui n'est pas valide. L'exemple représenté sur la figure 2.6 montre un cas d'échec de validation RESOLUTE après avoir modifié le code et commenté certaines propriétés du modèle afin de montrer l'affichage dans un cas de non-conformité. Le message d'erreur signale que le processus en question (en rouge) doit être associé à une mémoire.

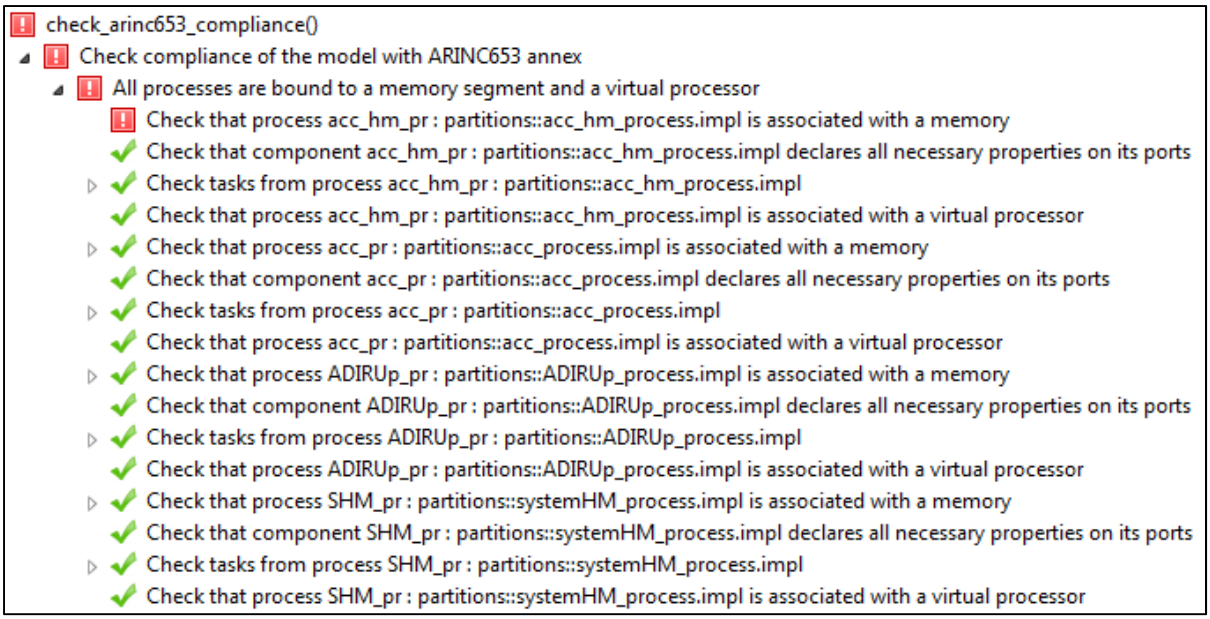


Figure 2.6 Résultat de la validation RESOLUTE dans un cas de non-conformité

Le modèle étudié dans cette section est une véritable source d'inspiration pour notre étude. D'une part, il nous a fourni une idée sur la modélisation d'un sous-système dans un contexte IMA en utilisant l'annexe ARINC 653 d'AADL. D'autre part, nous avons expérimenté l'outil RESOLUTE et nous avons montré son importance dans la vérification et la validation d'un modèle AADL.

Cependant, malgré sa pertinence, ce modèle ne présente pas toute l'architecture IMA. Les communications à travers le réseau AFDX ne sont pas modélisées. Dans la section suivante, nous décrivons un autre exemple de modèle AADL de l'ADIRU qui met l'accent sur l'architecture AFDX et les communications entre les partitions IMA.

2.3 Modèle Khoroshilov[2]

Le projet de Khoroshilov (Khoroshilov A., 2012) a pour objectif de proposer une nouvelle annexe officielle à la communauté AADL pour supporter la modélisation des réseaux AFDX.

[2] https://github.com/khoroshilov/aadl-networking-refmodel

2.3.1 Modèle

Dans ce modèle, tout le système IMA est décrit par:

- une couche logicielle, qui modélise les processus, les tâches et leurs connexions;
- une couche matérielle, qui décrit la plateforme matérielle comme les processeurs, les commutateurs et les bus physiques;
- les processeurs virtuels, un pour chaque partition;
- un bus virtuel, qui représente le réseau AFDX.

La couche matérielle décrit l'architecture physique des modules et du réseau AFDX. Un module est considéré comme un *Core Processing Module* (CPM) qui forme une plateforme matérielle offrant des capacités de calcul génériques. C'est la plateforme dans laquelle les applications (partitions logicielles) sont hébergées pour être exécutées. Les modules sont modélisés par le composant **`system`** du langage AADL. La figure 2.7 fournit un peu plus de détails sur la description architecturale du composant CPM. Un CPM est composé d'un terminal AFDX (*end system*) et d'une *Central Processing Unit* (CPU) pour le traitement des données. Les deux sous-composants sont reliés par un bus PCI (*Peripheral Component Interconnect*) pour que ces composants puissent communiquer ensemble.

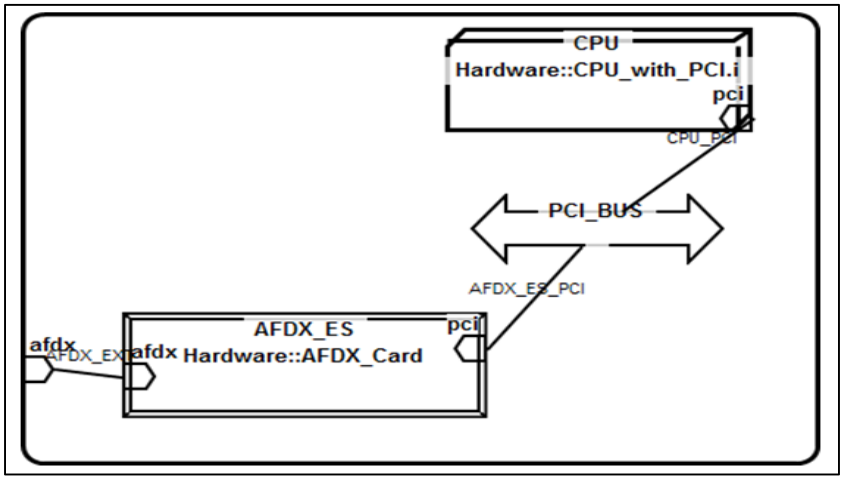


Figure 2.7 Diagramme du composant CPM en AADL

Le réseau AFDX physique est composé de commutateurs et de bus physiques reliant les différents modules. Les commutateurs sont simplement modélisés par le composant **`device`** puisque l'architecture interne d'un commutateur n'est pas intéressante dans cette étude de cas. Cependant, des propriétés utiles sont assignées aux commutateurs. Les bus physiques (*wire*) sont des supports physiques de communication, modélisés simplement par le composant **`bus`**.

La structure globale du système IMA est présentée d'une manière simplifiée sur la figure 2.8. Contrairement au modèle précédent (Hugues et Delange), les partitions (**`virtual processor`**) sont définies en dehors des processeurs physiques. Elles sont reliées aux processeurs d'exécution par une association (*binding*) avec la propriété **`Actual_Processor_Binding.`** Chaque processus de la couche logicielle est aussi relié à une partition virtuelle par la même propriété. La configuration des VL est effectuée par la propriété **`Actual_Connection_Binding`** reliant les ES, les bus et les commutateurs au VL.

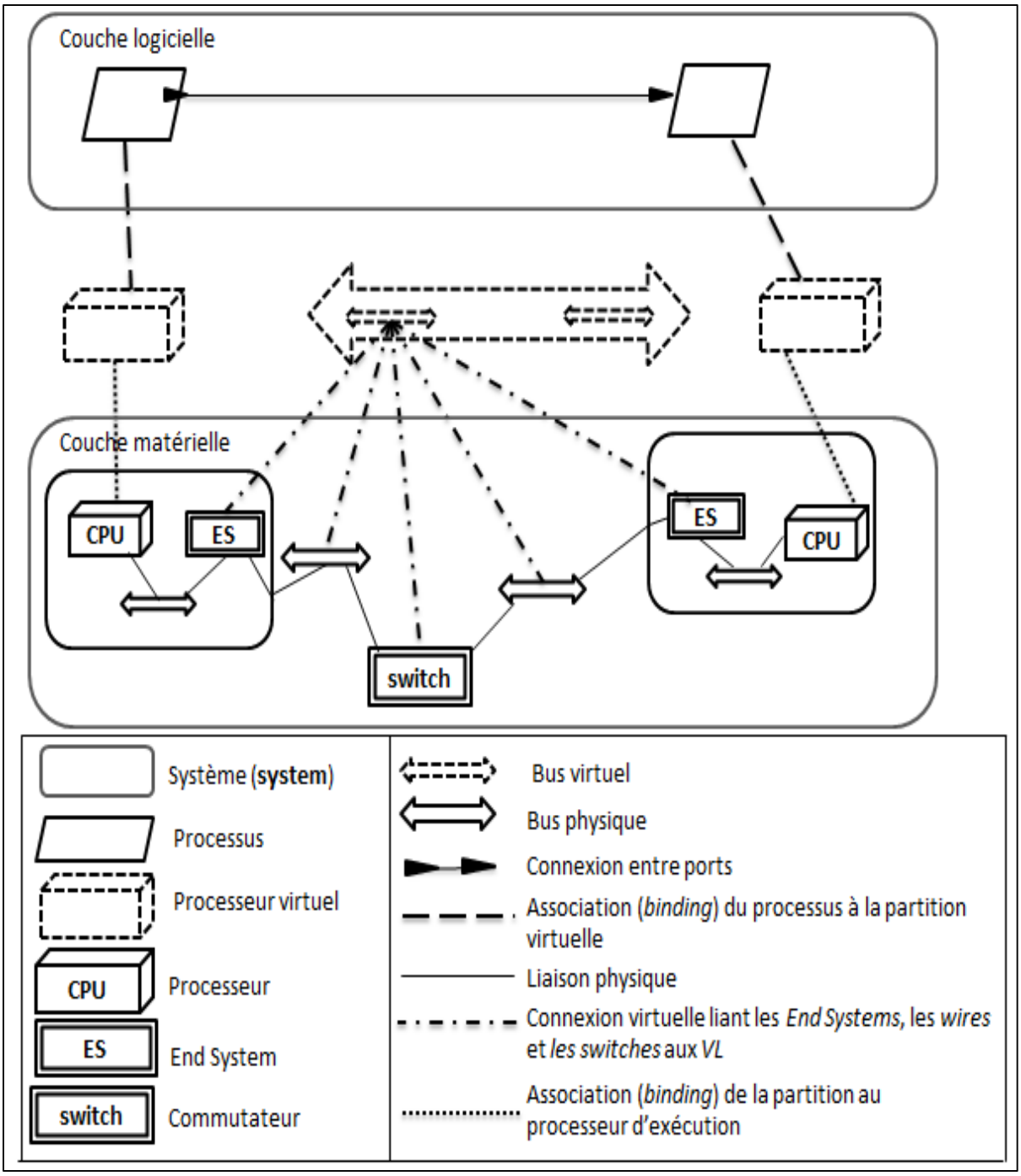


Figure 2.8 Modèle général et simplifié du réseau AFDX proposé par Khoroshilov

Khoroshilov a défini un nouvel ensemble de propriétés (*property set*) pour le réseau AFDX nommé **`AFDX_Properties`**, où il a spécifié des propriétés reliées aux VL, aux partitions et aux commutateurs. Certaines propriétés sont obligatoires et d'autres sont

optionnelles selon l'exigence du standard ARINC 664. Ces propriétés sont décrites dans les tableaux suivants.

Tableau 2.1 Propriétés reliées aux *Virtual Links*

Propriétés	Description
`BAG: TIME applies to (virtual bus);`	Bandwidth Allocation Gap: cette propriété est obligatoire pour chaque VL
`Lmax: AADLINTEGER 64 Bytes .. 1518 Bytes units SIZE_UNITS applies to (virtual bus);`	Taille maximale d'une trame dans le VL: propriété obligatoire
`Lmin: AADLINTEGER 64 Bytes .. 1518 Bytes units SIZE_UNITS => 64 Bytes applies to (virtual bus);`	Taille minimale d'une trame dans le VL: propriété optionnelle
`SkewMax: TIME applies to (virtual bus);`	Temps maximal entre deux trames redondantes: propriété obligatoire
`VLID: AADLINTEGER 0..65535 applies to (virtual bus);`	Identifiant d'un VL: propriété optionnelle, l'identifiant doit être unique dans tout le réseau AFDX

Tableau 2.2 Propriétés reliées aux partitions

Propriétés	Description
`UDP: AADLINTEGER 1 .. 65535 applies to (port);`	Port UDP: si c'est un port d'entrée, le numéro de port doit être unique pour tous les ports d'entrée d'une partition: propriété optionnelle
`PartitionID: AADLINTEGER 0 .. 255 applies to (virtual processor,process);`	L'adresse IP d'une partition: propriété optionnelle

Tableau 2.3 Propriétés reliées aux commutateurs

Propriété	Description

`VL_Route_Table : list of record (`
`  vl : reference (virtual bus);`
`  in_port : reference (bus access);`
`  out_ports : list of reference (bus access);`
`  jitter : TIME;`
`  priority : enumeration (high, low);`
`  accountingPolicy : enumeration (byte, frame);`
`  sharedAccountId : aadlstring;`
`  ) applies to (device, system);`
	Table de routage
	• *virtual link*: obligatoire
	• port input: optionnel
	• ports output: optionnel
	• *maximum allowed jitter*: obligatoire
	• priorité: obligatoire
	• politique: obligatoire
	• *sharedAccountId*: optionnel

2.3.2 Validation RESOLUTE de la partie ARINC 664

Pour valider la conformité de ce modèle au standard ARINC 664, nous avons défini de nouvelles règles spécifiques à ce besoin. Voici les validations implémentées:

- commutateurs: vérifier que les propriétés des tables de routage sont définies;
- chaque VL: vérifier que les propriétés **`BAG`**, **`Lmax`**, **`SkewMax`**, … sont spécifiées;
- vérifier qu'il existe une bijection des connexions entre les processus et les Virtual Links;
- vérifier que les VL sont configurés avec la propriété **`Actual_connection_binding`** qui lie les *ES*, *wires* et les *switches* aux VL.

Nous avons implémenté quelques-unes de ces règles en RESOLUTE sur le modèle de Khoroshilov pour expérimenter avec la capacité de RESOLUTE à traduire les règles que nous désirons ajouter. La figure 2.9 présente le résultat de la validation.

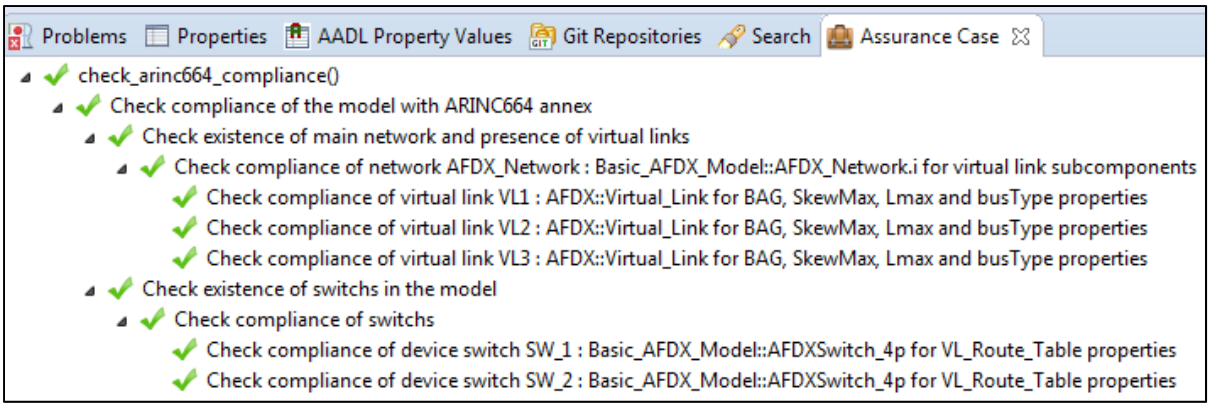


Figure 2.9 Validation RESOLUTE ARINC 664 du modèle de Khoroshilov

Tous les cas sont validés avec succès. Ces règles s'assurent que le modèle contient un composant **`virtual bus`** qui contient lui-même des *virtual links*, que ces *virtual links* possèdent les propriétés nécessaires comme **`BAG`**, **`SkewMax`**, **`Lmax`**, etc. De manière analogue, pour les commutateurs, nous avons vérifié que les propriétés des tables de routage sont définies pour chaque **`device`** de type **`switch`**.

2.4 Notre modèle d'ADIRU[3]

Après l'étude des deux modèles précédents, nous présentons dans cette section notre modèle d'un ADIRU en combinant l'architecture AFDX décrite par Khoroshilov avec les exigences ARINC 653 introduites par Delange et Hugues. Pour modéliser un système IMA représentatif et pertinent, il est souhaitable de présenter dans le même modèle toutes les spécifications ARINC 653 et ARINC 664. Dans la réalisation de notre modèle, nous visons donc à profiter de ces deux annexes: l'annexe officielle ARINC 653 et l'annexe ARINC 664 de Khoroshilov, une future annexe officielle.

La modélisation est effectuée avec la version la plus récente à ce jour du langage AADL qui est la version 2 en profitant de l'annexe officielle ARINC 653. L'outil de modélisation utilisé

[3] https://github.com/rimbendhaou/basic_afdx_adiru/tree/master/Afdx_Adiru_Model

dans ce projet est OSATE2[4], un logiciel libre qui bénéficie d'un ensemble d'outils intégrés facilitant la modélisation, la simulation et la validation (SEI, 2004).

Il est à noter que nous avons simplifié au maximum le système de l'ADIRU pour pouvoir le modéliser et le présenter comme étude de cas. Seules quelques fonctionnalités et quelques données représentatives sont considérées dans notre modèle.

Dans cette perspective, nous considérons seulement les éléments représentés sur la figure 2.10. Nous modélisons quelques partitions de l'ADIRU qui interagissent entre elles et qui échangent des données avec un système de gestion de vol (FMS) et un concentrateur de données (*Remote Data Concentrator*, RDC).

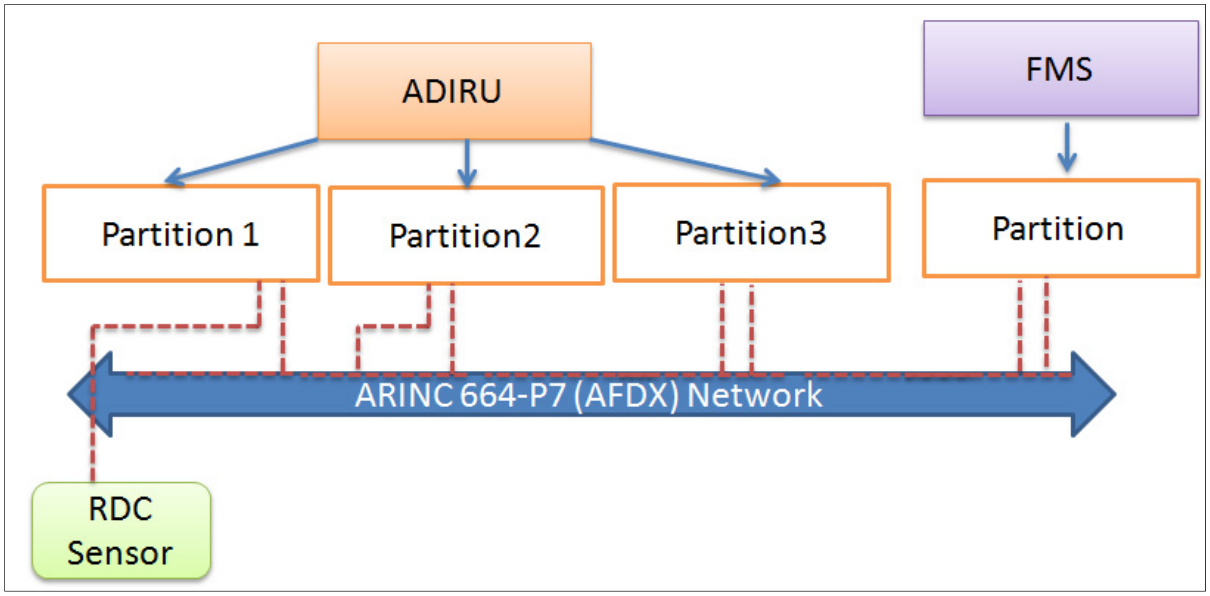


Figure 2.10 Schéma simplifié des éléments à modéliser

FMS: le système de gestion de vol est un système qui assiste l'équipage pendant un vol en lui délivrant des informations sur le pilotage, la navigation, la planification de vol, etc.

[4] http://osate.org/

RDC: un équipement qui supporte l'échange d'information entre les capteurs et actionneurs d'une part et la plateforme IMA d'autre part. Il s'interconnecte aux capteurs et concentre les données pour les délivrer aux calculateurs dédiés aux traitements de ces données.

2.4.1 Modèle

Nous avons fait deux tentatives pour modéliser l'ADIRU. En nous inspirant du modèle de Hugues et Delange, nous avons utilisé l'annexe ARINC 653 pour décrire la structure d'un système IMA (le principe des partitions, la séparation spatiale et temporelle, …) avec les propriétés existantes dans le langage AADL. En effet, ce modèle présente bien la politique de déploiement des applications logicielles dans la plateforme matérielle d'exécution. Mais en termes d'ICD, notre principal intérêt se focalise dans la communication et l'échange de données entre systèmes ou partitions. Dans un contexte IMA, la communication est assurée par un réseau Ethernet spécifique.

Donc dans notre deuxième tentative nous avons pris en compte les notions spécifiées dans le standard ARINC 664 partie 7 qui standardise le réseau Ethernet AFDX. Nous nous sommes fortement basés sur la proposition de Khoroshilov (Khoroshilov A., 2012) pour la modélisation du réseau AFDX.

Dans cette modélisation, nous avons séparé les différents composants du système IMA en couches pour avoir une description claire et cohérente de chaque couche du modèle. En pratique, cette séparation donne plus de flexibilité quant à la tâche de modélisation dans le cas où différentes équipes coopèrent ensemble en séparant leurs préoccupations. La description de l'architecture logicielle peut être faite sans dépendre de l'implémentation matérielle.

2.4.1.1 La couche matérielle

Pour décrire les différents composants de la couche matérielle présentée par le diagramme sur la figure 2.8, nous nous plaçons à un niveau d'abstraction assez élevé où la structure

interne des composants n'est pas explicitée. Cette couche décrit les modules, les commutateurs, les RDC et les bus physiques reliant ces composants (figure 2.11).

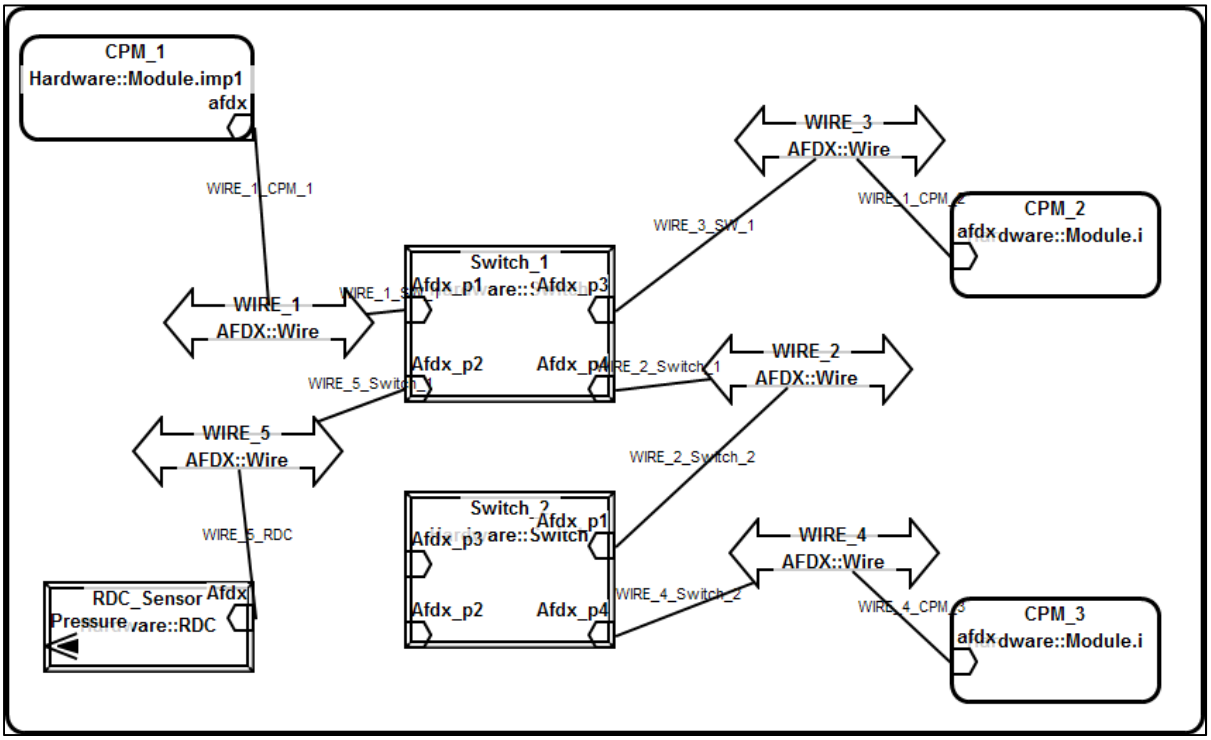


Figure 2.11 Modélisation de la couche matérielle
(présentation de haut niveau, sans détails)

La structure interne des modules CPM est modélisée de la même manière que celle décrite dans le modèle de Khoroshilov. Un RDC est modélisé par le composant AADL `device` puisque nous nous limitons à un niveau plus haut d'abstraction de son architecture interne. Il est relié par un bus physique à un commutateur qui le relie au reste du réseau AFDX.

2.4.1.2 La couche logicielle

Cette couche décrit les composants logiciels qui sont les processus applicatifs composés chacun de plusieurs fils d'exécution (*threads*). Le niveau de la description des *threads* reste donc très abstrait puisque ce niveau de détail n'est pas pris en compte dans notre approche. Les ports d'entrées/sorties assurent la communication et l'échange de données entre les processus. Des propriétés AADL sont spécifiées pour les différents types de ports.

Pour les ports de type échantillonnage, la propriété **`Sampling_Refresh_Period`** spécifie le temps de rafraichissement des ports (pour les **`data ports`** ou les **`event data ports`**) comme montre l'exemple suivant:

```
P2_out1: out data port DataType::Inertial_Altitude
{ARINC653::Sampling_Refresh_Period => 25 ms;};
```

Pour le port de type fil d'attente (*Queuing*) la propriété **`Queue_Size`** spécifie la taille de la file d'attente d'un port:

```
Queue_Size: aadlinteger 0 .. Max_Queue_Size => 1
applies to (event port, event data port, subprogram access);
```

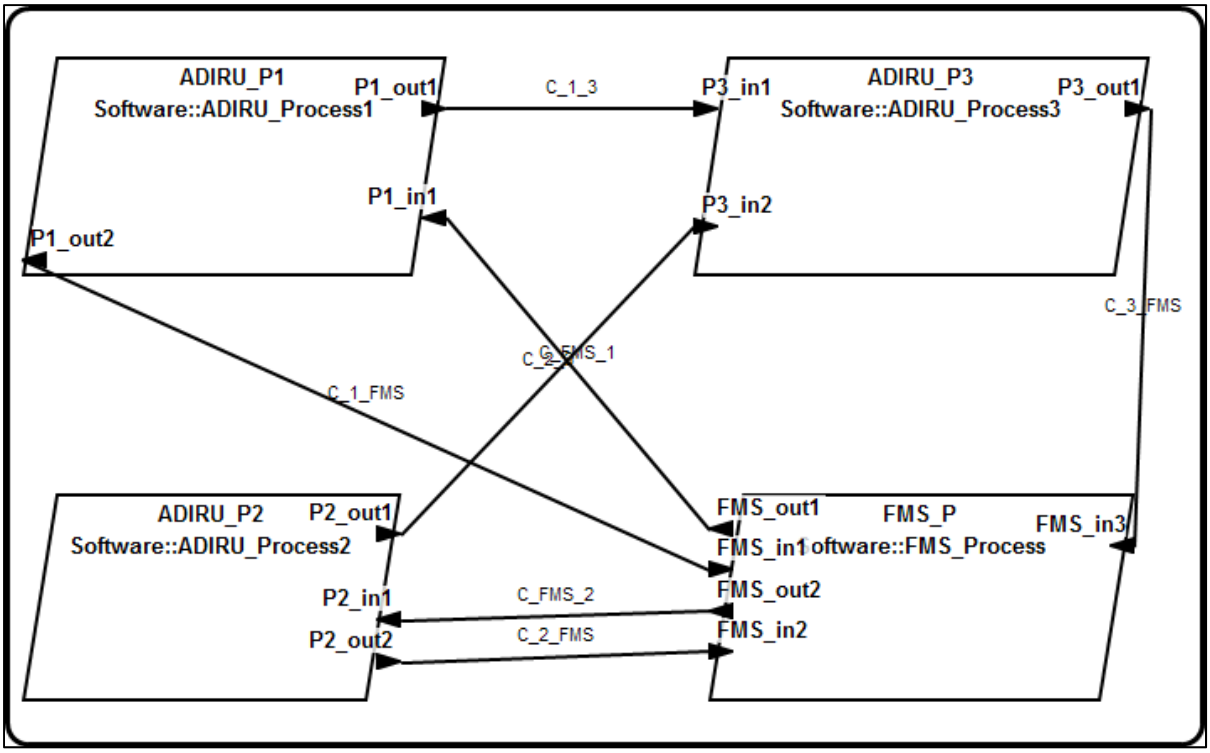


Figure 2.12 Diagramme de la couche logicielle (haut niveau d'abstraction)

Les connexions entre les ports sont ainsi définies pour déterminer les interconnexions logiques entre les processus. Dans le modèle AADL, ces informations sont indiquées dans la clause **`connections`** en indiquant les deux extrémités (source et destination) de la connexion comme suit:

```
CONNECTIONS

C_1_FMS: PORT ADIRU_P1.P1.out2->FMS_P.FMS_in1;
```

2.4.1.3 Les processeurs virtuels

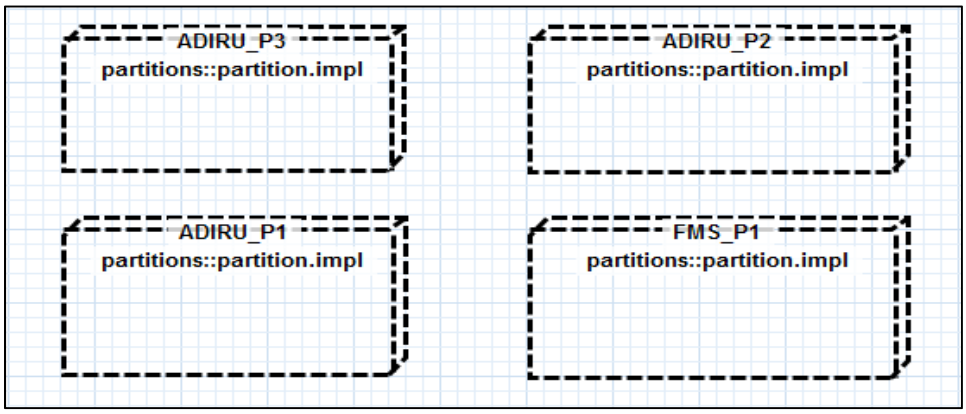


Figure 2.13 Modélisation des composants processeurs virtuels

Comme recommandé par l'annexe ARINC 653 d'AADL, les partitions sont modélisées par des processeurs virtuels. Nous avons supposé dans notre étude de cas, trois partitions pour l'ADIRU et une partition pour le FMS. Chaque processeur virtuel doit être attaché à un et un seul processeur physique (`processor`) qui présente la plateforme d'exécution. Par contre, un processeur peut être relié à plusieurs partitions puisque le concept IMA favorise l'intégration des partitions correspondantes à des applications logicielles différentes, dans une plateforme matérielle commune.

2.4.1.4 Le réseau AFDX

Le réseau AFDX est présenté par le composant `virtual bus` composé de trois VL représentées par le composant AADL `virtual bus`.

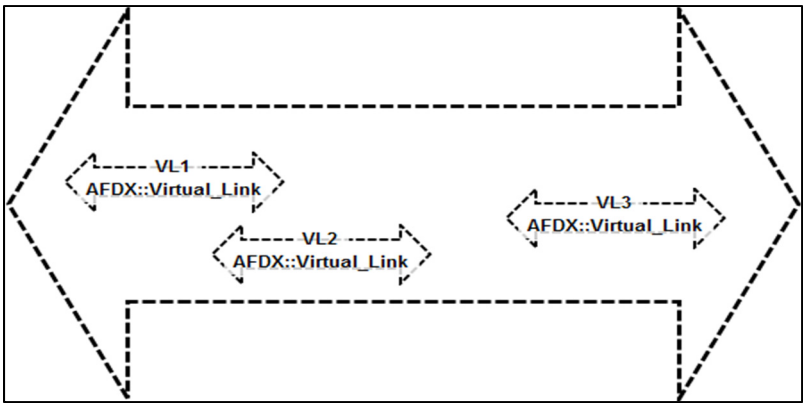


Figure 2.14 Réseau AFDX

Des informations supplémentaires sont associées aux VL par l'intermédiaire des propriétés spécifiques (ajoutées par Khoroshilov). L'exemple suivant illustre les propriétés assignées aux VL:

```
VIRTUAL BUS IMPLEMENTATION AFDX_Net.i
    SUBCOMPONENTS
      VL1 : VIRTUAL BUS AFDX::Virtual_Link;
      VL2 : VIRTUAL BUS AFDX::Virtual_Link;
      VL3 : VIRTUAL BUS AFDX::Virtual_Link;
       PROPERTIES
      -- virtual links configuration
      AFDX_Properties::BAG => 1 ms applies to VL1,VL2,VL3;
      AFDX_Properties::Lmax => 1518 Bytes applies to VL1;
      AFDX_Properties::Lmax => 512 Bytes applies to VL2,VL3;
      AFDX_Properties::SkewMax => 1 ms applies to VL1,VL2,VL3;
```

où:

- la propriété **`AFDX_Properties::BAG`** indique le temps minimum entre deux émissions;
- la propriété **`AFDX_Properties::Lmax`** spécifie la taille maximum d'une trame;
- la propriété **`AFDX_Properties::SkewMax`** précise le temps maximum entre deux trames redondantes.

Les caractéristiques des tables de routage sont spécifiées avec la propriété **`AFDX_Properties::VL_Route_Table`**.

AFDX_Properties::VL_Route_Table:

VL1 :

- Jitter => TIME (maximum allowed jitter)
- Priority => **(high or low)**
- Accounting_policy => **(Frame or byte)**

VL2: …

VLn: …

La figure 2.15 présente un schéma simplifié du réseau AFDX et de ses composants.

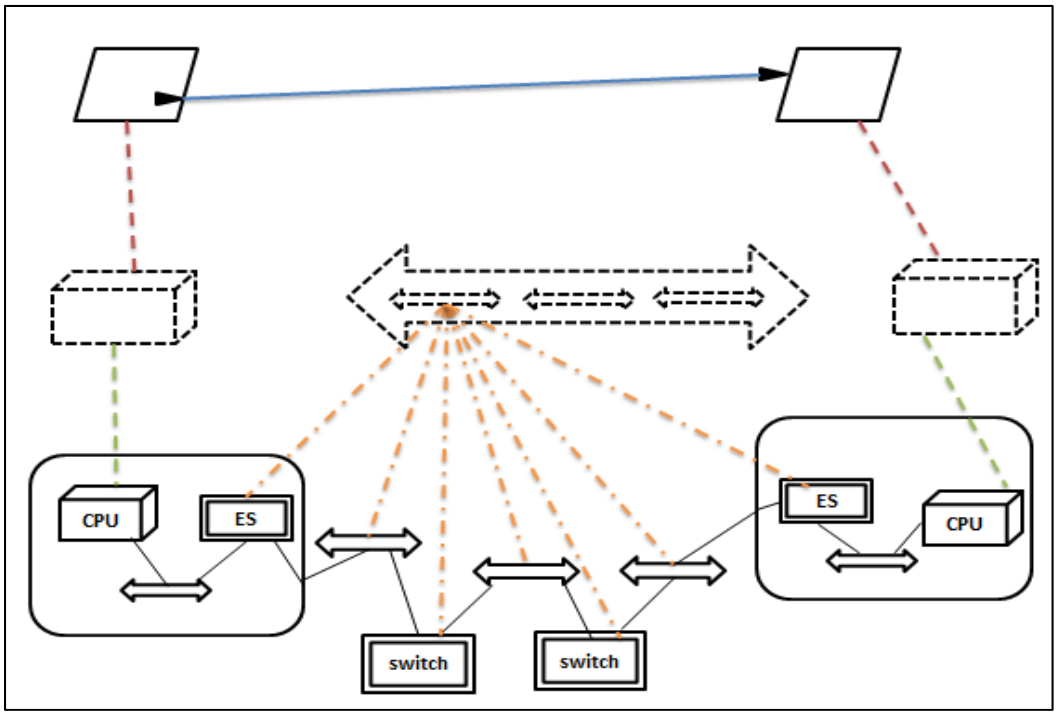
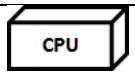
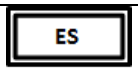
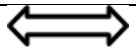
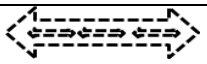
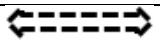


Figure 2.15 Système global

Tableau 2.4 Légende du schéma

	Processus
	Processeur virtuel
	Système représentant le module CPM
	Processeur d'exécution CPU
	Carte AFDX *End system*
	Bus physique
	Réseau AFDX
	VL
	Port data
	Communication inter-partition
	Lien physique
	Association (*binding*) du processus à la partition
	Association (*binding*) de la partition au processeur d'exécution
	Association représentant la connexion virtuelle liant les *End Systems*, les *wires* et *les switches* aux *VL* (assurée par la propriété AADL `Actual_Connection_Binding`)

2.4.2 Validation du modèle avec RESOLUTE

2.4.2.1 Vérification de la conformité du modèle à l'annexe AADL du standard ARINC 653

Nous avons intégré les règles de validation RESOLUTE de Delange et Hugues dans notre modèle afin de vérifier si le modèle satisfait toutes les exigences de l'annexe ARINC 653. Le résultat de la validation, présenté sur la figure 2.16, montre que toutes les règles sont vérifiées sauf la dernière contrainte.

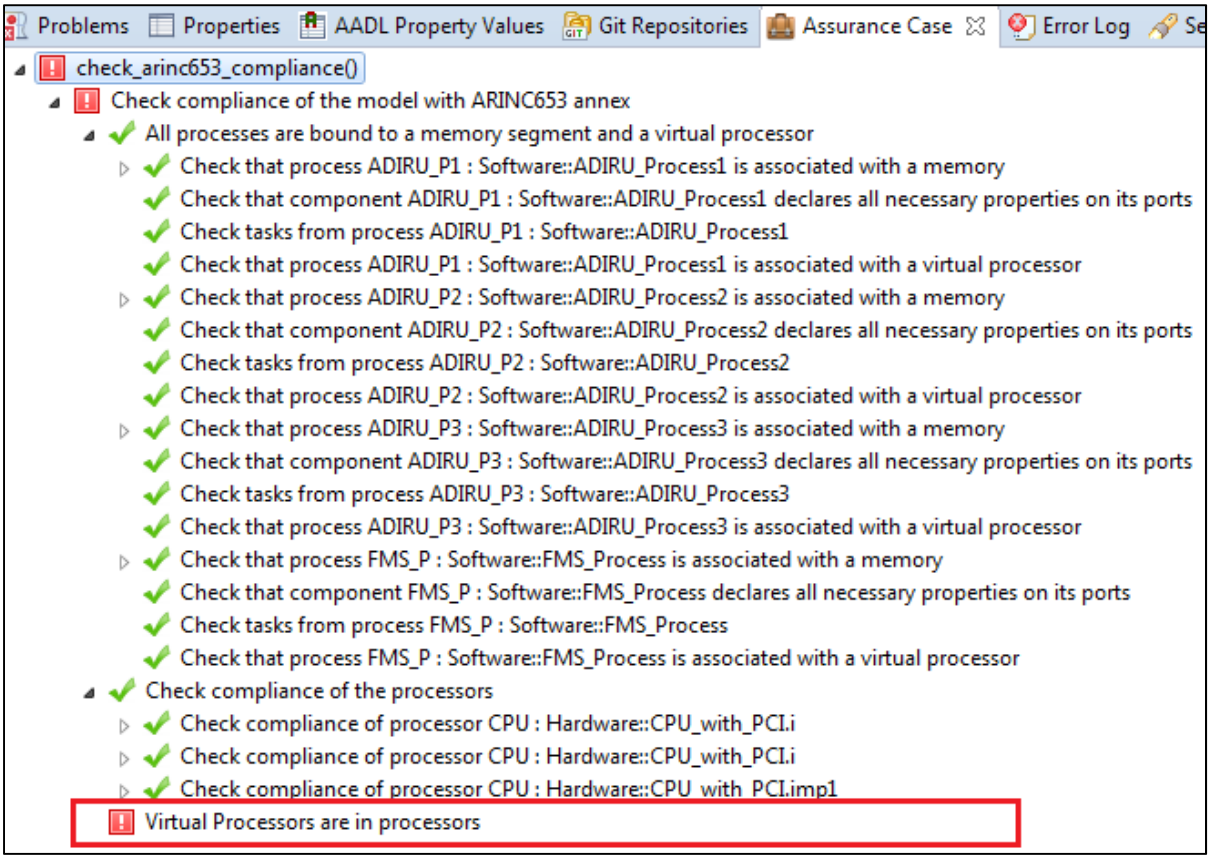


Figure 2.16 Validation RESOLUTE des règles de conformité ARINC 653

La dernière contrainte n'est pas respectée (en rouge sur la figure 2.16) parce que dans notre modèle nous avons associé les partitions (processeurs virtuels) aux processeurs physiques en les reliant par la propriété `Actual_Processor_Binding` et non pas en les incluant

directement dans les processeurs, en nous conformant à la façon de faire préconisée dans les travaux de Khoroshilov.

Comme déjà mentionné, la proposition de Khoroshilov vise à fournir une nouvelle annexe ARINC 664 pour la communauté AADL. Pour modéliser un système IMA en AADL, nous considérons souhaitable de profiter de l'annexe ARINC 653 et de l'annexe ARINC 664 pour présenter tous les aspects et les composants d'un sous-système IMA dans un même modèle. En comparant les deux modèles proposés par Hugues et Delange d'une part et Khoroshilov d'autre part, la démarche de conception n'est pas la même, surtout au niveau de la définition des partitions virtuelles. En suivant le premier modèle, les partitions virtuelles doivent être définies dans des processeurs physiques, où l'intégrateur qui réalise le modèle peut spécifier les propriétés reliées au partitionnement temporel. Dans le deuxième modèle, on ne peut pas ajouter ces propriétés car les processeurs virtuels ne sont pas des sous-composants du processeur physique. Le modèle proposé par Khoroshilov, même s'il décrit plus précisément le réseau AFDX, ne prend pas en compte le principe de partitionnement temporel de l'annexe ARINC 653.

Aussi, nous avons adopté le modèle de Khoroshilov puisqu'il sépare les partitions des plateformes d'exécution pour avoir plus de flexibilité dans l'intégration des partitions logicielles dans des plateformes matérielles. Par conséquent, la configuration du déploiement des partitions dans des ressources matérielles devient plus flexible en modifiant uniquement l'allocation d'une partition d'un processeur physique vers un autre. En outre, l'échec de la satisfaction de la dernière contrainte n'affecte pas la validité de notre modèle puisque AADL permet ce type de *binding* et nous nous concentrons sur la communication inter-partition dans ce mémoire, qui est assurée par le réseau AFDX.

Cette version de notre modèle a permis de mettre en lumière une incompatibilité entre une annexe existante (ARINC 653) et une annexe (ARINC 664) en devenir dans l'écosystème AADL.

2.4.2.2 Vérification de la conformité du modèle au standard ARINC 664

Pour valider la conformité de notre modèle au standard ARINC 664, nous avons utilisé la validation RESOLUTE ARINC 664 que nous avons proposée dans la section 2.3.2. Toutes les règles définies sont validées, car nous avons inclus dans le modèle toutes les propriétés requises aux bons endroits.

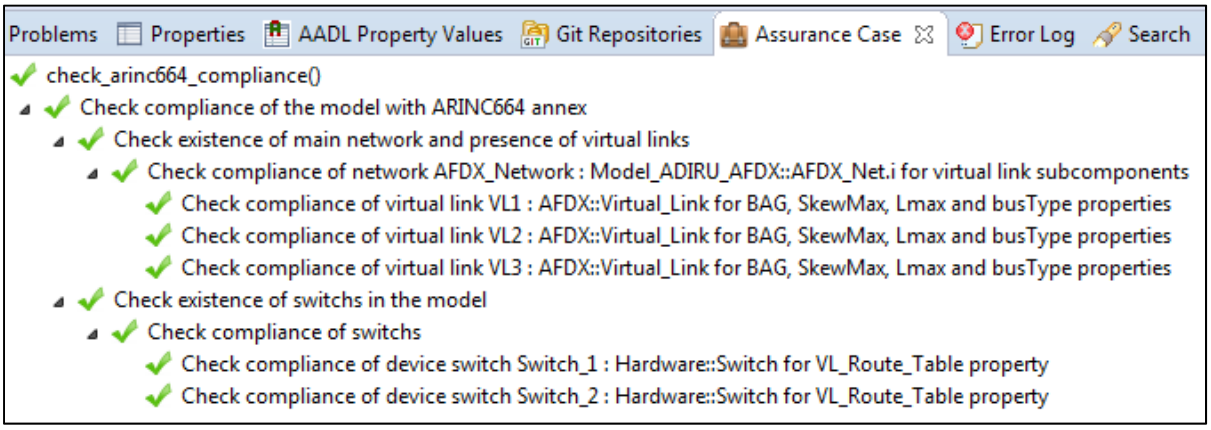


Figure 2.17 Validation RESOLUTE des règles de conformité ARINC 664

2.5 Évaluation des capacités de modélisation d'AADL

AADL est un langage de modélisation convivial pour la modélisation des systèmes avioniques. En effet, un modèle AADL peut être exprimé par différentes représentations:

- une représentation textuelle qui fournit une description détaillée et expressive;
- une représentation graphique, pour avoir une description plus claire et une vision globale du système;
- une représentation en XML (le modèle d'instance) pour faciliter l'interopérabilité entre outils (Feiler, P., 2004). Dans une représentation XML les données sont faciles à repérer et à extraire. Les informations pertinentes aux descriptions des ICD peuvent être analysées par d'autres outils ou bien affichées dans des pages Web (HTML).

La modélisation des concepts du standard ARINC 653 a été faite à l'aide de l'annexe ARINC 653 officielle d'AADL. Nous avons pu décrire les principes de partitionnement, de la

séparation spatiale et de la séparation temporelle. Nous avons défini les communications inter-partition et assigné les caractéristiques nécessaires aux ports (échantillonnage ou file d'attente). En profitant d'une éventuelle annexe AFDX, nous avons structuré l'architecture et les composants du réseau AFDX. Des propriétés pertinentes sont assignées aux éléments du réseau (les *VL*, les *Switches*).

L'écosystème d'AADL offre des outils de validation du modèle à des exigences et des contraintes prédéfinies. L'expérimentation du langage RESOLUTE (supporté par l'outil OSATE2) pour la validation des exigences architecturales de notre modèle illustre l'utilisation de ce langage pour décrire et vérifier les contraintes désirées.

Du point de vue de la description des interfaces de systèmes avioniques, le principal défi avec AADL demeure le support pour décrire la structure ou le format des données échangées à travers les ports. En AADL, les données sont représentées par un simple composant `data` sans aucune information sur sa structure. Même en ajoutant des propriétés spécialisées à des composants `data`, ceux-ci demeurent trop peu expressifs pour capturer les informations qui nous intéressent dans les ICD de systèmes avioniques. Il faut donc complémenter les modèles AADL avec un autre langage pour capturer les informations pertinentes aux communications.

2.6 Conclusion

Nous avons décrit les étapes suivies pour la modélisation du système ADIRU et ses interactions avec d'autres composants (FMS, RDC) à travers le réseau AFDX. La première étape est de chercher les informations pertinentes sur les sous-systèmes à intégrer et de se concentrer sur les informations relatives aux interfaces. Une fois que ces informations sont collectées, l'étape de modélisation avec AADL peut être entamée.

Nous avons présenté les composants de notre modèle en mentionnant les annexes AADL qui ont été utilisées dans notre expérimentation. Nous avons pu vérifier la conformité de ce modèle aux exigences des plateformes IMA par le langage de validation RESOLUTE pour vérifier sa performance comme outil de validation.

Cette expérimentation a été évaluée en soulignant les avantages et les inconvénients d'AADL dans le contexte de la description des ICD. Les annexes ARINC 653 et ARINC 664 sont utiles pour représenter et décrire l'architecture et les différents composants d'un système IMA. Ensuite, l'outil RESOLUTE permet à l'intégrateur de valider la conformité de son modèle aux standards ARINC 653 et ARINC 664 ou pour faire d'autres validations pour vérifier l'interopérabilité de ces sous-systèmes en définissant les règles RESOLUTE appropriées. La modélisation AADL donne une idée claire sur la structure du sous-système et de ses ports d'entrées/sorties grâce à une représentation graphique et textuelle qui décrit précisément ce modèle.

Dans ce chapitre nous avons proposé un modèle AADL d'un système avionique en combinant les aspects IMA et AFDX dans un même modèle. Ce travail a relevé une incohérence entre l'annexe officielle ARINC 653 et l'ébauche d'annexe ARINC 664. Nous considérons l'identification de cette incohérence comme une contribution importante de notre travail.

La modélisation des structures des données échangées à travers les ports en AADL pose un défi important, alors que la description détaillée de ces données est indispensable dans les ICD. Les informations échangées entre les partitions d'un système IMA sont les informations les plus pertinentes sur les interfaces pour notre partenaire industriel. Le chapitre suivant présente notre solution pour remédier aux défis rencontrés avec la modélisation des données.

CHAPITRE 3

Description des données échangées

Dans le but d'extraire les ICD des sous-systèmes avioniques IMA, notre intérêt s'est porté dès le début sur l'usage du langage AADL pour modéliser les systèmes afin d'en extraire les ICD. Dans le chapitre précédant, nous avons décrit notre tentative de modélisation à travers une étude de cas. Nous avons pu décrire toutes les informations concernant l'architecture, la structure, les composants, les ports d'entrées/sorties pour un sous-système.
Cependant, dans le cas d'un ICD pour systèmes avioniques, la définition de la structure des données échangées est très importante. Le langage AADL ne permet pas de définir finement le format de ces données. Les informations échangées entre les partitions d'un système IMA sont sous forme de trames AFDX standardisées.
Dans le présent chapitre, nous décrivons la structure de la trame AFDX. Nous présentons les outils utilisés pour la description de format de données ainsi que l'expérimentation de la modélisation de ces données.

3.1 Structure d'une trame AFDX

Le réseau AFDX est un réseau Ethernet commuté, constitué d'un ensemble de commutateurs qui acheminent les trames vers leurs destinations. Les *End Systems* (ES) sont les nœuds (ou les terminaux) du réseau AFDX qui jouent le rôle des transmetteurs ou récepteurs des données. Les ES se comportent comme une interface entre les applications logicielles et le réseau commuté comme présenté sur la figure 3.1.

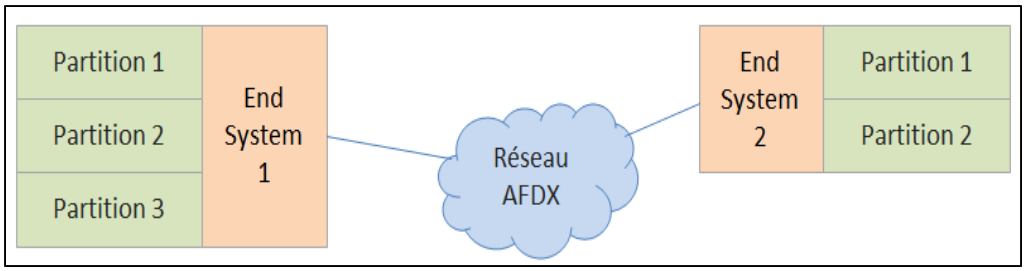


Figure 3.1 *End Systems* du réseau AFDX

Chaque ES est relié à un commutateur par un câblage constitué de deux paires de fil torsadées : une paire pour la transmission (Tx) et une paire pour la réception (Rx) (figure 3.2), pour éviter les collisions sur le câble. Les commutateurs se chargent de l'acheminement des trames à l'aide d'une table de routage préconfigurée. Ces commutateurs sont configurés pour acheminer les trames entrantes vers une ou plusieurs destinations. Une trame AFDX doit avoir une seule adresse source qui représente un *End System*.

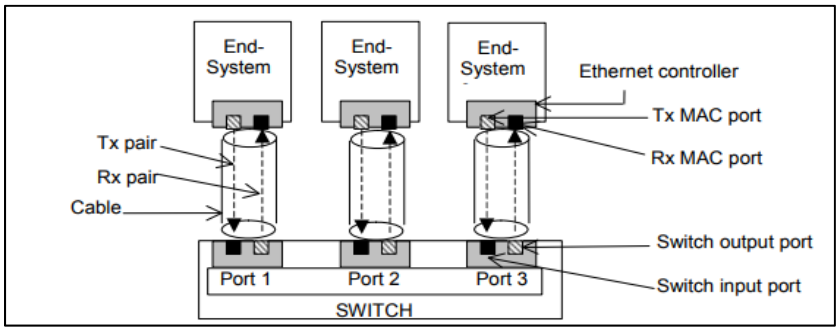


Figure 3.2 Exemple de topologie physique. Figure extraite de (AEEC., 2009a)

La figure 3.3 schématise un exemple de communications AFDX. Les deux partitions (partition 1 et partition 2) de l'ES 1 envoient des trames aux partitions des ES 2 et ES 3 via des VL. Chacune de ces deux partitions possède une adresse IP unique (par exemple, les adresses IP sources 10 et 20 respectivement pour les partitions 1 et 2).

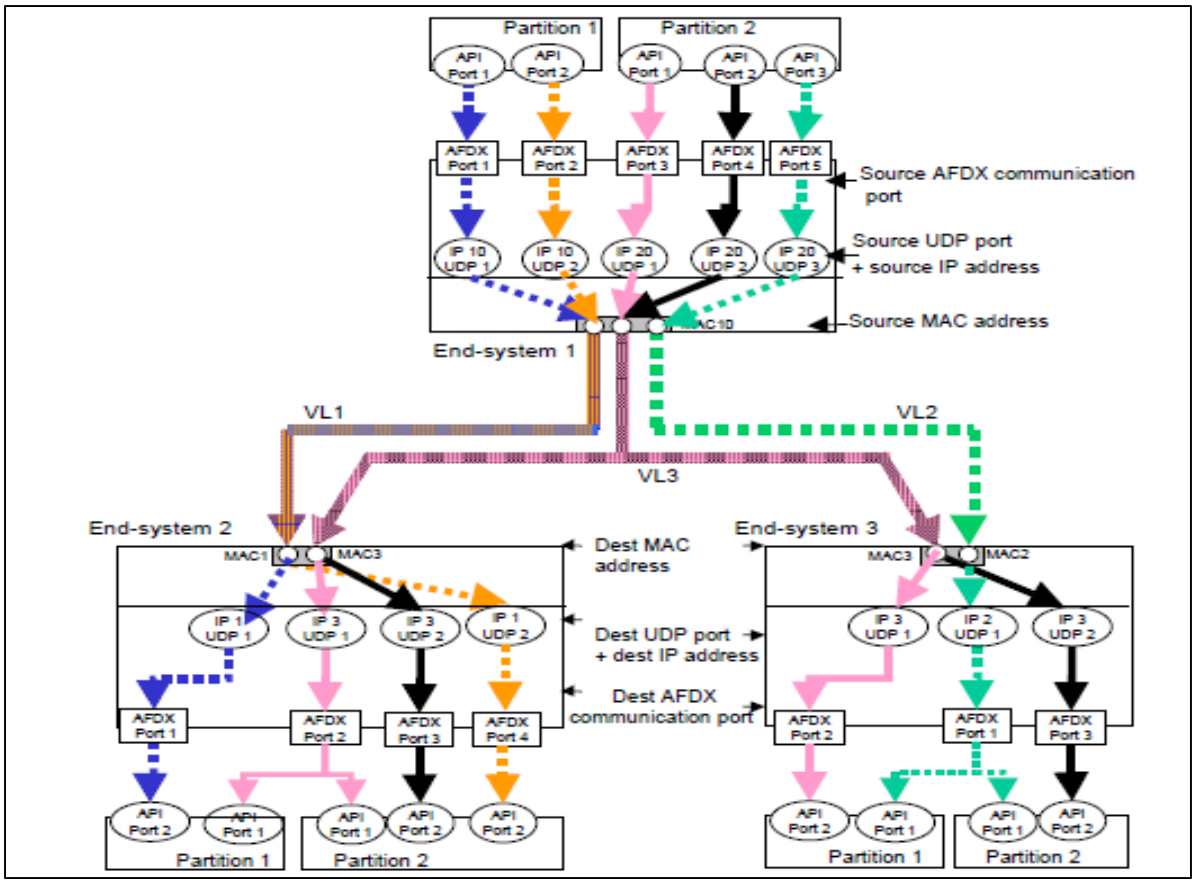


Figure 3.3 Exemple de communications (chaque couleur représente une communication) dans un réseau AFDX. Figure extraite de (AEEC, 2009a)

Une trame transmise est dirigée vers le ou les ES de destination en utilisant l'identifiant du VL encodé dans son adresse MAC destination. Une trame AFDX est donc identifiée par un quintuplé : port UDP source, adresse IP source, adresse MAC destination, adresse IP destination et port UDP destination.

Les messages échangés sur le réseau ont un format spécifique et standardisé par ARINC 664-partie 7. En général, une trame AFDX est une trame Ethernet mais qui est bien spécifique aux réseaux AFDX. La figure 3.4 décrit la structure générale d'une trame AFDX.

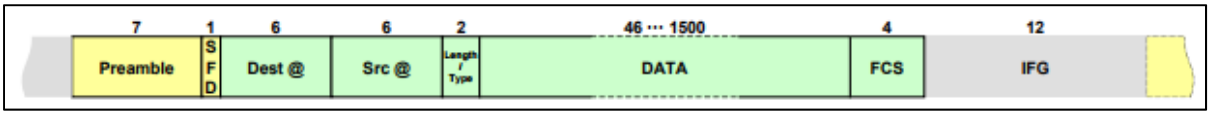


Figure 3.4 Structure générale d'une trame AFDX. Figure extraite de (AEEC, 2009a)

Sur la figure 3.4, la trame AFDX (représentée en vert) commence par l'adresse MAC de destination, suivie de l'adresse MAC source, qui font toutes deux partie de l'entête Ethernet. Avant la transmission de la trame, le ES envoie une séquence d'octets appelée préambule pour signaler la transmission d'un nouveau message. Celui-ci permet aux ES récepteurs de se synchroniser et de se préparer pour la réception. Après le préambule, le ES envoi un *Start Frame Delimiter* (SFD) pour signaler que la trame débute immédiatement après le SFD.

Une trame AFDX est composée principalement d'un entête Ethernet (14 octets), des données (minimum 46 et maximum 1500 octets) et d'un champ FCS (*Frame Check Sequence*). Ce dernier sert à vérifier si la trame est arrivée en bon état ou non. L'entête Ethernet contient l'adresse MAC destination, l'adresse MAC source et le type ayant la valeur 0x0800 qui signifie IPv4 (*Internet Protocol version 4*). Chacune des adresses MAC possède un format spécifique.

L'adresse destination est composée de deux champs: un champ constant et un champ qui contient l'identifiant du VL, puisqu'il s'agit d'un envoi multicast vers un ou plusieurs ES.

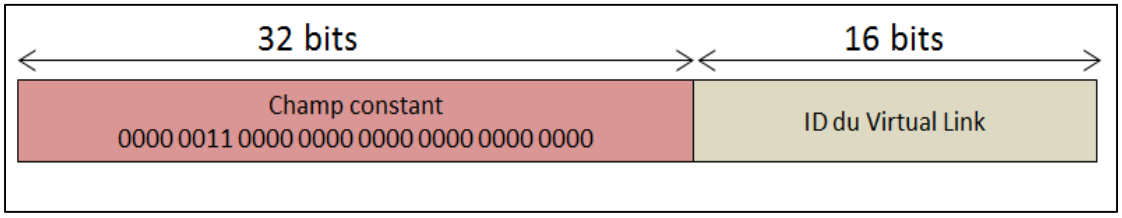


Figure 3.5 Adresse destination

L'adresse source est une adresse unicast. Un ES est identifié par le *network ID* et *l'equipment ID*.

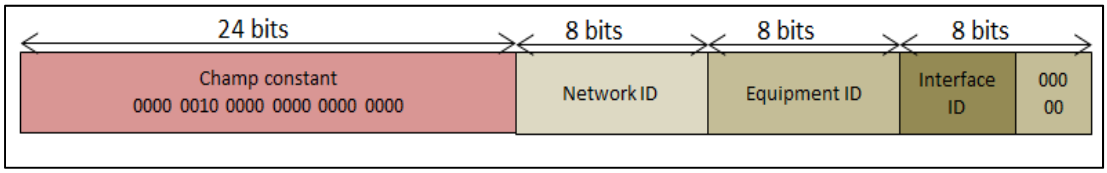


Figure 3.6 Adresse source

La figure 3.7 fournit plus de détails sur la structure des données Ethernet (*Ethernet payload*).

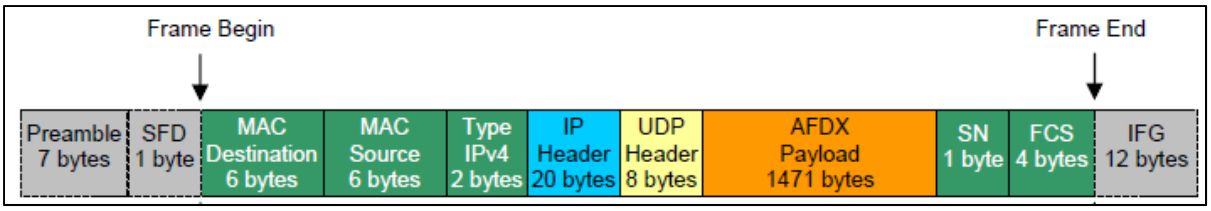


Figure 3.7 Structure détaillée d'une trame AFDX. Figure extraite de (An, D. *et al*., 2015)

L'*Ethernet Payload* contient l'entête IP, l'entête UDP puis les données AFDX suivies du numéro de séquence (SN). Ce numéro est ajouté à chaque trame; il est incrémenté pour chaque trame successive envoyée par un ES. Lors de la réception, les trames reçues sont réorganisées par leur numéro et les trames dupliquées sont supprimées pour conserver une seule trame pour chaque numéro de séquence.

L'entête IP a une taille de 20 octets et contient les champs présentés sur la figure 3.8.

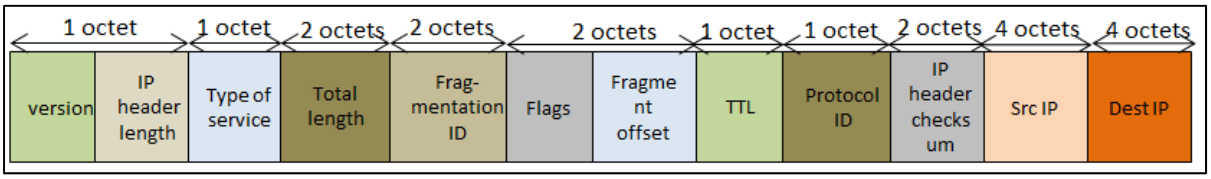


Figure 3.8 Entête IP

Les figures précédentes montrent qu'une trame AFDX possède une structure bien déterminée et bien spécifique. Un langage de description des formats de données s'avère nécessaire pour décrire la structure des messages transférés entre les ES. La section suivante présente le langage et l'outil utilisés pour la description du format des données.

3.2 DFDL

Pour la description des structures des données échangées, nous avons commencé par expérimenter la notation ASN.1 (*Abstract Syntax Notation One*) qui a été proposée par notre partenaire industriel. ASN.1 ou notation de syntaxe abstraite est conçue pour spécifier, à haut niveau d'abstraction, les messages échangés entre des programmes, indépendamment des plateformes et des langages de programmation. En pratique, ASN.1 est utilisé pour décrire les informations générées par des protocoles haut niveau. Dans notre expérimentation, nous

avons rencontré des problèmes avec l'interprétation des données. En effet, il est difficile de décrire une syntaxe abstraite pour des trames AFDX ou données ARINC 429 qui peuvent contenir différents types d'encodage (BCD, BNR). La notation ASN.1 est utilisée pour décrire des protocoles qui suivent des règles d'encodage préétablies. Les détails d'encodage sont cachés dans les spécifications abstraites de ASN.1 et dans les supports d'exécution fournis par ses outils (Larmouth, J., 2000).

Ensuite, le langage DFDL a été proposé par le partenaire industriel comme solution alternative. DFDL[5] (*Data Format Description Language*) est un langage publié par le *Open Grid Forum*[6] comme langage de modélisation pour la description des formats des données binaires ou textuelles. Le choix de ce langage est également motivé par la l'existence d'un exemple de description de la structure d'une trame Ethernet en DFDL. Cet exemple est très proche de ce que nous souhaitons modéliser puisqu'une trame AFDX est une trame Ethernet mais qui est adoptée pour le besoin du réseau AFDX. De plus, par rapport à ASN.1, DFDL a l'avantage de permettre de spécifier comment les informations sont encodées et décodées.

D'une manière générale, DFDL est conçu pour résoudre le problème d'échange de données entre applications différentes. Ce langage fournit une manière standard de décrire différents types de formats de données. Il génère une description logique du contenu des données indépendamment de leur format physique.

La description du format des données se fait par le modèle ou le schéma DFDL qui permet le '*parsing*' ou la lecture des données binaires à partir de leur format natif. La figure 3.9 montre un exemple de processeur DFDL qui fait à la fois le processus du '*parsing*[7]' et le '*unparsing*' chemin inverse (en pointillé).

---

[5] https://www.ogf.org/documents/GFD.174.pdf
[6] https://www.ogf.org
[7] Le '*parsing*' : prendre un message physique et le convertir en données structurées XML. Le '*unparsing*' est l'inverse.

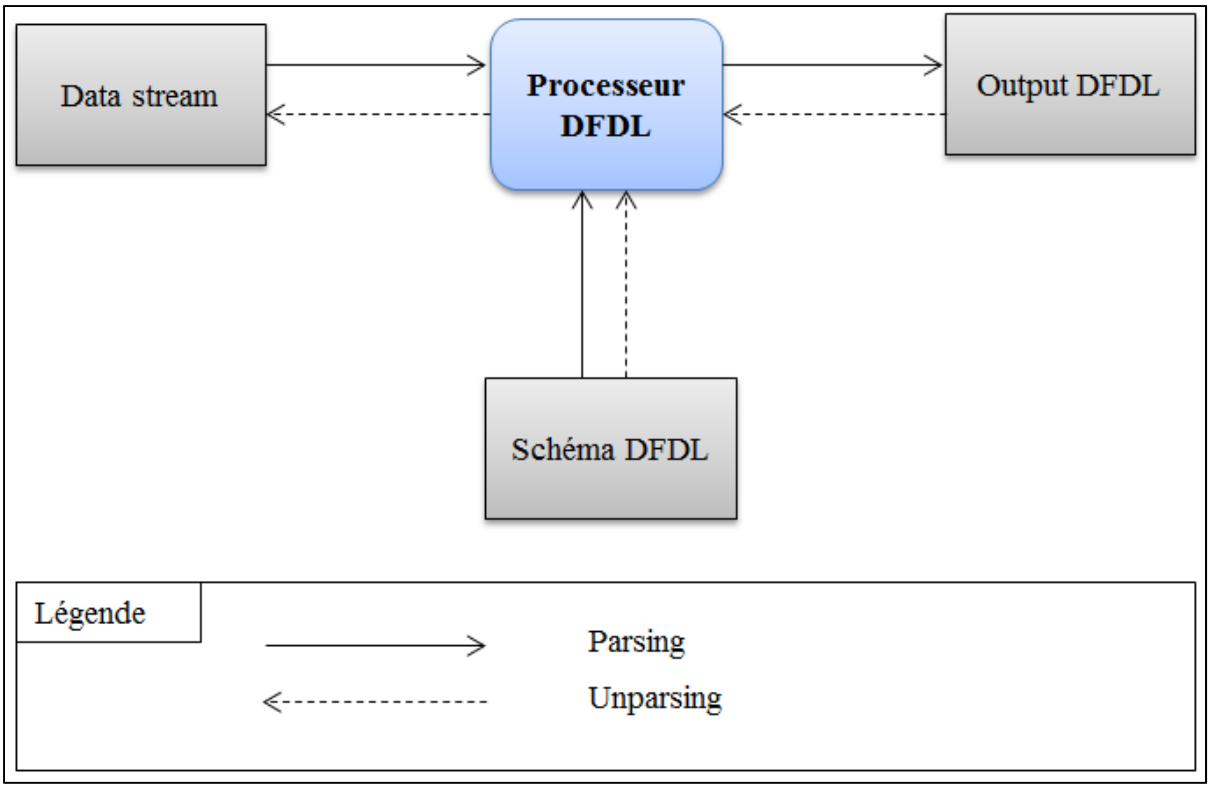


Figure 3.9 Processeur DFDL

Le langage DFDL utilise :

- un sous ensemble du *W3C XML schema* pour décrire la structure logique des données;
- des annotations *XSD* pour décrire la représentation physique des données.

Dans notre mémoire, nous avons utilisé un processeur DFDL disponible en logiciel libre nommé ***Daffodil*** [8] **,** disponible depuis 2015. Daffodil est en développement actif pour supporter plusieurs types de données. Le site officiel de Daffodil contient des exemples d'implémentation de schémas DFDL pour décrire des données de types différents, comme l'exemple de la description de la structure d'une trame Ethernet. Cet exemple s'est avéré très utile pour notre cas, puisqu'une trame AFDX est une trame Ethernet de base.

[8] https://opensource.ncsa.illinois.edu/confluence/display/DFDL/Daffodil%3A+Open+Source+DFDL

3.3 Description DFDL de la structure d'une trame AFDX

En suivant l'exemple de Daffodil, nous avons adopté une décomposition en couches de la trame Ethernet qui reflète le principe d'encapsulation des couches du modèle OSI comme présenté sur la figure 3.10.

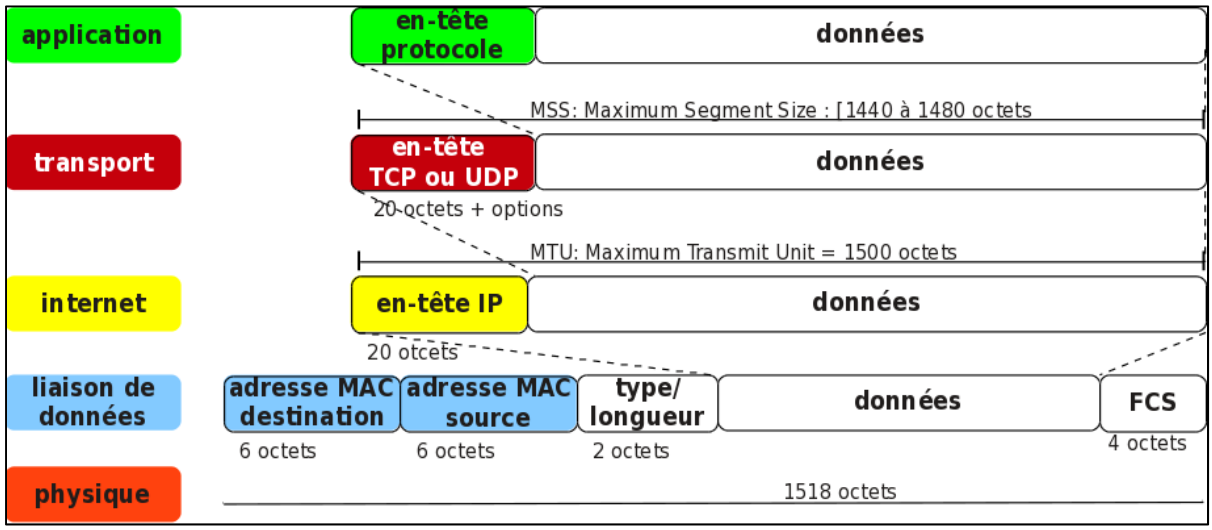


Figure 3.10 Décomposition en couches de la trame Ethernet selon le principe d'encapsulation. Source: http://inetdoc.developpez.com/tutoriels/modelisation-reseau

De la même manière, nous avons décrit la structure en couches comme pour une trame AFDX, tel qu'indiqué sur la figure 3.11.

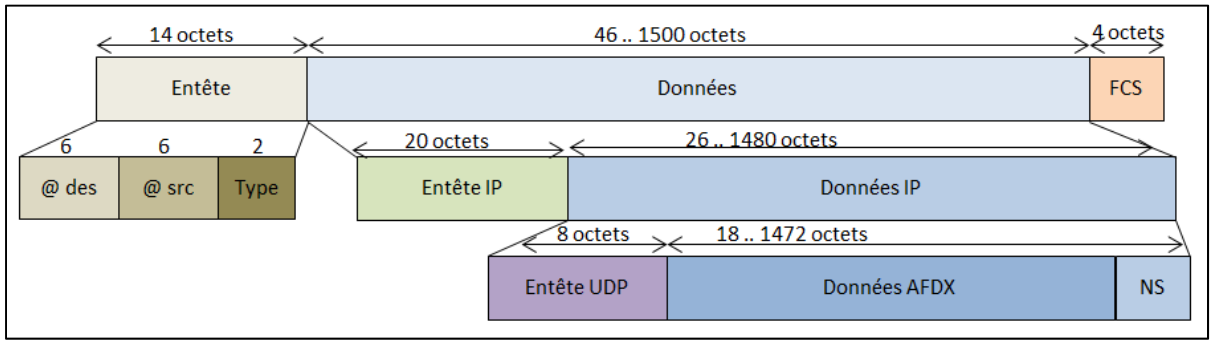


Figure 3.11 Principe d'encapsulation d'une trame AFDX

Le protocole UDP ou *User Datagram Protocol* est le protocole utilisé dans la couche transport pour assurer une transmission simple et plus efficace en mode non connecté et sans

contrôle de transmission, puisque la politique de la bande passante (*bandwidth*) AFDX et la gestion de redondance assurent une probabilité minimale de perte de trame.

Pour décrire la première couche en DFDL, la figure 3.12 présente un extrait du schéma DFDL qui définit la séquence des éléments qui compose la trame AFDX.

```
<!-- AFDX Frame -->
<xs:element name="AFDXFrame">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="afdx:MacHeader" />
      <xs:element ref="afdx:EthernetPayload" />
      <xs:element name="FCS" type="afdx:hexByte" dfdl:length="4"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 3.12 Description de la couche Ethernet en DFDL

L'élément **`AFDXFrame`** est le type complexe, composé d'une séquence d'éléments. Les sous-éléments de la trame AFDX sont l'entête Ethernet référé par l'élément **`afdx:MacHeader`**, le *Payload* Ethernet référé par **`afdx:EthernetPayload`** et l'élément FCS qui est de type binaire et de taille de quatre octets. Chacun des éléments mentionnés dans l'attribut **`ref`** est décrit de même comme étant un élément de type complexe pour détailler sa structure par une séquence de sous-éléments. La figure 3.13 montre par exemple la description de l'*Ethernet Payload* qui contient les éléments de la couche réseau (l'entête IP et l'*IP Payload*).

```
<!-- Ethernet Payload -->
<xs:element name="EthernetPayload">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="afdx:IPv4Header" />
      <xs:element ref="afdx:IPv4Payload" />
      <xs:element name="SeqNumber" type="afdx:hexByte" dfdl:length="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 3.13 Description DFDL de la structure de l'*Ethernet Payload*

Grace aux annotations DFDL, nous pouvons vérifier la validité de certains champs comme les champs constants (en rouge sur les figures 3.5 et 3.6). L'exemple suivant montre le cas de validation d'un champ constant de l'adresse de destination MAC. Dans ce cas de test, le processeur DFDL doit s'assurer que la condition citée dans l'attribut **`test`** de la balise **`dfdl:assert`** est bien valide. L'attribut message de la même balise contient le message d'erreur qui va être affiché si le test est négatif. La balise de l'assertion est contenue dans la balise **`xs:appinfo`** qui contenue dans la balise **`xs:annotation`**. L'annotation doit apparaitre au début de l'élément concerné par le test, qui est dans notre exemple l'élément **`MacDestAddrCstField`**.

```
<xs:element name='MacDestAddrCstField' type="xs:hexBinary" dfdl:length="4" dfdl:lengthUnits="bytes"
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:assert message="First thirty two bits of MAC destination address must be a constant"
                   test="{ (xs:string(.) eq '03000000') }" />
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Figure 3.14 Exemple d'annotation DFDL

D'une manière générale, le processeur DFDL fait la conversion (*parsing*) des données de la trame du format physique (bits, octets) vers le format logique (structure de données en XML) en utilisant le schéma DFDL, qui définit comment effecteur cette conversion. Si une erreur survient, un message d'échec adéquat est affiché.

De la même manière, nous avons décrit la structure de la trame AFDX hiérarchiquement en décrivant ses éléments couche par couche. La dernière couche contient les données AFDX. Les concepteurs des sous-systèmes avioniques sont libres dans le choix de la structure des données AFDX selon ce qui convient avec leur application (Yanik M., 2007). Le standard ARINC 664 – partie 7 identifie deux types de messages AFDX: explicites et implicites. Le type explicite inclut les informations qui permettent aux récepteurs d'interpréter correctement les données, tandis que le type implicite ne contient pas ces informations d'aide à l'interprétation. Dans ce qui suit, nous présentons un exemple de données AFDX contenant tout simplement un mot ARINC 429 dont nous détaillons la structure de son format standard.

Nous avons vu un exemple spécifique de mot ARINC 429 lors de la discussion sur l'ILS dans la section 2.1.1.

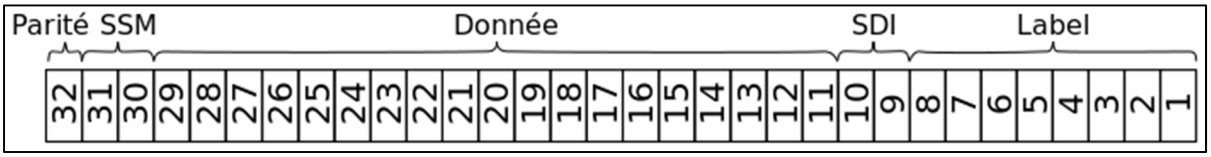


Figure 3.15 Structure générale d'un mot ARINC 429. Source: https://fr.wikipedia.org/wiki/ARINC_429

Le mot ARINC 429 est composé, comme présenté sur la figure 3.15, de cinq champs principaux:

- champ pour le bit de parité;
- champ pour le Sign/Status Matrix (SSM);
- champ pour la donnée;
- champ pour le Source Destination Identifier (SDI);
- champ pour le label du mot.

Le champ donnée est constitué de 19 bits, du bit 11 au bit 29, réservés pour contenir la valeur de la donnée, mais ces bits ne sont pas tous utilisés pour le codage de la donnée dans tous les messages. En effet, cela dépend du type de la donnée à coder pour distinguer entre les bits de rembourrage (*padding*) et les bits de donnée. La structure DFDL de ce mot est présentée sur la figure 3.16.

```
<!-- ARINC429 data -->
<xs:element name="ARINC429Data">
  <xs:complexType>
    <xs:sequence>
      <xs:sequence dfdl:hiddenGroupRef="afdx:hiddenLabel"/>
      <xs:element name="Label" type="afdx:bit" dfdl:length="8"
                  dfdl:inputValueCalc="{ (../bit1 +  ../bit2 * 2 + ../bit3 * 4)+
                  ((../bit4 +  ../bit5 * 2 + ../bit6 * 4 ) * 10 )+ ((../bit7 +  ../bit8 * 2 )* 100)
                  }"/>
      <xs:element name="sdi" type="afdx:bit" dfdl:length="2" />
      <xs:element name="padding" type="afdx:bit" dfdl:length="2"/>
      <xs:sequence dfdl:hiddenGroupRef="afdx:hiddenRawData"/>
      <xs:element name="Data_Value" type="afdx:Double" dfdl:length="16"
                  dfdl:inputValueCalc="{ ../rawData * $afdx:data-range
                  }"/>
      <xs:element name="sign" type="afdx:bit" dfdl:length="1"/>
      <xs:element name="statusMatrix" type="afdx:bit" dfdl:length="2"/>
      <xs:element name="parity" type="afdx:bit" dfdl:length="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 3.16 Schéma DFDL de la structure d'un mot ARINC 429

Le décodage du champ data est déterminé en utilisant deux paramètres: la plage de valeurs (*range*) et le nombre de bits utilisés pour coder la valeur de la donnée. Ces paramètres sont définis dans le schéma DFDL par les variables **`data-range`** qui présente la plage de valeurs et le **`data-nb-bit`** qui présente le nombre de bits utilisé. Avec le langage DFDL, on peut ajouter des fonctions pour le calcul des champs complexes comme dans le cas du champ **`data`** et du champ **`label`**. Sur la figure 3.16, l'attribut **`dfdl:inputValueCalc`** permet de définir une fonction, utilisée dans le *parsing*, en utilisant des opérateurs arithmétiques comme (**`div`**, **`mod`**, *). Cependant, seulement les opérateurs XSD sont supportés en DFDL à l'heure actuelle et non pas tous les opérateurs (par exemple l'opérateur de calcul de puissance n'est pas supporté). La séquence définit avec l'attribut **`dfdl:hiddenGroupRef`**, dont la structure est présentée sur la figure 3.17, sert à assurer le passage dans le sens inverse (le *unparsing,* soit la conversion d'une structure de données XML en format physique) en effectuant le traitement décrit par l'attribut **`dfdl:outputValueCalc`**.

```
<xs:group name="hiddenRawData">
  <xs:sequence>
    <xs:element name="rawData" type="afdx:uLong" dfdl:length="16"
      dfdl:outputValueCalc="{ ../Data_Value div $afdx:data-range }" />
  </xs:sequence>
</xs:group>
```

Figure 3.17 Schéma DFDL qui décrit le *unparsing* du champ data

Le label d'un mot ARINC 429 est toujours représenté sur trois chiffres. La figure 3.18 présente les valeurs du poids de chaque bit du label.

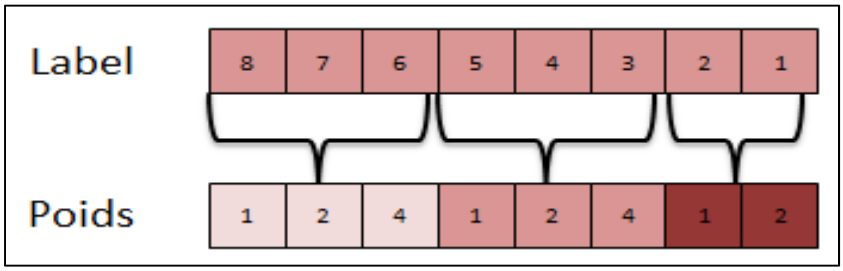


Figure 3.18 Poids des bits du label d'un mot ARINC 429

Ce bout de code DFDL représente l'élément **`Label`** de l'élément **`ARINC429Data`** décrit précédemment sur la figure 3.16 et la manière de le décoder[9].

```
                <xs:sequence dfdl:hiddenGroupRef="afdx:hiddenLabel"/>
                <xs:element name="Label" type="afdx:bit" dfdl:length="8"
                            dfdl:inputValueCalc="{ (../bit1 +  ../bit2 * 2 + ../bit3 * 4)+
                            ((../bit4 +  ../bit5 * 2 + ../bit6 * 4 ) * 10 )+ ((../bit7 +  ../bit8 * 2 )* 100)
                            }"/>
<xs:group name="hiddenLabel">
  <xs:sequence>
    <xs:element name="bit1" type="afdx:bit" dfdl:length="1"
                dfdl:outputValueCalc="{ (../Label mod 10 ) div 1 mod 2}" />
    <xs:element name="bit2" type="afdx:bit" dfdl:length="1" dfdl:outputValueCalc="{ (../Label mod 10) div 2 mod 2  }"/>
    <xs:element name="bit3" type="afdx:bit" dfdl:length="1" dfdl:outputValueCalc="{ (../Label mod 10) div 4  mod 2}"/>
    <xs:element name="bit4" type="afdx:bit" dfdl:length="1" dfdl:outputValueCalc="{ ((../Label div 10) mod 10) div 1 mod 2  }"/>
    <xs:element name="bit5" type="afdx:bit" dfdl:length="1" dfdl:outputValueCalc="{ ((../Label div 10) mod 10) div 2 mod 2 }"/>
    <xs:element name="bit6" type="afdx:bit" dfdl:length="1" dfdl:outputValueCalc="{ ((../Label div 10) mod 10) div 4 mod 2 }"/>
    <xs:element name="bit7" type="afdx:bit" dfdl:length="1" dfdl:outputValueCalc="{ (../Label div 100) div 1 mod 2  }"/>
    <xs:element name="bit8" type="afdx:bit" dfdl:length="1" dfdl:outputValueCalc="{ (../Label div 100) div 2 mod 2  }"/>
  </xs:sequence>
</xs:group>
```

Figure 3.19 Schéma DFDL de l'élément **`Label`**

3.4 Résultat de l'extraction des informations de la trame AFDX

Le *parsing* Daffodil se fait par la commande suivante en indiquant comme paramètres d'entrée le fichier du schéma DFDL et le fichier de la trame AFDX (décodée en hexadécimal).

[9] https://github.com/rimbendhaou/basic_afdx_adiru/blob/master/Afdx_Adiru_Model/DFDL/

```
daffodil parse --schema DFDL_schema.xsd --output formatted_frame.xml
AFDX_frame.afdx
```

Dans cet exemple le résultat (*output*) du *parsing* va être sauvegardé dans un fichier XML (**`formatted_frame.xml`**). La figure 3.20 illustre le processus du *parsing* de Daffodil.

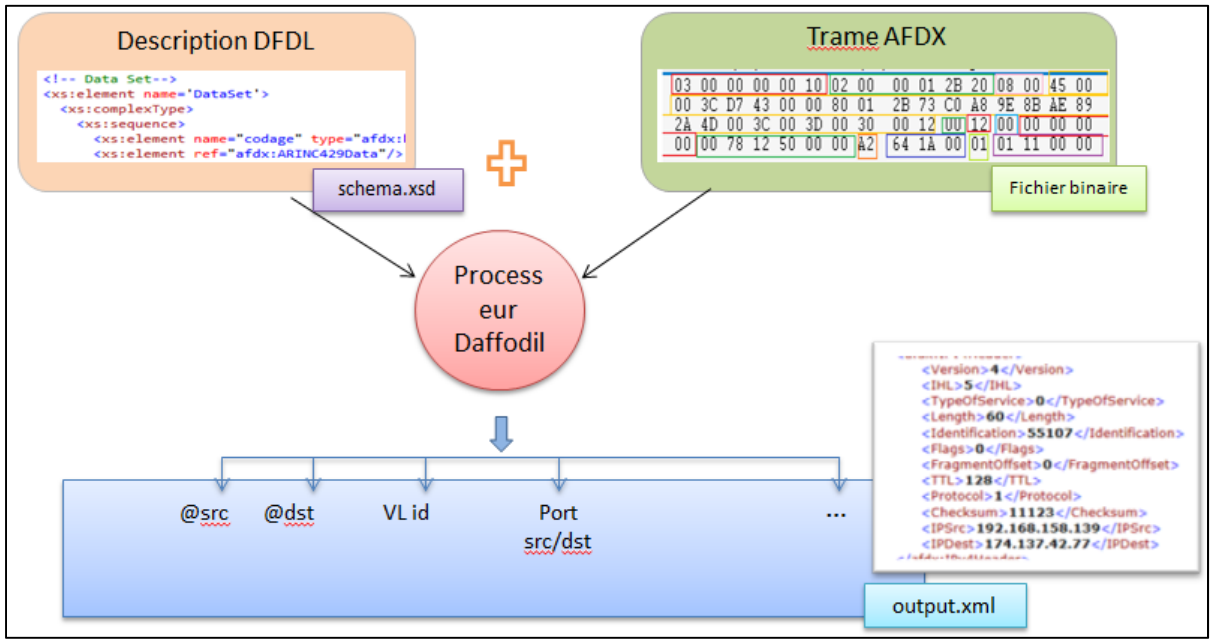


Figure 3.20 Processus du *parsing* de Daffodil

Le résultat du *parsing* est un fichier XML contenant toutes les informations décodées à partir de la trame AFDX de test. Ce fichier contient les informations structurées dans des balises crées et adaptées à la structure définie dans le schéma DFDL. Le fichier XML est facile à gérer et à interpréter. Il est interopérable avec d'autres outils pour l'extraction des données. Dans notre cas, nous visualisons toutes les données nécessaires en utilisant le langage XSLT (*Extensible Stylesheet Language Transformations*), en transformant le contenu XML en HTML. Nous obtenons ainsi une page Web formatée et lisible. La figure 3.21 montre le résultat de l'extraction des informations à partir du résultat XML. Ce résultat représente les informations extraites d'une trame AFDX. Ces informations sont une partie des informations que nous considérons importantes à inclure dans les ICD et qui présentent les messages

échangés. La présentation est sous format tabulaire puisque ce format est actuellement très utilisé par les intégrateurs en avionique.

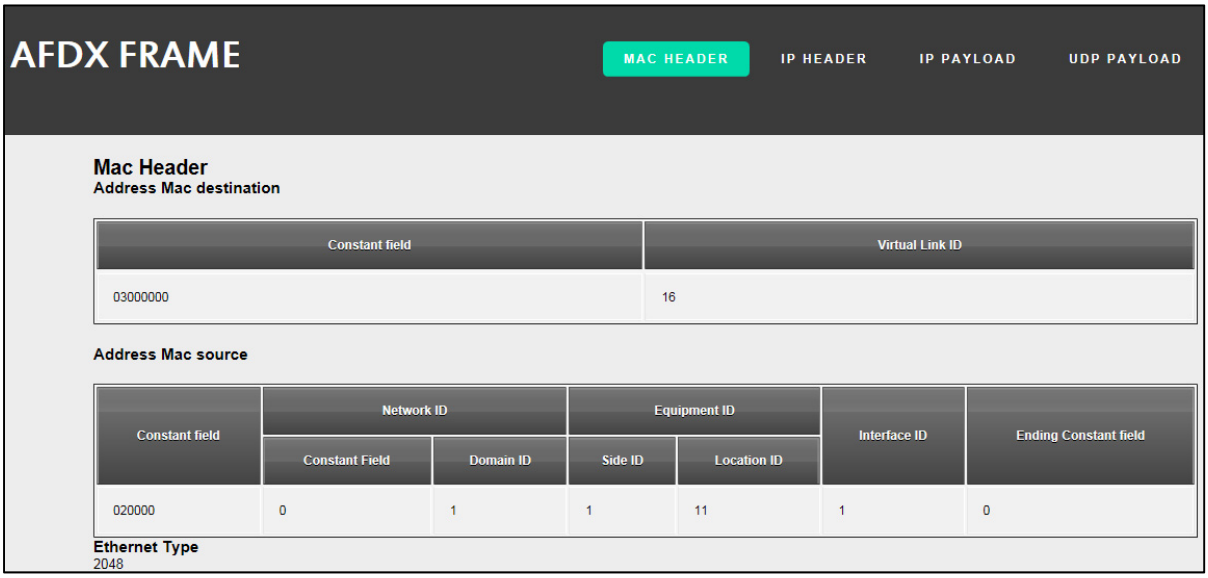


Figure 3.21 Résultat de l'extraction des informations relatives à l'entête Ethernet

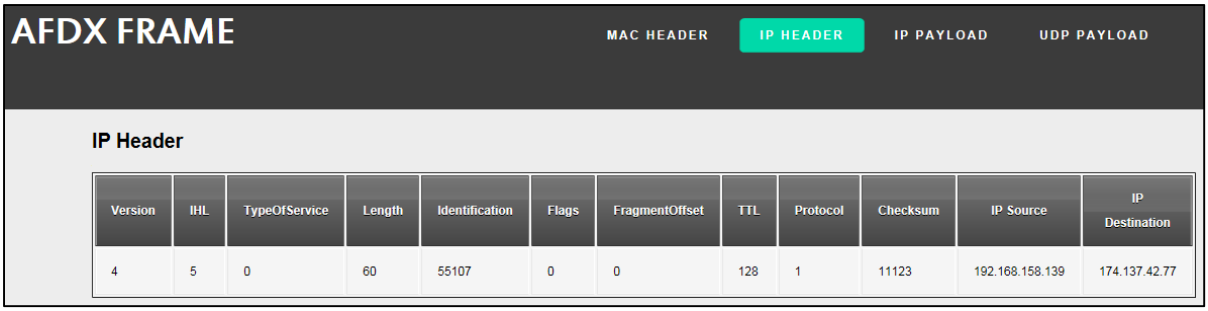


Figure 3.22 Affichage du résultat de l'extraction des informations de l'entête IP

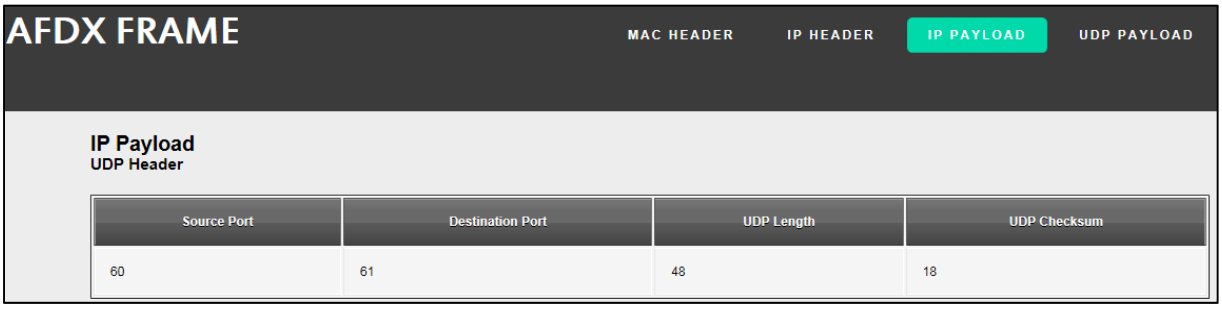


Figure 3.23 Affichage du résultat de l'extraction des informations relatives à l'*IP Payload*

Dans cet exemple, nous avons pris comme cas de test le mot ARINC 429 *total pressure* ayant le label 242.

Bit No. Parameter	32	31 30 29 SSM	28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 MSB Data Field LSB	10 9 SDI	8 7 6 5 4 3 2 1 Label (1 2 4) (1 2 4) (1 2)
Total Pressure: 1050 mb	0	1 1 0	1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 P P	0 0	0 1 0 0 0 1 0 1

Figure 3.24 Exemple de codage du mot ARINC 429 *total pressure.*

Figure extraite de (AEEC., 2001)

Le résultat de l'affichage du mot ARINC 429 est présenté par la figure 3.25.

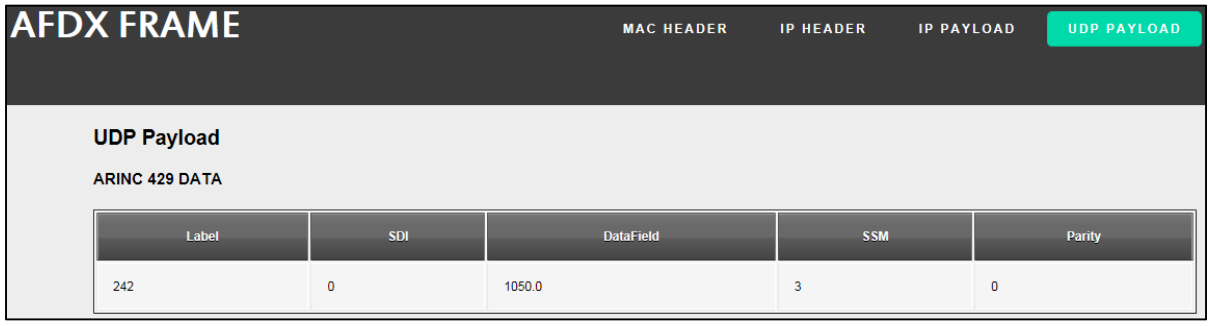


Figure 3.25 Résultat de l'affichage du mot ARINC 429

Daffodil nous a permis ainsi d'expérimenter le chemin inverse (l'*unparsing*) pour retrouver la trame AFDX à partir du fichier résultant XML (**`formatted_frame.xml`**) et de la description DFDL. La commande suivante est utilisée pour faire le processus d'*unparsing*.

```
daffodil unparse --schema schema.xsd --output AFDX_frame.afdx
formatted_frame.xml
```

La trame en sortie (**`AFDX_frame.afdx`**) est bien équivalente à celle que nous avons spécifiée en entrée du processus de *parsing*.

3.5 Intégration des modèles AADL et DFDL

Dans le chapitre précèdent, nous avant décrit le modèle AADL de l'ADIRU. Nous avons défini le réseau AFDX, les partitions et les ports à travers lesquels les données sont échangées. Les données sont circulées dans des trames AFDX dont nous avons défini la structure avec DFDL.

Pour faire le lien entre le modèle AADL et le schéma DFDL, nous avons défini de nouvelles propriétés dans AADL qui relient le fichier du schéma DFDL et le fichier de la trame binaire aux ports et aux données correspondants (figure 3.26).

```
property set ARINC664 is

--Name of the DFDL Frame Schema File
Frame_Schema : aadlstring applies to (port,data);

--Name of the AFDX Frame File
Frame_AFDX : aadlstring applies to (port,data);

end ARINC664;
```

Figure 3.26 Définition des propriétés DFDL

Ces propriétés offrent une convention de nommage où le nom du fichier de la description de la structure de la trame cité en AADL doit correspondre au nom du schéma DFDL. Le fichier du schéma DFDL contenant la définition de la structure de la trame est associé aux ports appropriés qui sont les ports qui échangent les données contenues dans la trame. Le fichier de la trame AFDX mentionné dans la propriété **`Frame_AFDX`** dans AADL doit aussi correspondre au fichier binaire de la trame contenant le **`data`** échangé à travers le port correspondant. La figure 3.27 illustre un exemple de l'usage de la convention de nommage.

```
-------------------------------------------------------------
--- 2.1 ADIRU Process 1
-------------------------------------------------------------
  PROCESS ADIRU_Process1
    FEATURES
      P1_in1 : IN DATA PORT;
      P1_out1 : OUT DATA PORT;
      P1_out2 : OUT DATA PORT DataType::Pressure
      { ARINC664::Frame_Schema => "schema_DFDL.xsd";
        ARINC664::Frame_AFDX   => "frame_AFDX.afdx"; };
  END ADIRU_Process1;
```

Figure 3.27 Association des propriétés DFDL

3.6 Conclusion

Dans ce chapitre, la solution que nous avons adoptée pour décrire le format des données transmises dans le réseau AFDX a été présentée. Cette expérimentation a montré que DFDL peut être un langage efficace pour décrire les trames AFDX. Le processeur Daffodil que nous avons utilisé pour la lecture des données a aussi prouvé ses performances dans les deux sens (*parsing* et *unparsing* des données). La description du format des données avec DFDL sert à compléter le modèle AADL en lui ajoutant les informations relatives aux données échangées. Avec la solution que nous proposons, un intégrateur peut utiliser le langage DFDL pour définir la structure de ces données échangées et joindre les fichiers de schéma DFDL aux ports qui échangent ces données. En complétant AADL avec DFDL, l'intégrateur obtient donc le modèle AADL de son sous-système qui contient l'architecture du système, et les entrées/sorties (messages) avec les schémas DFDL décrivant les données échangées. Nous avons proposé ainsi une méthode pour modéliser les trames AFDX en DFDL et une manière d'intégrer la description DFDL dans AADL. Nous avons présenté une partie des ICD ainsi extraits de nos modèles, soit les informations pertinentes sur les messages échangés, en le convertissant dans un format convivial.

CONCLUSION

Ce travail de recherche a été réalisé dans le contexte du projet CRIAQ AVIO-506, qui portait sur la gestion des ICD de systèmes avioniques. Notre travail concerne plus particulièrement l'expérimentation du langage AADL afin de modéliser les systèmes avioniques IMA en nous concentrant sur les interfaces et les informations pertinentes qui doivent être incluses dans les ICD. Notre objectif était d'évaluer les capacités d'AADL et son écosystème d'outils libres dans la modélisation des informations relatives aux interfaces et dans la validation de la conformité de ces modèles, entre autres par rapport aux exigences des normes ARINC.

Pour atteindre notre objectif, nous avons commencé notre projet par une étude bibliographique du langage AADL pour nous familiariser avec sa syntaxe, ses composants, et ses outils. Nous avons développé aussi une connaissance autour des systèmes avioniques pour comprendre leurs architectures, fédérée et IMA, et les normes ARINC les plus pertinentes.

Une étape de collecte d'information a été présentée dans les deux contextes, fédérés et IMA, afin de comprendre les différences en termes d'interfaces dans chacun de ces deux contextes.

Ensuite, une étude de cas a été menée pour modéliser un ADIRU dans un contexte IMA. Dans la modélisation, nous avons profité de deux exemples de modèles AADL réalisés dans d'autres travaux de recherche ayant des objectifs différents. Le premier s'intéresse à la modélisation d'un système IMA et utilise l'annexe officielle ARINC 653 d'AADL et le deuxième s'intéresse à modéliser le réseau AFDX dans le but de proposer une annexe officielle liée entre autres aux ARINC 664. Nous avons donc essayé de combiner les deux aspects ARINC 653 et ARINC 664 dans notre modèle pour avoir un modèle plus complet. En effet, pour modéliser la structure du réseau AFDX (les bus, les commutateurs, les partitions et les plateformes d'exécution) et les propriétés (table de routage, *virtual link*), nous avons adopté le modèle proposé par Khoroshilov puisqu'il reflète plus la structure des systèmes IMA. Ensuite, nous avons intégré les propriétés ARINC 653 pertinentes à notre modèle (comme les propriétés du partitionnement). Pour valider la couverture de notre modèle aux

aspects ARINC 653 et 664, le langage RESOLUTE supporté par l'outil OSATE2 a été exploité.

En premier lieu, nous avons fait recours aux règles RESOLUTE définies par Hugues et Delange pour vérifier la conformité de notre modèle à l'annexe ARINC 653. Nous avons démontré qu'il y en a une incompatibilité entre les deux annexes (l'annexe officielle ARINC 653 d'AADL et l'annexe potentielle ARINC 664). L'incompatibilité se manifeste dans le déploiement des partitions dans les plateformes d'exécution. Selon Hugues et Delange, les partitions virtuelles doivent être définies dans les processus physiques où l'intégrateur du sous-système peut spécifier les propriétés du partitionnement temporel. Par contre, selon Khoroshilov, les partitions (processeurs virtuels) sont reliées aux processeurs physiques par une propriété de *Binding*. Nous avons adopté le modèle de Khoroshilov parce qu'il supporte plus le principe de flexibilité dans l'intégration des partitions logicielles IMA dans les plateformes matérielles.

En deuxième lieu, pour valider la conformité de notre modèle aux normes ARINC 664 nous avons défini les règles RESOLUTE jugées pertinentes et nous avons vérifié la satisfaction de notre modèle à ces règles.

À travers notre expérience de modélisation, nous avons trouvé qu'AADL est un langage de modélisation convivial pour la modélisation des systèmes avioniques. Il nous a permis de modéliser des concepts du standard ARINC 653 à l'aide de l'annexe AADL associée. Nous avons donc pu décrire le principe de partitionnement et le principe de séparation spatiale et temporelle. Nous avons défini également les communications inter-partition et affecté les caractéristiques nécessaires aux ports (échantillonnage ou file d'attente). Nous avons profité notamment d'une annexe AFDX en cours d'élaboration pour décrire la structure de l'architecture et les composants du réseau AFDX et pour assigner des propriétés pertinentes aux éléments du réseau comme les *VL* et les commutateurs.

Un défi important que nous avons rencontré avec AADL est la description de la structure et du format des données échangées à travers les ports. En effet, AADL décrit les données par un simple composant **`data`** sans aucune information sur la structure de ce **`data`**. Même en

ajoutant des propriétés spécialisées à des éléments **`data`**, ceux-ci restent insuffisant pour décrire les informations pertinentes à inclure dans les ICD de systèmes avioniques. Pour complémenter les modèles AADL nous avons utilisé le langage DFDL pour définir la structure de ces informations et joindre les fichiers de schéma DFDL aux ports qui échangent ces données. Nous obtenons donc comme résultat un modèle AADL du sous-système qui contient l'architecture du système et les entrées/sorties (messages) avec les schémas DFDL décrivant les données échangées. Cette expérimentation a montré que DFDL, qui est conçu pour résoudre les problèmes d'échange de données entre applications différentes, peut être un langage efficace pour décrire les trames AFDX. Le processeur Daffodil que nous avons utilisé pour la description des données a prouvé ses performances dans les deux sens (*parsing* et *unparsing* des données). La description du format des données avec DFDL a donc servi à compléter le modèle AADL en lui ajoutant les informations relatives aux données échangées. À la fin, nous avons utilisé XSLT pour convertir les sorties du compilateur Daffodil (qui sont des fichiers XML) en pages HTML sous format de tableaux qui est le format le plus utilisé et apprécié par les industriels pour la présentation des ICD.

À travers ce travail de recherche, nous avons fait des contributions en proposant un modèle AADL qui contient à la fois les aspects ARINC 653 (IMA) et les aspects ARINC 664 (AFDX). Le modèle proposé a montré une incohérence entre l'annexe officielle ARINC 653 et une nouvelle ébauche d'annexe ARINC 664 (celle de Khoroshilov). Nous avons proposé une méthode pour modéliser les trames AFDX en DFDL et une approche d'intégration de la description DFDL dans AADL en introduisant des propriétés spécifiques. De plus, nous avons extrait de nos modèles certaines des informations pertinentes des ICD, qui sont les messages échangés, en les convertissant dans un format convivial.

RECOMMANDATIONS

Dans une perspective de proposer aux industriels une solution peu couteuse ou libre pour la gestion des ICD, des travaux futurs de recherche peuvent être menés afin d'améliorer l'exploitation d'AADL dans l'extraction automatique des ICD.

En effet, dans notre expérimentation nous avons choisi le système ADIRU qui interagit avec un FMS et un RDC comme preuve de concept pour vérifier si c'est utile de les modéliser avec AADL afin d'extraire les informations des interfaces. Nous avons eu des résultats prometteurs mais il reste à valider notre solution avec des industriels pour vérifier sa pertinence en termes de temps de réalisation et de coûts. La validation expérimentale doit être menée avec de vrais intégrateurs afin de comparer notre approche avec leurs outils déjà existants et vérifier si l'approche proposée réduit les coûts de production des sous-systèmes avioniques.

Il est aussi recommandé de diversifier les cas d'exemples des sous-systèmes avioniques pour en sortir avec une confirmation plus solide qu'AADL permet de modéliser les différents cas des systèmes avioniques quel que soit sa spécificité.

Concernant la validation avec RESOLUTE, nous concluons que ce langage est performant et flexible pour traduire les règles de validation que nous avons besoin. Il permet de parcourir tout le modèle AADL y compris les composants et ses sous-composants, les propriétés, les connexions et les ports. De futurs travaux peuvent être réalisés pour l'exploiter dans la validation des intégrations entre les sous-systèmes.

Dans notre projet nous avons relié le modèle AADL et la description des données DFDL en associant le nom du fichier contenant le schéma DFDL aux ports des modèles AADL. Cependant nous recommandons comme futur travail d'améliorer l'intégration entre la description DFDL et le modèle AADL pour la rendre plus interactive et plus automatisée.

ANNEXE I

Descriptions des équipements avioniques cités dans ce mémoire

L'annexe contient les définitions et descriptions des équipements avioniques cités dans notre mémoire.

ILS

L'ILS est un récepteur qui reçoit des signaux et les traite pour fournir des informations de guidage de l'aéronef lors de son atterrissage. Le schéma suivant explique l'interaction de l'ILS avec les autres systèmes ou équipements.

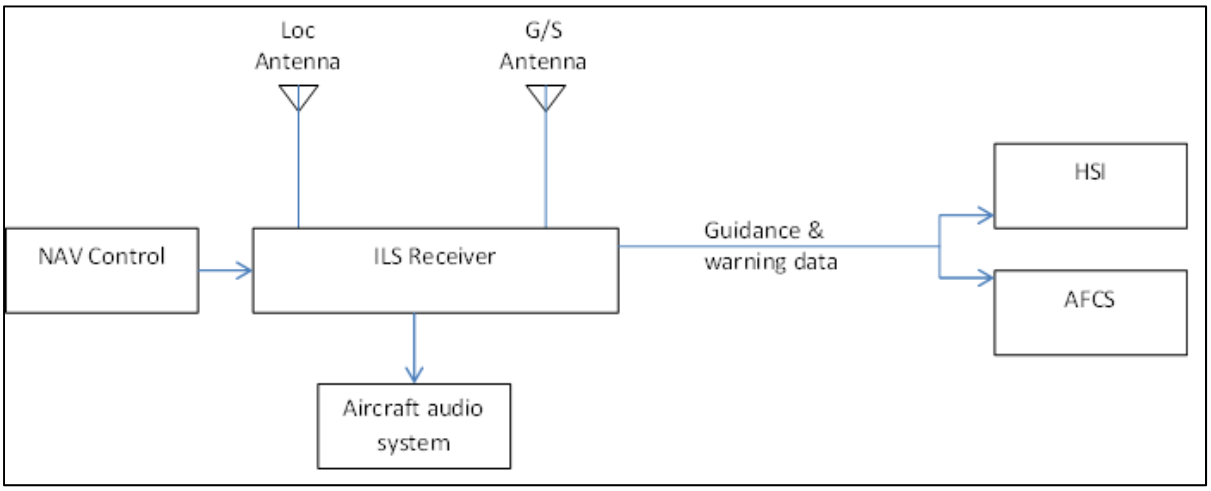


Figure A.I.1 Schéma général de l'ILS. Source: (http://aelmahmoudy.users.sourceforge.net/electronix/egair/radar.htm)

L'ILS traite des signaux comme les signaux du *Localizer* et du *Glide slope* et les transforme en données digitales. Il fournit ces données de guidage au pilote et à l'*Automatic Flight Control System* (AFCS).

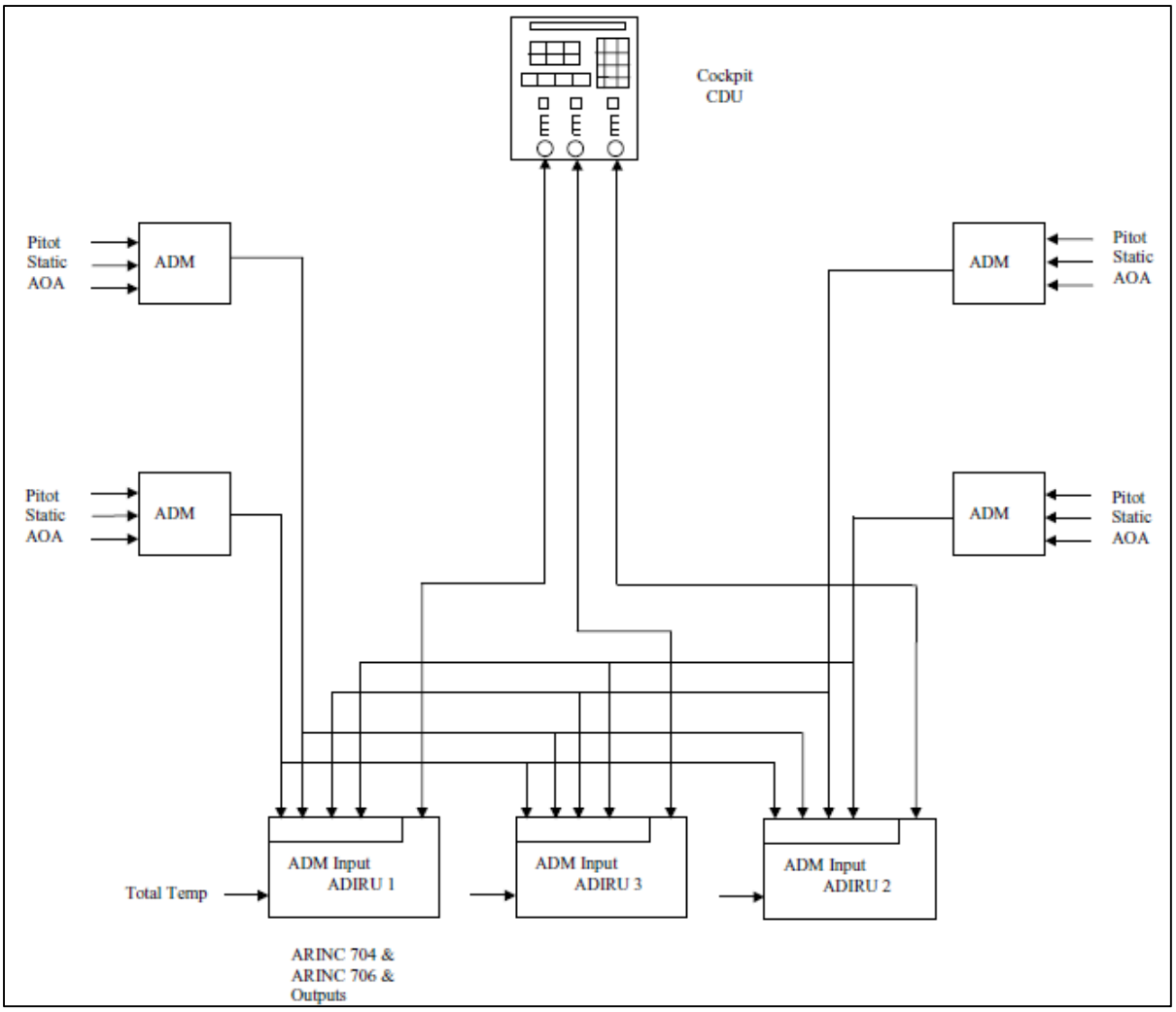
Exemple de données transmises par l'ILS selon le document de spécification de l'ILS (ARINC 710).

Parameter	Label		ILS Operational Range	Max Transmit Interval msec	BNR Word Range	Binary Output		BCD Output	
	Type	No.				Sig Bits	Resolution Units	Sig Digs	Least Signif Digits
Localizer Deviation	BNR	173	±0.155 DDM (min)	62.5	±0.4	12	0.0001 DDM	–	–
Glide Slope Deviation	BNR	174	±0.22 DDM (min)	62.5	±0.8	12	0.0002 DDM	–	–
ILS Frequency	BCD	033	108.00 - 111.95	200	–		–	4	0.01 MHz
Runway Heading	BCD	017	0 - 359.9°	200	–	–	–	–	–
Runway Heading	BNR	105	0 - 359.9°	50	±180°	11	0.09°	4	0.1°
ILS Ground Station Ident	Special	263 & 264	–	200	–	–	–	–	–

Figure 3.I.2 Data word standards (output de l'ILS) (AEEC., 1997)

ADIRU

L'ADIRU est une unité de référence inertielle anémobarométrique. C'est un équipement qui combine deux fonctionnalités principales considérées souvent comme deux sous-systèmes: la centrale anémobarométrique et le système de référence inertielle. L'ADIRU fait partie de l'*Air Data Inertial Reference System* (ADIRS). Dans un ADIRS, on peut trouver jusqu'à trois ADIRU redondants pour avoir une meilleure précision et augmenter la fiabilité. Le schéma suivant représente un exemple d'architecture de l'ADIRS avec trois installations d'ADIRU. La figure est prise du document de spécification ARINC 738 (de son annexe 2-2).



3.I.3 Exemple d'architecture de l'ADIRS avec trois ADIRU (AEEC., 2001)

L'ADIRU est un système ayant deux fonctionnalités principales, souvent considérées comme deux sous-systèmes: l'*Inetrtial Reference System* (IRS) et l'*Air Data System* (ADR).

FMS

FMS: Flight Management System ou système de gestion de vol est un système qui fournit des renseignements pour assister le pilote pendant le vol. Les informations délivrées par ce système sont reliées à la navigation, au pilotage, à la consommation, etc. Citons à titre d'exemple d'informations fournies par le FMS: la position actuelle, la distance restante, le temps d'arrivé estimé, etc.

La figure suivante montre la relation entre le FMS et l'ADIRU. Dans cet exemple, l'ADIRU est représenté par ses deux sous-systèmes, l'IRS et l'ADS, marqués en rouge dans la figure.

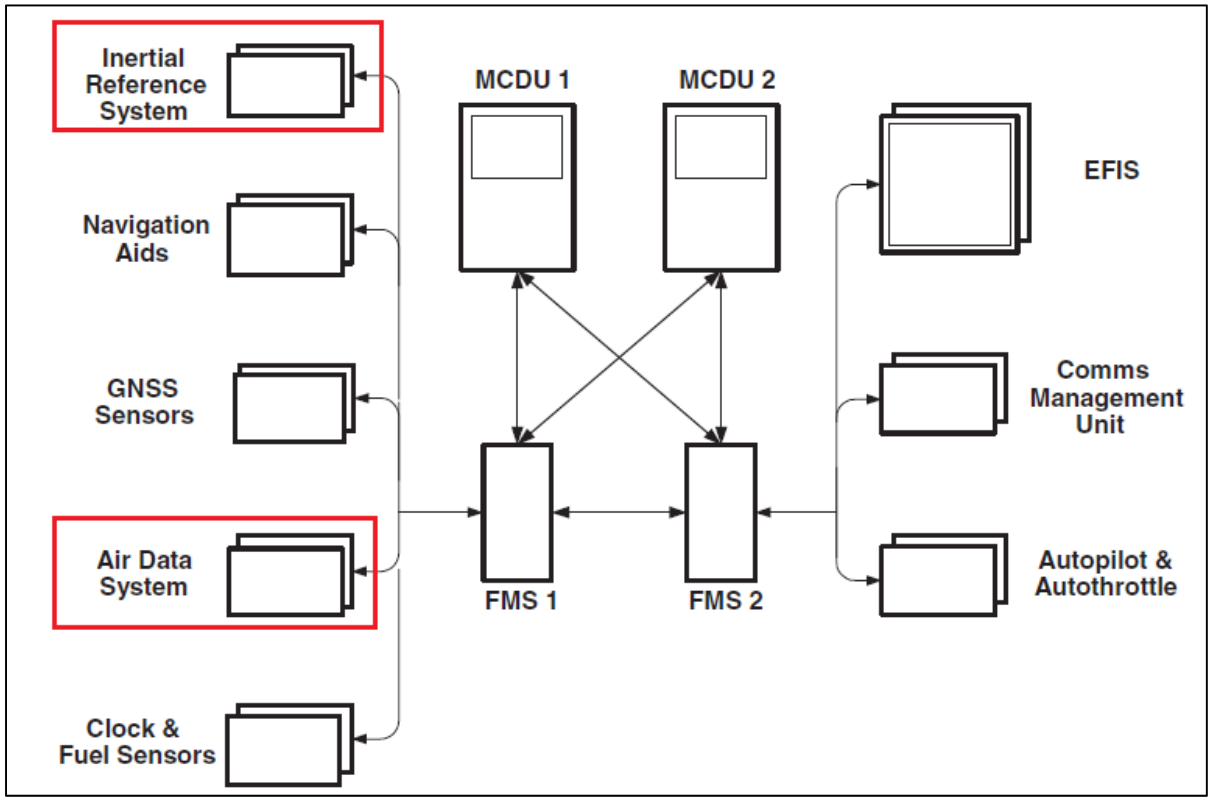


Figure 3.I.4 Flight Management System (Moir, I *et al*., 2013)

ANNEXE II

ADIRU

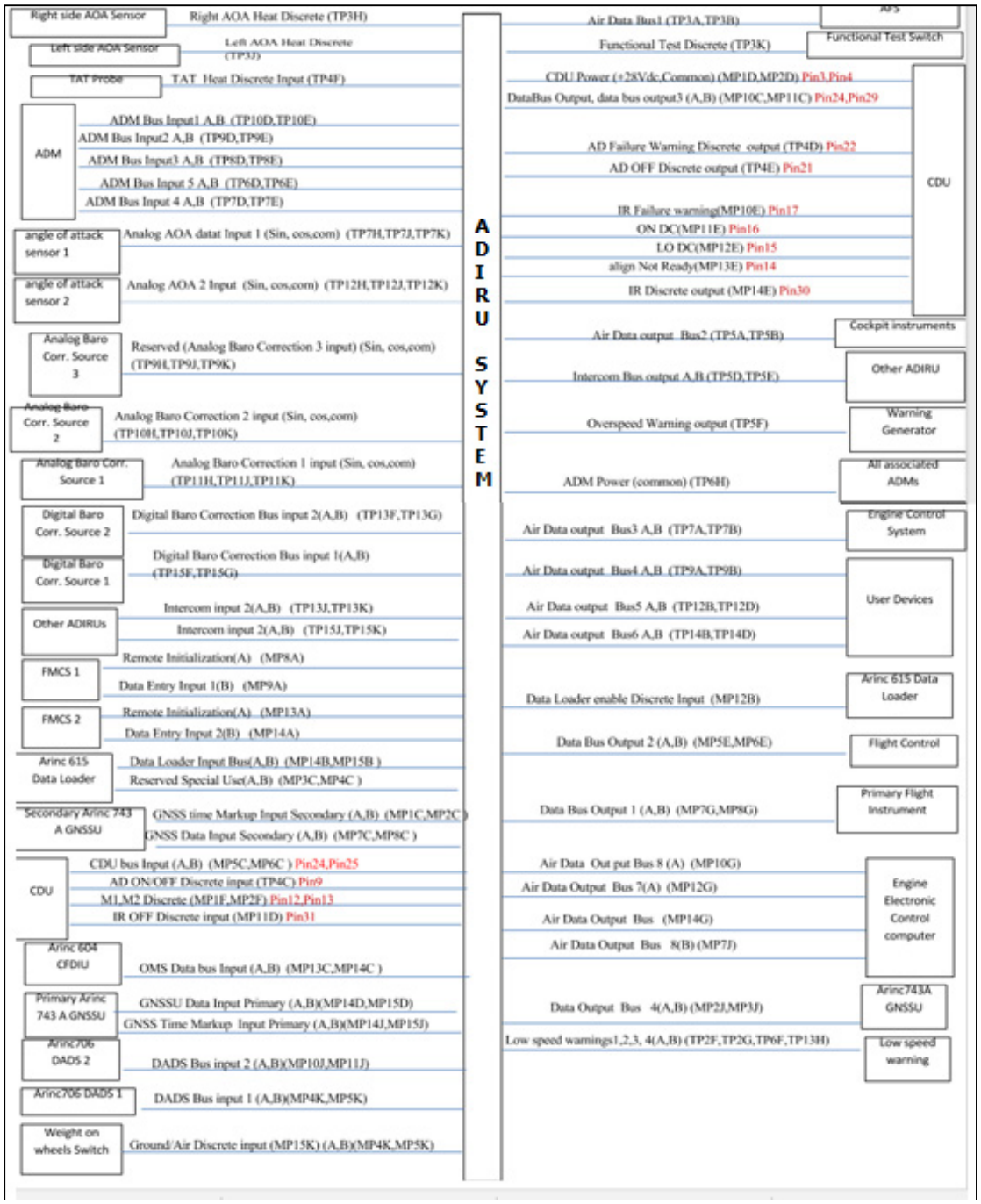


Figure 3.II.1 Interconnexions de l'ADIRU avec les autres sous-systèmes

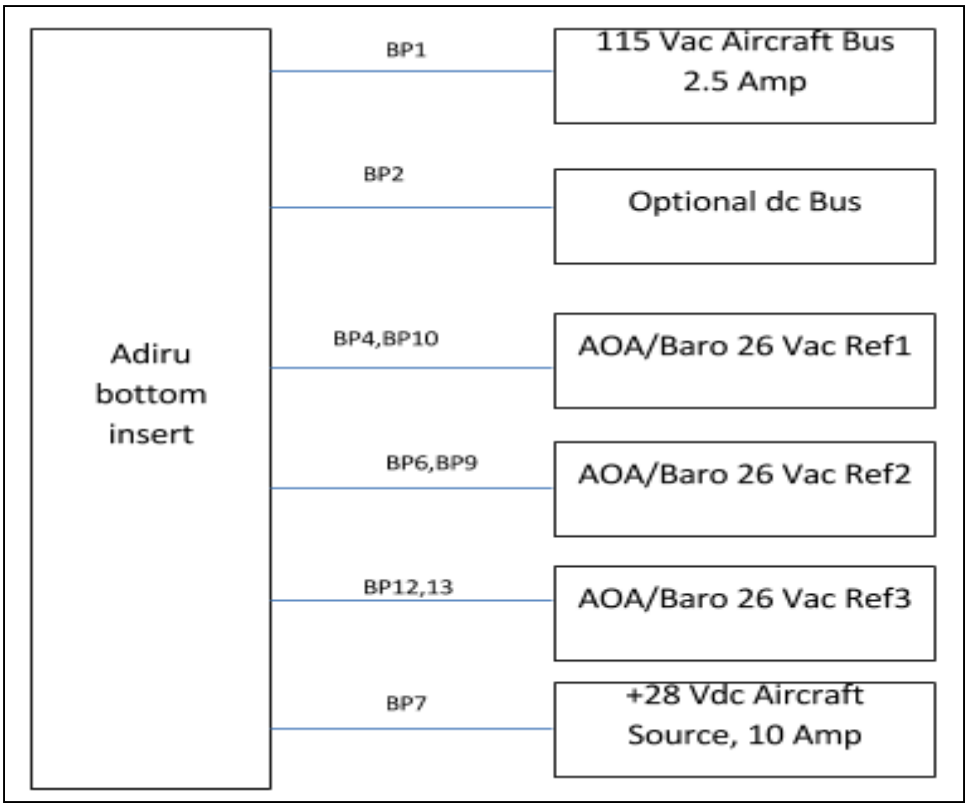


Figure A.II.2 Schéma des connecteurs (ADIRU Bottom Insert)

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

(AEEC., 1997) AEEC., 1997b. ARINC-710-10: MARK 2 airborne ILS receiver. Aeronautical Radio.

(AEEC., 2001) AEEC., 2001. ARINC-738A-1: Air Data and Inertial Reference System. Aeronautical Radio.

(AEEC., 2009) AEEC., 2009a. ARINC-664-P7: Aircraft data network part 7 avionics full-duplex switched Ethernet network. Aeronautical Radio.

(AEEC., 2010) AEEC., 2010. ARINC 653: Avionics Application software standard interface. Aeronautical Radio.

(AEEC., 2012) AEEC., 2012. ARINC 429P1-18: Digital Information Transfer System (DITS) part 1 functional description, electrical interfaces, label assignments and word formats. Aeronautical Radio.

(Alena R. *et al*., 2007) Alena, R. L., Ossenfort, J. P., Laws, K. I., Goforth, A., & Figueroa, F. (2007, March). Communications for integrated modular avionics. In 2007 IEEE Aerospace Conference, pp. 1-18.

(An, D. *et al*., 2015) An, D., Kim, K. H., & Kim, K. I. (2015). Optimal configuration of virtual links for avionics network systems. International Journal of Aerospace Engineering, 2015.

(Ananda C.M. *et al*., 2013) Ananda, C. M., Nair, S., & Mainak, G. H. (2013). ARINC 653 API and its application-An insight into Avionics System Case Study. Defence Science Journal, 63(2), pp. 223-229.

(Evensen, K., & Weiss, K., 2010) Evensen, K., & Weiss, K. (2010). A comparison and evaluation of real-time software systems modeling languages. In AIAA Infotech@ Aerospace 2010, April 20-22 2010, Atlanta, Georgia.

(Feiler, P., 2004) Feiler, P. (2004, October). Open source AADL tool environment (OSATE). In AADL Workshop, Paris, pp. 1-40.

(Feiler, P. H., Lewis, B., & Vestal, S., 2004). Feiler, P. H., Lewis, B., & Vestal, S. (2004). The SAE avionics architecture description language (AADL) standard: A basis for model-based architecture-driven embedded systems engineering. In IFIP TC-2 Workshop on Architecture Description Language, WADL 2004, August 22- 27, 2004, Toulouse, France, Vol. 176, pp. 3-15.

(Feiler, P. H., & Gluch, D. P., 2012). Feiler, P. H., & Gluch, D. P. (2012). Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language. Addison-Wesley.

(Hugues, J., & Delange, J., 2015) Hugues, J., & Delange, J. (2015). Modeling and Analyzing IMA Architectures with AADL, From Modeling to Safety Evaluation and Code Generation: A Case-Study. In: Proceedings of SAE Aerotech 2015, September 22-24 2015, Seattle, USA.

(Khoroshilov A., 2012) Khoroshilov, A., Albitskiy, D., Koverninskiy, I., Olshanskiy, M., Petrenko, A., & Ugnenko, A. (2012). AADL-based toolset for IMA system design and integration. SAE International Journal of Aerospace, 5(2012-01-2146), pp.294-299.

(Larmouth, J., 2000). Larmouth, J., (2000). ASN. 1 complete. USA: Morgan Kaufmann Publishers.

(Le Sergent, T., & Le Guennec, A., 2014) Le Sergent, T., & Le Guennec, A. (2014). Data-Based System Engineering: ICDs management with SysML. In Embedded Real Time Software and Systems conference, 2014.

(Louadah, H. *et al*., 2014) Louadah, H., Champagne, R., & Labiche, Y. (2014, September). Towards Automating Interface Control Documents Elaboration and Management. In 7th International Workshop on Model-Based Architecting and Construction of Embedded Systems, ACES-MB 2014, Co-located with ACM/IEEE 17th International Conference on

Model Driven Engineering Languages and Systems, MoDELS 2014 (Valencia, Spain) (pp. 26-33).

(Moir, I *et al*., 2013) Moir, I., Seabridge, A., & Jukes, M. (2013). Civil avionics systems. John Wiley & Sons.

(Pajares, M. *et al*., 2010) Pajares, M. Á. M., Díaz, C. M., Pastor, I. L., & de la Hoz, C. F. ICD Management (ICDM) tool for embedded systems on aircrafts. In Embedded Real Time Software and Systems Conference (2010).

(Rahmani, K., et Thomson, V., 2009) Rahmani, K., & Thomson, V. (2009). New interface management tools and strategies for complex products. In The 6th International Product Lifecycle Management Conference (PLM09) (p. 10).

(SAE International, 2009) SAE International, Platform/Subsystem Common Interface Control Document Format, standard AS5658, 2009.

(Samolej, S., 2011) ARINC Specification 653 Based Real-Time Software Engineering. e-Informatica, 5(1), 39-49.

(SEI, 2004) OSATE, S. (2004). An extensible source AADL tool environment. SEI AADL Team Technical Report, December 2004, http://www.aadl.info/aadl/documents/AADLToolUserGuide0-3-0.pdf

(Watkins, C. B., et Walter, R., 2007) Watkins, C. B., & Walter, R. (2007, October). Transitioning from federated avionics architectures to integrated modular avionics. In 26th IEEE/AIAA Digital Avionics Systems Conference, October 21-25 2007, pp. 1-10. Piscataway, NJ, USA: IEEE.