MODELISATION D'EXIGENCES POUR LA SYNTHESE D'ARCHITECTURE AVIONIQUE : APPLICATION A LA SURETE DE FONCTIONNEMENT

L. ZIMMER, M. LAFAYE

Dassault Aviation – Direction Générale Technique
78 quai Marcel Dassault - 92552 Saint-Cloud - France
laurent.zimmer@dassault-aviation.com
michael.lafaye@dassault-aviation.com

P.A. YVARS

Institut Supérieur de Mécanique de Paris (SupMéca)
QUARTZ
3 rue Fernand Hainaut – 93407 Saint-Ouen - France
pierre-alain.yvars@supmeca.fr

RESUME : Les plateformes informatiques embarquées doivent supporter l'exécution d'un nombre croissant de fonctions systèmes et respecter scrupuleusement de multiples contraintes fonctionnelles et non fonctionnelles. Leur architecture est donc de plus en plus complexe et par conséquent la définition préliminaire d'architectures admissibles basées sur la compétence et l'expérience d'un petit nombre de concepteurs experts est une activité de plus en plus difficile. A l'avenir, ils devront s'aider d'outils d'assistance à la génération d'architectures correctes par construction. Dans ce contexte, nous nous sommes intéressés à la modélisation formelle des exigences et à la résolution informatique de ces modèles formels pour trouver des architectures admissibles. Nous avons étudié une problématique de déploiement assisté de fonctions systèmes sur une plateforme d'avionique modulaire embarquée avec des exigences de sûreté de fonctionnement portant sur les fonctions. Nous avons développé une approche à base de modèle qui utilise le langage DEPS (DEsign Problem Specification), un langage dédié à la modélisation et à la résolution des problèmes de conception. Les résultats obtenus montrent qu'il est possible de modéliser des exigences complexes de sûreté de fonctionnement au niveau requis par l'architecte système et que leur prise en compte pendant la résolution génère des solutions correctes vis-à-vis de celles-ci.

MOTS-CLES : *modélisation, synthèse d'architecture, spécification formelle, déploiement, exigences, sûreté de fonctionnement.*

1 INTRODUCTION

Les systèmes avioniques sont de plus en en plus complexes. D'après [1], dans le domaine militaire nous sommes passés de 15 sous-systèmes et moins de 40% des fonctions systèmes à dominante logicielle pour le F16 à 135 sous-systèmes dont 90% des fonctions à dominante logicielle pour le F35. L'évolution dans le domaine civil est moins extrême mais elle suit la même tendance. En parallèle le nombre et la complexité des spécifications techniques ou réglementaires ont augmenté tout comme la complexité de l'organisation industrielle. Cet accroissement global de la complexité engendre une explosion des coûts et des délais de développement qui amène les industriels à revoir leurs méthodes et leurs outils de conception. Selon des études financées par la DARPA [1, 2, 3], le nouveau processus de conception à imaginer doit reposer sur le développement de 4 éléments clefs :

1. des outils de conception à base d'abstraction ;
2. des métriques de complexité des systèmes ;
3. des méthodes avancées de synthèse d'architecture ;
4. une gestion robuste des incertitudes.

Le travail présenté concerne exclusivement le premier [2] et le troisième point [3]. Il s'agit pour le premier point de développer ou d'utiliser un langage de description formelle (FDL) capable de représenter à la fois un système complexe ainsi que les spécifications ou exigences qui portent sur celui-ci. Le FDL doit notamment permettre de représenter le système à des niveaux d'abstraction compatibles des différentes étapes de la conception et notamment les étapes de conception préliminaire. Les langages envisagés sont AADL, UML ou SysML.

Il s'agit pour le deuxième point de disposer très tôt dans la conception de moyens « avancés de synthèse d'architecture » qui permettraient d'explorer automatiquement l'espace de conception pour rechercher des architectures admissibles c'est-à-dire compatibles avec les différentes exigences système.
L'idée étant que la complexité des systèmes à concevoir mettra à terme la tâche d'énumération et d'évaluation des architectures candidates hors de portée des experts s'ils ne bénéficient pas d'une assistance informatique.

Pour développer un outillage de conception capable de synthétiser des architectures admissibles des chaînes de traitement logicielles ont été proposées dans des travaux précurseurs [4]. Elles comportent un générateur de contraintes dédié qui prend en entrée des données système et dont la sortie est exploitée par un outil de résolution. Dans cette approche l'essentiel du développement porte sur le générateur ; l'outil de résolution est pris sur étagère et il n'y a pas de langage de modélisation en entrée. D'autres travaux s'appuient sur l'utilisation ou l'enrichissement d'un langage de modélisation de système proposant différentes descriptions (exigences, structure, comportement etc.) et l'exploite à l'aide d'outils, d'algorithmes de vérification ou de résolution [3, 5] pour énumérer ou évaluer des architectures candidates.

Plus récemment des travaux portent sur le développement d'un langage de modélisation de problème de conception en vue de leur résolution [6]. Il s'agit donc d'enrichir les langages de résolution pour aborder les problèmes de synthèse. Dans ce cadre, les travaux que nous présentons montrent l'intérêt d'utiliser le langage DEPS pour d'une part modéliser un système et des exigences qui lui sont associées et d'autre part résoudre pour trouver une solution de synthèse.

L'étude de cas, objet de cette évaluation, est une problématique de déploiement de fonctions avion sur une architecture informatique embarquée de type avionique modulaire intégrée. Nous nous sommes limités dans ce papier à la fois en types d'éléments d'architecture (les calculateurs) et en types d'exigences (la sûreté de fonctionnement) mais nous avons veillé attentivement au caractère généralisable de l'approche.

Le papier s'organise comme suit : nous positionnons d'abord nos travaux dans le contexte de l'avionique modulaire, de la sûreté de fonctionnement et de la répartition des rôles entre acteurs puis nous présentons le langage DEPS et l'outillage associé que nous utilisons. Ensuite nous décrirons successivement l'étude de cas et le problème posé. Puis nous discuterons de la modélisation qui a été faite en DEPS et des résultats que nous avons obtenus. Enfin nous présenterons des perspectives d'évolution de ce travail tant du point de vue de la généralisation de l'étude de cas que des évolutions du formalisme DEPS.

2 PROBLEMATIQUE

2.1 Cadre général

Le concept d'avionique modulaire intégrée (IMA) [7, 8], défini dans les années 1990, s'est aujourd'hui imposé comme l'une des références pour le développement de plateformes avioniques civiles. Permis par l'émergence de technologies augmentant la puissance des calculateurs, ce concept vise à regrouper plusieurs fonctions logicielles non vitales, auparavant exécutées par des calculateurs dédiés, sur un même calculateur.

Un des apports de l'IMA est de découpler le développement logiciel du matériel sous-jacent, i.e. assurer une modularité de ces logiciels qui peuvent ainsi être déployés indifféremment sur les calculateurs en fonction de leurs besoins en ressources, pourvu que ces derniers soient « compatibles » IMA.

Cette compatibilité se traduit par un ensemble d'interfaces applicatives (API) générique à tous les calculateurs et une allocation statique des ressources temporelles et d'I/O.
En IMA le concept de partition traduit cette allocation des ressources (cf. Figure 1). Un logiciel, réalisant tout ou partie d'une fonction avion, peut ainsi être projeté sur une ou plusieurs partitions suivant le besoin en ressources et les exigences associées.

En contrepartie, cette capacité de mutualisation des logiciels associée à l'accroissement de leur nombre a entrainé une augmentation de la complexité du problème d'intégration. Il s'agit de trouver un schéma d'allocation des calculateurs aux logiciels répondant aux besoins en termes de ressources tout en respectant les contraintes de latence d'exécution des chaines fonctionnelles traversantes (i.e. traitées via l'exécution de plusieurs logiciels) ou encore de sûreté de fonctionnement. Du fait de ces contraintes multiples et du nombre important de déploiements, trouver manuellement une allocation satisfaisante devient très difficile.

L'intégrateur système va utiliser sa connaissance des logiciels et des ressources calculateurs et réseau pour proposer plusieurs allocations puis évaluer à l'aide d'outils la conformité de ces allocations aux exigences jusqu'à trouver un déploiement satisfaisant. Il s'agit d'une démarche d'analyse a posteriori.

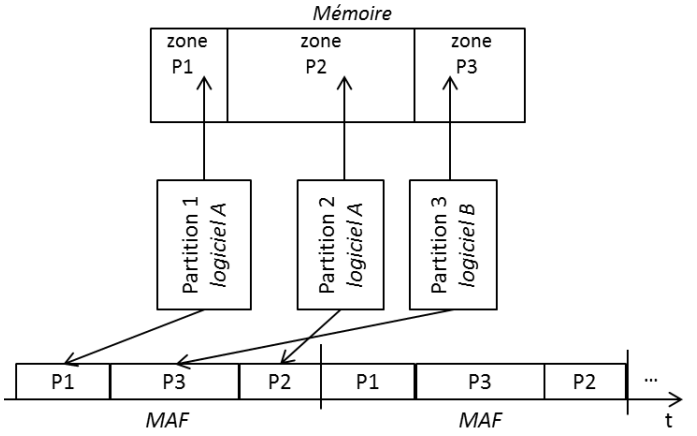


Figure 1 : principe d'allocation statique des ressources via le concept de partition

Partant de ce constat, nous avons développé une approche de déploiement des logiciels qui prend en compte a priori et non a posteriori les exigences qui portent sur les fonctions avion.

Afin d'illustrer notre démarche, nous nous placerons par la suite dans le cas d'un problème de déploiement de logiciels sur un ensemble de calculateurs IMA devant respecter uniquement des exigences de sûreté de fonctionnement. Ces exigences sont parmi les plus importantes, tout avion civil étant pensé dans une approche visant à réduire au maximum les risques d'accident et donc potentiellement de pertes humaines.

2.2 Les exigences de sûreté de fonctionnement

2.2.1 Production des exigences

La sûreté de fonctionnement vise à établir des niveaux admissibles de fiabilité, disponibilité et maintenabilité des systèmes essentiels de l'avion afin de garantir la sécurité de l'appareil en vol comme au sol. Les exigences permettant d'atteindre ces niveaux résultent de l'analyse des cas de pannes et défaillances, pouvant amener par exemple à une perte de fonctionnalité ou une corruption de données.

En terme d'architecture plateforme, ces analyses conduisent à un ensemble de « patterns ». Un « pattern » est une proposition de découpage d'une fonction avion en composants logiciels avec des exigences associées de ségrégation ou de dissimilarité. Par exemple, la robustesse à un cas de panne d'une fonction conduit à un « pattern » de type redondance du logiciel exécutant cette fonction et son déploiement sur deux calculateurs distincts. On parlera de ségrégation des ressources utilisées. Ces exigences ne sont pas uniquement applicables au niveau logiciel, mais peuvent porter par exemple sur l'ensemble de la chaine de traitement d'une fonction avion donnée, en exigeant que cette fonction soit réalisée via deux chaines ségréguées. Dans ce cas, l'ensemble des logiciels réalisant la première chaine devront impérativement être déployés sur des calculateurs distincts des logiciels réalisant la seconde chaine. Une illustration de « patterns » et d'exigences associées sera donnée dans la partie cas d'étude.

2.2.2 Utilisation à l'intégration

Afin de prendre en compte au plus tôt ces exigences et patterns pour l'identification de schémas d'intégration satisfaisants, il est nécessaire de pouvoir formaliser ces éléments de sûreté de fonctionnement à partir des sorties « manuscrites » livrées après analyse amont. Pour que cette formalisation ne soit pas seulement exploitable par des spécialistes du domaine de la sûreté de fonctionnement mais également par l'intégrateur plateforme, il est nécessaire que cette formalisation se situe au bon niveau entre pouvoir d'expression et abstraction. Notre objectif est ensuite d'utiliser ces éléments afin de déterminer au plus tôt et de manière informatisée, les possibilités de déploiements candidats (cf.
Figure 2).

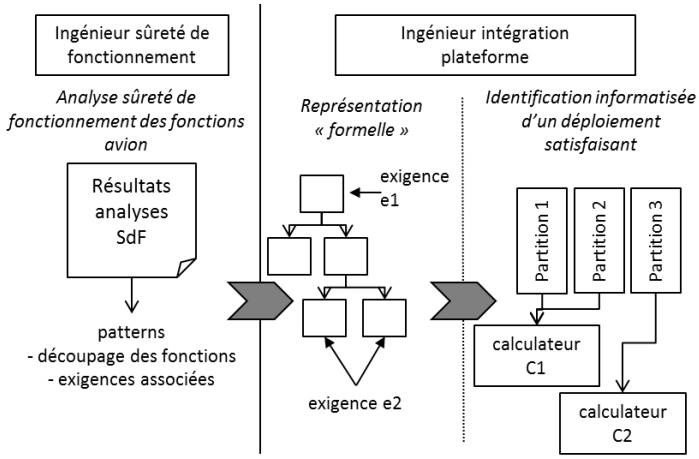


Figure 2 : principe de prise en compte au plus tôt des exigences de Sûreté de Fonctionnement

Nous sommes donc confrontés à un problème de représentation « formelle » des éléments d'entrée, ainsi qu'à un problème d'identification de solutions compatibles des exigences. Nous nous positionnons ainsi dans une démarche de modélisation et de synthèse, par opposition aux démarches actuelles centrée sur l'analyse.
Nous avons fait le choix d'utiliser un même formalisme pour traiter conjointement la question de la représentation des éléments système et celle de la recherche de solutions compatibles de l'espace des exigences. Il s'agit du langage DEPS.

3 LE LANGAGE DEPS

3.1 Paradigme

Le langage DEPS (Design Problem Specification) est ce qu'on appelle communément un langage dédié ou DSL (Domain Specific Language). Le domaine d'application visé est la spécification et la résolution de problèmes de l'ingénieur, en particulier ceux que l'on rencontre en conception de produits ou de systèmes.
DEPS est un langage dédié externe (par opposition à interne ou embarqué). Le code source est donc indépendant de tout langage généraliste hôte.

Le langage DEPS peut être vu comme une combinaison entre un langage de modélisation logiciel ou système et un langage de programmation mathématique. Aux premiers ont été empruntés les traits de structuration et d'abstraction qui permettent de représenter les éléments et le système étudié. Aux seconds ont été empruntés les concepts mathématiques nécessaires à la résolution des problèmes de l'ingénieur : inconnues, équations et inéquations.

Cette combinaison permet à la fois de représenter les problèmes de conception et poser puis résoudre ou optimiser les systèmes d'équations et d'inéquations qui les régissent [6].

3.2 Caractéristiques essentielles

3.2.1 Le Modèle

Le trait fondamental du langage est le Modèle. Tout Modèle est défini à l'aide du mot clé *Model* suivi de son nom et de sa liste d'arguments (éventuellement vide). Il comporte dans l'ordre : une zone de déclaration-définition des constantes du modèle, une zone de déclaration des variables, une zone de déclaration-création des éléments et une zone de définition des propriétés. La définition d'un Modèle DEPS se termine par le mot clé *End.*

Les propriétés d'un Modèle sont les équations et les inéquations qui portent sur les constantes et les variables de ce Modèle. Un Modèle contient donc tous les ingrédients nécessaires à la pose des contraintes qui régissent une instance de ce Modèle.

Comme dans un langage objet on dispose de l'héritage et de la composition : un Modèle peut être étendu et hériter des propriétés d'un autre Modèle et il peut être composé d'éléments, instances d'autres Modèles.

Des éléments peuvent être passés en argument d'un Modèle créant un lien d'agrégation avec celui-ci.
Des constantes peuvent aussi être passées en argument d'un Modèle ce qui permet de créer des Modèles paramétrés.

Modéliser un problème revient donc à spécifier des Modèles.

3.2.2 La Quantité

En DEPS, les constantes comme les variables sont associées à des types de grandeurs physiques ou technologiques qu'on appelle quantités (*Quantity*). Elles sont nécessaires en Ingénierie de Systèmes.

Une quantité possède :
- Un type de quantité de base (*QuantityKind*). Par exemple, réel (*Real*), entier (*Integer*), longueur (*Length*) ;
- Une borne min (resp. max) qui représente la valeur minimale (resp. maximale) pouvant être prise par toute constante ou variable ayant pour type la quantité définie ;
- Une dimension qui représente la dimension au sens de l'analyse dimensionnelle de la quantité. Par exemple [L] pour une longueur ou [U] pour une grandeur sans dimension ;
- Une unité de la quantité. Par exemple le mètre m pour une longueur.

4 L'OUTILLAGE EXISTANT

4.1 L'environnement de développement

L'environnement de modélisation et de résolution intégré associé au langage DEPS comprend :

- Des fonctions d'édition de modèle,
- Des fonctions de gestion de projet basées sur un mécanisme de packages ;
- Un compilateur,
- Un solveur.

Une approche par intégration plutôt qu'une approche par transformation de modèles a été privilégiée, En effet, dans le cas de la résolution d'un problème de synthèse de système sous-défini, il va être nécessaire en cas de résultat de calcul non satisfaisant de réaliser une mise au point des modèles dans le langage DEPS. En optant délibérément pour une approche par intégration, nous favorisons ce processus de mise au point.

4.2 La résolution

Les méthodes de calcul que nous utilisons sont tirées des travaux sur la résolution des CSP.

Un CSP (Constraint Satisfaction Problem) est défini par un triplet (*X, D, C*) tel que [9] :
- *X* est un ensemble fini de variables dites variables contraintes.
- *D* est un ensemble fini de domaines de ces variables.
- *C* est un ensemble fini de contraintes sur les variables de l'ensemble *X*.

Les domaines des variables peuvent être discrets (CSP) ou continus (NCSP pour Numerical CSP). On appelle contrainte, n'importe quel type de relation mathématique qui porte sur les valeurs d'un ensemble de variables : égalité, différence, inégalité logique et/ou algébrique (linéaire ou non linéaire).

Résoudre un CSP revient ainsi à instancier chacune des variables de X tout en satisfaisant l'ensemble C des contraintes du problème. Partant des domaines de valeurs de chacune des variables du problème, l'algorithme de résolution alterne contraction des domaines et choix d'un sous domaine pour une variable du problème jusqu'à aboutir à une solution ou bien un échec. Ce dernier est alors traité par un retour en arrière sur les points de choix précédents.

L'étape de contraction est réalisée à l'aide d'algorithmes dédiés qui mettent à profit les contraintes explicitées du problème pour réduire les domaines de chaque variable.
Dans le cas d'un problème sur-contraint, un échec peut apparaitre dès la première propagation ou bien au final après avoir exploré les parties restantes de l'arbre de recherche. Dans ce cas, la méthode garantit qu'il n'y a pas de solution au problème posé.

Le solveur implémente une méthode de propagation de type HC4 révisé [10] sur des équations et inéquations portant sur quatre types de domaines : les intervalles ouverts de réels, les intervalles d'entiers, les ensembles énumérés de valeurs flottantes et les ensembles énumérés de valeurs entières signées. Les contractions sont réalisées directement sur les domaines typés sans repasser dans les intervalles de réels. L'algorithme de recherche de solution est une méthode de branch and prune. Les stratégies round-robin et first-fail sont disponibles.

Dans le cas d'un problème sur-contraint, un échec peut apparaitre dès la première propagation ou bien au final après avoir exploré les parties restantes de l'arbre de recherche. Dans ce cas, l'échec s'interprète comme la preuve qu'il n'y a pas de solution au problème posé et non pas comme une défaillance de l'algorithme de résolution.

L'architecture orientée-objet du solveur a été pensée de manière à pouvoir être étendue à d'autres méthodes de propagation et/ou de résolution (box-consistance, méthodes locales, …).

5 APPLICATION AU CAS D'ETUDE

5.1 Description du problème

Notre cas d'étude (cf.

Figure **3**) consiste en un problème de déploiement de sept fonctions avion (notamment des applications de freinage, de remontée de pannes, de communication, etc.) sur une plateforme composée de quatre calculateurs, à déterminer en fonction des exigences de sûreté de fonctionnement associées.

Pour rappel, nous nous plaçons dans le rôle de l'intégrateur plateforme et supposons l'analyse de sûreté de fonctionnement des sept fonctions avion terminée.

Nous avons donc en entrée pour chaque fonction un document décrivant le découpage en composants logiciels puis la projection en termes de partitions, avec des exigences de ségrégation matérielle (projection des composants ségrégués sur des calculateurs distincts) ou inversement de co-localisation pour assurer la cohérence de données à traiter séquentiellement dans un temps maitrisé court. Nous allons détailler ci-après la décomposition du cas d'étude (cf.
Figure 3) sur deux fonctions avion.

La première fonction consiste en une fonction de gestion du système d'atterrissage. Cette fonction dite SAT est initialement projetée sur un seul composant logiciel.

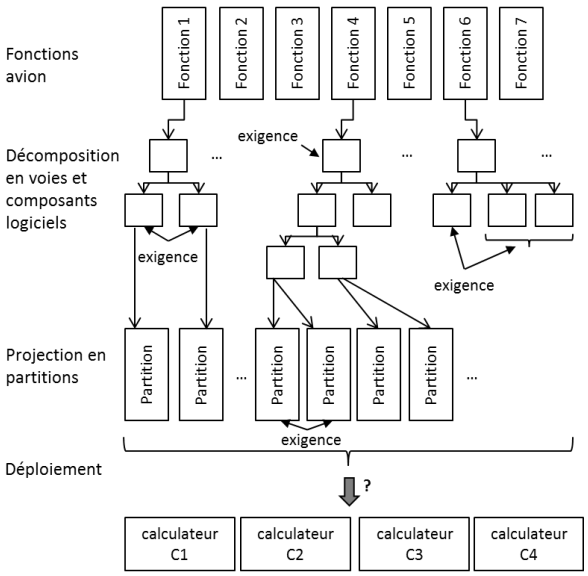
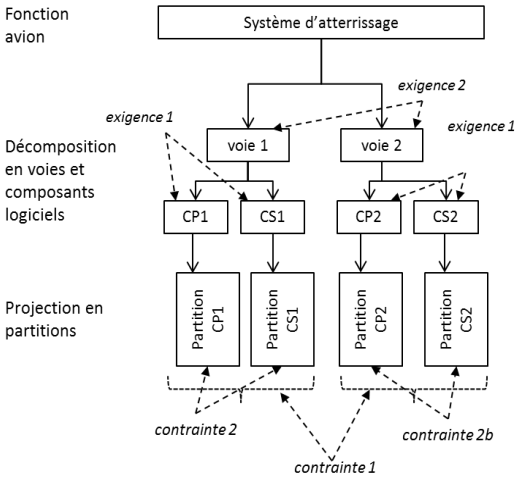


Figure 3 : description générale du cas d'étude

L'analyse de sûreté de fonctionnement nous impose deux exigences :
- Exigence 1 : un second composant logiciel dissimilaire (appelée CS pour composant « safety ») doit être prévu, afin de prendre le relais en cas de perte du composant principal (appelé CP). Ces composants doivent être ségrégués matériellement afin qu'une panne matérielle affectant CP n'affecte pas CS ;
- Exigence 2 : cette chaine de traitement (ou voie) CP + CS doit être redondée et ségréguée matériellement afin d'être robuste à la perte de la première chaine.



.

Figure 4 : décomposition de la fonction SAT

D'un point de vue traitement, le fournisseur de la fonction nous indique que chaque composant peut être projeté sur une seule partition. Nous nous retrouvons donc avec un schéma de décomposition résultant sur quatre partitions, avec les contraintes suivantes (cf. Figure 4) :
- Contrainte 1 : les partitions {CS1, CP1} doivent être matériellement ségréguées des partitions {CS2, CP2} (cf. Exigence 1);
- Contrainte 2 : les partitions CS1 et CP1 doivent être matériellement ségréguées (cf. Exigence 2) ;
- Contrainte 2bis : les partitions CS2 et CP2 doivent être matériellement ségréguées (cf. Exigence 2).
La seconde fonction que nous détaillons consiste en une fonction de gestion des remontées de pannes (SRP).
Cette fonction est initialement projetée sur un seul composant logiciel CGP (composant de gestion pannes).

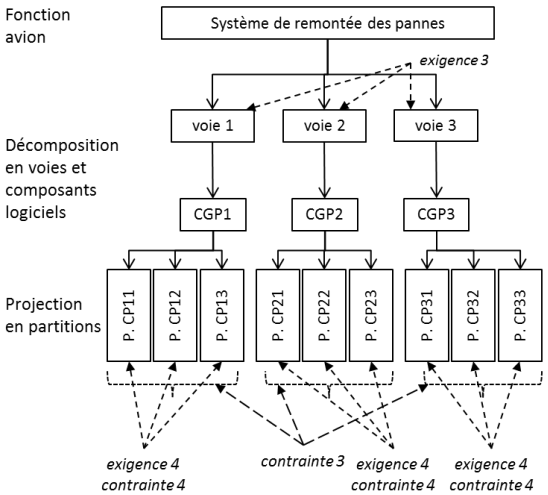


Figure 5 : décomposition de la fonction SRP

L'analyse de sûreté de fonctionnement nous impose deux exigences :
- Exigence 3 : afin d'offrir une disponibilité suffisante, le composant doit être tripliqué, chaque occurrence devant être matériellement ségréguée des autres ;

- Exigence 4 : chaque composant sera découpé suivant trois traitements projetés sur trois partitions différentes qui doivent, pour des raisons de temps, de séquencement et de rapidité des traitements, être co-localisées sur un même calculateur. Nous nous retrouvons avec un schéma de décomposition amenant à 9 partitions avec les contraintes suivantes (cf.

Figure 5):

- Contrainte 3 : les triplets de partitions {PCGP11, PCGP12, PCGP13}, {PCGP21, PCGP22, PCGP23} et {PCGP31, PCGP32, PCGP33} doivent être matériellement ségréguées;
- Contrainte 4 : toutes les partitions d'un même triplet doivent être projetées sur un même calculateur.

5.2 Formalisation en DEPS

5.2.1 Les éléments du système

Nous avons modélisé les éléments constitutifs d'une fonction avion aux différents niveaux d'abstraction nécessaires à l'expression des exigences de sûreté de fonctionnement.

```
Package Partition ;
Uses Universal;
Quantity CpuIndex          Model Partition()
 Kind : Integer ;          Constants
 Min  : 1 ;                Variables
 Max : 4 ;                 icpu : CpuIndex;
 Unit : u ;                Elements
End                        Properties
                           End
```

Figure 6 : Model DEPS d'une partition

Ce sont les modèles de : fonction avion, chaîne de traitement, composant logiciel, et partition (cf.
Figure 3).

- Une fonction avion est composée d'une ou plusieurs voies. Les voies sont des éléments du modèle de la fonction avion et sont donc des instances de chaîne de traitement.
- Une chaîne de traitement est composée d'un ou plusieurs composants logiciels ainsi que d'autres éléments (capteurs, actionneurs, réseau ...) hors du champ d'étude de cet article.
- Un composant logiciel est découpé en une ou plusieurs partitions.
- Une partition doit être projetée sur une ressource de calcul pour pouvoir s'exécuter (cf. Figure 6).

5.2.2 Les exigences

Le modèle qui suit montre comment on modélise au niveau des composants logiciels et des partitions l'exigence de redondance et de ségrégation de la chaine de traitement du système d'atterrissage (cf. Figure 7).
Les exigences sont propres à chaque fonction avion et elles portent à différents niveaux de décomposition de la fonction considérée.

```
Model NonColoc (P1, P2)        Model Segregation(C1, C2)
Constants                      Constants
Variables                      Variables
Elements                       Elements
 P1 : Partition ;               C1 : Compl ; (* composant
 P2 : Partition ;                              logiciel 1 partition*)
Properties                     C2 : Comp1 ;
  P1.icpu - P2.icpu <> 1;      NCL : NonColoc(C1.P, C2.P) ;
End                            Properties
                               End
Model ChaineTraitementSA ()
Constants
Variables
Elements
 CP : Comp1() ;  (* composant principal *)
 CS : Comp1() ;  (* composant safety     *)
 SEG : Segregation (CP, CS) ;
Properties
End
```

Figure 7 : Modèle d'une chaîne de traitement du système d'atterrissage

En procédant ainsi on a pu modéliser l'exigence d'indépendance matérielle entre les deux voies dupliquées de la chaine de traitement du système d'atterrissage tout comme l'exigence de co-localisation sur un même calculateur des partitions des composant logiciels du système de remontée de pannes.

5.3 Résultats

Après modélisation des sept fonctions avions et de leurs exigences respectives on a obtenu les solutions de déploiement des partitions de ces fonctions sur les calculateurs. Les méthodes de résolution de DEPS retrouvent bien dans toutes les solutions que, conformément aux exigences, la fonction SAT a besoin de quatre calculateurs pour son déploiement tandis que la fonction SRP en a besoin de trois (cf. Figure 8).

.

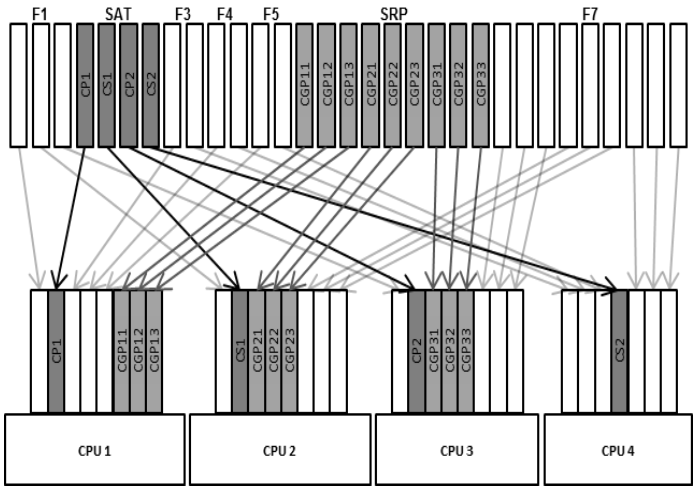


Figure 8 : Un déploiement des sept fonctions avion

6 CONCLUSIONS ET PERSPECTIVES

Dans cet article nous avons montré qu'il est possible de capturer des exigences de sûreté de fonctionnement difficiles à formaliser directement dans le formalisme (V, D, C) des CSP.

Nous avons utilisé le langage DEPS qui nous a permis de modéliser les bonnes abstractions pour d'une part décrire un modèle sous-défini d'architecture IMA et d'autre part des modèles d'exigences qui portent sur les bonnes abstractions.

Le problème de déploiement posé a été résolu en appliquant des méthodes de satisfaction de contraintes aux propriétés du problème.

Les traits de structuration offerts par le langage facilitent la réutilisation des modèles « métier » développés.

Les travaux en cours portent sur l'expression d'autres exigences d'architecture et sur leur prise en compte conjointe pour trouver des solutions de déploiement admissibles sur des problèmes à l'échelle.

REFERENCES

[1] Becz, S., Pinto, A., Zeidner, L. E., Banaszuk, A., Khire, R., and Reeve, H. M.,Design System for Managing Complexity in Aerospace Systems," 13th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Texas, 2010

[2] Pinto A., Becz S., Reeve H.M., Correct-by-construction design of aircraft electric power systems, in AIAA Aviation Technology, Integration, and Operations Conf., 2010.

[3] Zeidner L, Reeve H, Khire R, Becz S., Architectural Enumeration and Evaluation for Identification of Low-Complexity Systems, MATIO10, Texas, 2010.

[4] Bieber P., J.P Bodeveix J.P., Castel C., Doose D., Filali M., Minot F., Pralet C., Constraint-based Design of Avionics Platform - Preliminary Design Exploration ERTS2008 Toulouse January 2008.

[5] Albarello, N., Welcomme J.B. and Reyterou C., A formal design synthesis and optimization for systems architectures, MOSIM'12, Bordeaux, France, 2012.

[6] Yvars P.A., Zimmer L., DEPS un langage pour la spécification de problèmes de conception de systèmes, MOSIM'14, Nancy, 2014.

[7] ARINC Specifications 653-1 Avionics Application Software Standard Interface, SAE ITC, 2017

[8] DO-297 Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations. RTCA, 2017.

[9] E. Tsang, Foundations of Constraint Satisfaction. London and San Diego: Academic Press, 1993.

[10] Benhamou F., Goualard F., Granvilliers L., Puget J.F., Revising Hull and Box consistency,16th International Conference on Logic Programming, 1993.