Automatisation de la synthèse d'architectures appliquée aux aéronefs à voilure tournante

Chris Hartmann

Thèse de Doctorat

Chris HARTMANN

Mémoire présenté en vue de l'obtention
du grade de Docteur de l'Ecole Centrale de Nantes
Sous le label de l'UNIVERSITÉ BRETAGNE LOIRE

École doctorale : Sciences Pour l'Ingénieur

Spécialité : Productique - Mécanique
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes

Soutenue le 30 Janvier 2018

Automatisation de la synthèse d'architectures appliquée aux aéronefs à voilure tournante

JURY

Président :	**Rivest Louis**, Professeur, École de technologie supérieure de Montréal
Rapporteurs :	**Jankovic Marija**, Maître de Conférences HDR, Centrale Supelec **Pailhès Jérôme**, Professeur des Universités, Art et Métiers ParisTech
Examinateurs:	**Aldanondo Michel**, Professeur, IMT - Mines Albi **Brissaud Daniel**, Professeur des Universités, INP Grenoble **Mermoz Emmanuel**, Docteur, HDR, Airbus Helicopters
Directeur de thèse :	**Bernard Alain**, Professeur des Universités, Ecole Centrale de Nantes
Co-encadrant de thèse :	**Chenouard Raphaël**, Maître de Conférences, Ecole Centrale de Nantes

UNIVERSITE BRETAGNE LOIRE

CENTRALE NANTES

Thèse de Doctorat

Chris Hartmann

Automatisation de la synthèse d'architectures appliquée aux aéronefs à voilure tournante

Automated architecture synthesis applied to rotary-wing aircrafts

Résumé

Les travaux, présentés dans ce manuscrit de thèse, s'inscrivent dans les courants de l'Ingénierie Système et de la synthèse assistée par ordinateur. Une méthodologie outillée à l'aide d'un logiciel a été développée et est détaillée.
Le processus de synthèse semi-automatisé est organisé en trois grandes phases : l'extraction du besoin et sa transformation en spécification du système à concevoir, une synthèse des architectures logiques et une analyse des architectures physiques.
L'extraction et la transformation du besoin est une étape manuelle dans la méthodologie proposée. Elle s'appuie grandement sur des travaux précédents du champ de l'Ingénierie Système. L'objectif de ce sous-processus est d'obtenir une représentation du système compréhensible par l'utilisateur et interprétable par le logiciel. Les parties prenantes, les situations de vie que le système va rencontrer, les besoins, les exigences et les interfaces avec l'environnement sont modélisés.
La synthèse, ou génération, des architectures logiques, est le résultat de la modélisation précédente du système. Un code C++ permet la transformation du problème de synthèse en expressions mathématiques qui sont résolues à l'aide d'un solveur CSP entier. Le résultat de ce sous-processus est un ensemble de graphes, triés par famille. Ces graphes représentent toutes les architectures logiques viables vis-à-vis des connexions entre ses sous-systèmes.
L'analyse des architectures physiques permet d'écrire, pour chaque architecture logique, un système d'équations physiques non-linéaires mais non-différentielles pour une première étape de pré-dimensionnement. Ces systèmes, écrits sous la forme de problèmes d'optimisation sont ensuite résolus à l'aide d'un solveur CSP réel.
Au final, les architectures sont triées suivant la valeur d'une variable d'état commune à toutes les alternatives.

Mots clés
Ingénierie Système, Satisfaction de Problèmes sous Contraintes, Architecture, Synthèse

Abstract

The research work presented in this thesis is related to the System Engineering field and the computer aided synthesis field. A methodology realized by a new software is developed and introduced.
The synthesis process is semi-automated and is devided into three phases: the need extraction and its translation into system requirements, a logical architecture synthesis and a physical architecture analysis.
The need extraction and its translation into system requirements are highly inspired from previous work from the System Engineering field. Nevertheless, the objective, at this step, is to provide the software and the user with a unique model understandable to both. Stakeholders, life situations, needs, requirements and interfaces with the environment are modelized.
The logical architecture synthesis, or logical architecture generation, is in accordance with the models we build previoulsy. That is to say that all logical architectures are instantiations of the system requirements expressed before. A C++ code translates this model into mathematical equations solved by an integer CSP solver. The result of this part of the methodology is a set of graphs, ranked by family. These graphs are views of the logical architectures. They express all the possible links between the sub-systems of the architectures.
The physical architecture analysis step is an automated equation writer. The equations are non-linear and non-differential and they are written for each logical architecture generated at the previous step. They are used for a first step of pre-sizing. These systems are then solved by a CSP solver for real numbers through an optimization. At the end, all the feasible architectures are ranked according to a unique state variable that is common to all possible solutions.

Key Words
System Engineering, Constraint Satisfaction Problems, Architecture, Synthesis

Remerciements

Après trois années de travail sur le sujet de thèse qui va être présenté dans ce document, je tiens à remercier toutes les personnes qui ont apporté leur contribution à ce projet.
Deux personnes étaient à l'origine du sujet, l'industriel E. Mermoz, titulaire d'une HDR et le Professeur A. Bernard. Sans eux rien de tout cela n'aurait été possible. Je les remercie donc tous les deux pour avoir imaginé cette problématique et pour m'avoir fait confiance pour essayer d'y répondre. Je remercie le Professeur A. Bernard pour m'avoir poussé à rentrer dans le monde académique malgré le caractère industriel de ma thèse. Cela m'a permis de rencontrer la communauté Française de l'Ingénierie Système. J'exprime ma reconnaissance au Senior Expert E. Mermoz pour avoir commencé mon initiation au monde des aéronefs à décollage vertical et pour avoir contribué à ce que cette initiation ne s'arrête pas à la fin de ma thèse. Par ailleurs, le Maître de Conférences R. Chenouard s'est joint à l'équipe encadrante et a apporté ses connaissances scientifiques dans des domaines qui m'étaient inconnus. Je le remercie chaleureusement pour les savoirs qu'il m'a transmis, pour ses conseils ainsi que pour certains bouts de code qui m'ont été bien utiles tout au long des travaux.
Je remercie les rapporteurs et membres du jury qui ont donné de leur temps pour prendre connaissance de mes travaux et être en mesure de les discuter lors de la soutenance. A cette occasion je les remercie pour la relecture ainsi que mon équipe encadrante, mon père P. Hartmann et E. Pariset. Sans eux et leurs relectures, le manuscrit n'aurait pas eu la même qualité.
J'ai une pensée pour les quelques doctorants du laboratoire que j'ai pu croiser. Même si nos contacts ont été très espacés, je repars avec quelques bons souvenirs dont le pot à la fin de ma journée de soutenance.
Je salue maintenant mes collègues au sein de mon entreprise que ce soit les plus éloignés, du monde de l'hydraulique, les experts, l'équipe de managers, les motoristes, la mécanique qui devient ma nouvelle maison mais surtout le service qui m'a accueilli il y a trois ans. J'ai une pensée pour les ingénieurs qui étaient là avant moi, ceux qui sont arrivés en même temps que moi et ceux que j'ai vu nous rejoindre. De même je remercie les alternants ingénieurs ou de DUT ainsi que les stagiaires et leur état d'esprit plus festif que le mien au départ de cette aventure. Enfin pour terminer ce petit paragraphe je passe le flambeau aux nouveaux doctorants qui m'ont accompagné durant la fin de ma thèse. Je vous remercie pour votre soutien et je serai là à mon tour si vous avez besoin de moi.
En arrivant dans cette région du sud de la France je faisais la différence entre amis et collègues. Puis, j'ai appris que beaucoup de natifs utilisent le mot « collègues » pour parler de leurs amis. Je ne le pensais pas forcément à mon arrivée, mais je me rends compte que certains de mes collègues de travail sont devenus de vrais « collègues ». J'espère donc que nous aurons encore l'occasion de vivre des bons moments professionnels et amicaux dans les années à venir.
Enfin, et pour terminer par le plus important, je remercie ma famille proche ou éloignée, de cœur ou de sang, qui m'a soutenu durant toute ma scolarité et qui, je le sais, continuera

à me soutenir dans le futur. Je ne vais pas citer tout le monde mais je me dois de laisser un mot pour mes parents et mon frère qui m'ont toujours accompagné, au moins par la pensée et le téléphone. Merci à vous trois pour votre soutien sans faille, votre fierté, votre impact sur ma vie personnelle et professionnelle. Vous êtes le point d'ancrage de ma vie. Je finirai par un message à mon frère Thibaut : tu es le prochain sur la liste des docteurs de la famille Hartmann et cela me rend d'avance très fier. Courage pour les années qu'il te reste, tu verras que le travail n'est pas si désagréable, sinon nous serions tous bien fous de nous lancer dans des thèses.

Table des matières

Liste des figures

Liste des tableaux

Liste des codes

Liste des acronymes

AFIS Association Française pour l'Ingénierie Système. 17, 27, 28, 40

aip primeca Atelier Inter-Etablissements de Productique et Pôle de Ressources Informatiques pour la MECAnique. 17, 27, 40

AO Objet Architectural, ou *Architectural Object*. 3–6, 8, 15, 18, 20–23, 25, 30, 31, 41, 48, 50–53, 65–68, 72, 73, 76, 78–89, 92–104, 106, 110, 112, 114–116, 118–129

BPMN *Business Process Model and Notation*. 47

BTP Boîte de Transmission Principale. 96, 106, 115, 118–120

CBR Case Based Reasoning. 20

CFRL Langage de Représentation Fonctionnelle Causal, ou *Causal Functional Representation Language*. 18

CNRS Centre National de la Recherche Scientifique. 18

CPD Description de Processus Causal, ou *Causal Process Description*. 18

CSP problème de satisfaction de contraintes, ou *Constraints Satisfaction Problem*. 31, 35, 41, 79, 82, 83, 86–89, 92, 100–102, 110, 112, 119, 121, 125, 130

DSM Matrice d'interconnexion, ou *Design Structure Matrix*. 6, 15, 25, 36, 39, 79, 81–85, 87–89, 93, 94, 120, 123, 124

FAST Functional Analysis Systems Technique. 19, 40, 49

FBS (Gero) Function Behaviour Structure. 20, 34

FBS (Umeda) Function Behaviour State. 20, 25, 37, 38

INCOSE Conseil International sur l'Ingénierie Système, ou *International Council on Systems Engineering*. 17, 40

M2M Modèle à Modèle, ou *Model to Model*. 16, 39

M2T Modèle à Texte, ou *Model to Text*. 16

MBSE Ingénierie des Modèles, ou *Model Based System Engineering*. 16, 28

MIMO Entrées multiples Sorties Multiples, ou *Multiple Inputs Multiple Outputs*. 6, 24, 92, 93, 110

OCL Langage de Contraintes Objet, ou *Object Constraint Language*. 24, 26, 112

OMG *Object Management Group*. 17, 39

QPAS *Qualitative Process Abduction System*. 37, 38

REM Représentation Energétique Macroscopique. 16

RFBS Requirement Function Behaviour Structure. 20

SAT problème de SATisfaisabilité booléenne, ou *boolean SATisfability problem*. 36, 40

SEBoK Le guide de l'Ingénierie Système, ou *The Guide to the Systems Engineering Body of Knowledge*. 15

SISO Entrées uniques Sorties uniques, ou *Single Inputs Single Outputs*. 24

SysML Le Langage de Modélisation de Systèmes, ou *Systems Modeling Language*. 6, 17, 24, 26, 27, 39, 41, 47, 79, 94

TRIZ résolution des problèmes inventifs, ou *Teoriya Resheniya Izobretatelskikh Zadach*. 25

UML Langage de Modélisation Unifié, ou *Unified Modeling Language*. 4–6, 8, 17, 41, 112

Glossaire

architecture logique Une architecture logique permet l'étude des connexions entre les différents sous-systèmes composant le système à concevoir. Il s'agit d'un graphe multi-entrée, multi-sortie pouvant être représenté graphiquement à l'aide d'un langage comme SysML, ou mathématiquement par une DSM. Dans la méthodologie présentée au cours de ce manuscrit, les architectures logiques sont utiles dès la deuxième étape de la synthèse architecturale. 3, 5, 6, 18, 21, 22, 25, 26, 30, 41, 50, 76, 78, 79, 86, 88–90, 92–94, 100, 104, 106–110, 112, 115, 118–121, 123, 137

architecture physique Une architecture physique permet l'étude des variables comportementales et des variables d'état du système. 3, 8, 25, 41, 50, 51, 53, 78, 92, 112–114, 118–121, 124, 125, 130, 138

bond graph La méthodologie bond graph permet de construire des modèles de systèmes physiques dynamiques avec une approche énergétique et un langage de représentation graphique unique pour tous les domaines physiques. D'après l'équipe MOCIS du laboratoire LAGIS. 16, 18, 19, 23, 35, 41, 125

cardinalité La cardinalité d'un port correspond au nombre de fois que ce port est connecté dans une méta-architecture ou une architecture logique. La cardinalité est forcément comprise entre les multiplicités minimale et maximale du port. 82, 88–92, 100

flux Un flux représente un transfert de matière, d'énergie ou d'un signal entre deux ports appartenant à des objets différents. Les flux sont typés avec le même type que le port qui l'émet ou le reçoit. 4–6, 19, 20, 24, 31, 35, 37, 44, 47, 49–51, 53, 64, 65, 76, 78, 79, 81, 89, 95, 98, 101, 106, 114, 115, 118, 137

ingénierie système L'Ingénierie Système, ou *System Engineering* est une méthode interdisciplinaire structurant la définition et la réalisation d'un système à concevoir. Cette approche repose sur une vision systémique et permet la définition d'un produit à l'aide de plusieurs vues. 17, 27, 31, 39

isomorphisme Deux architectures logiques, pouvant être représentées sous forme de graphes, sont dites isomorphiques si les seules différences qu'elles ont sont l'inversion de deux objets architecturaux du même type. Par exemple, si une architecture logique contient deux objets architecturaux moteur à explosions, c'est-à-dire avec une occurrence de deux, l'occurrence une et l'occurrence deux peuvent être interchangées sans modifier le graphe. On dit que les deux graphes sont isomorphiques. 76, 92, 102, 104, 106

méta-architecture Les méta-architectures sont des architecture logiques simplifiées. En effet, elles traitent bien de l'interconnexion des objets architecturaux, mais ne tiennent pas compte des multiplicités sur les ports. Du fait que les multiplicités ne sont pas respectées, chaque objet architectural ne peut être présent qu'une seule

fois au maximum. Il s'agit donc de grandes familles d'architectures, mais pas d'architectures faisables. 3, 67, 76, 79, 80, 82, 86–89, 92, 101–105

méthodologie APTE Cette méthode permet l'analyse de la valeur et l'analyse fonctionnelle d'un produit, d'un procédé de fabrication ou encore d'une organisation, dans le but d'augmenter la qualité du produit fini et de minimiser les coûts en permettant une analyse systématique. 17, 47

Minizinc Minizinc est un langage conçu pour la spécification des problèmes d'optimisation et de décision sur des variables entières ou réelles. Un modèle Minizinc ne définit pas la façon dont le problème est résolu, même si le modèle peut contenir des annotations qui seront utilisées pour guider le solveur dans sa résolution. Minizinc est conçu pour être utilisé facilement avec différents solveurs. Cela est fait en transformant un modèle d'entrée Minizinc et un fichier de données en un fichier Flatzinc. Les modèles Flatzinc consistent en une déclaration de variables, de contraintes et d'une fonction objectif si le problème est un problème d'optimisation. La traduction des modèles Minizinc en modèles Flatzinc est différente suivant le solveur qui sera utilisé. 76, 86, 87, 101, 102

multiplicité La multiplicité sert à compter le nombre minimum et maximum de connexions admissibles pour un port. Par exemple, si un port à une multiplicité de [0;2], alors il peut être connecté une fois, deux fois ou pas connecté. 32, 35, 79, 86–89, 92, 95, 96, 100–102, 104, 105

occurrence Dans ce manuscrit, le terme occurrence est utilisé pour définir le nombre d'instances d'un objet architectural de la base de données. Si par exemple, une solution architecturale contient deux moteurs à explosions, on peut dire que l'objet architectural moteur à explosions a une occurrence de deux. 3, 51, 66, 67, 76, 86–88, 94, 101, 102, 106

port Un port est un objet pouvant recevoir ou donner un flux d'un type défini pouvant appartenir à trois familles : matière, énergie, signal. Un port est typé avec une direction qui peut être soit d'entrée soit de sortie. Un port appartient à un objet architectural. 5, 6, 15–17, 19–21, 23, 24, 26, 30–33, 35–37, 48, 50–53, 62, 65, 67, 68, 70, 71, 73, 76, 78, 79, 81–85, 87–89, 93–96, 98, 100, 101, 104, 105, 114, 118–121, 123–129, 137, 138

taxonomie Science des lois et des principes de la classification des organismes vivants; par extension, science de la classification. Selon le Centre National de Ressources Textuelles et Lexicales. 18

Chapitre 1

Introduction

1.1 Introduction du sujet de recherche, besoin industriel

Il est courant de lire dans des livres ou articles du domaine de la conception des systèmes que la phase amont du processus de conception est très importante, mais trop souvent pas assez développée. L'étude amont est fondamentale car une grande partie du coût global du projet est déterminée par les choix qui sont faits lors des phases d'émergence du besoin, de spécification du système et enfin de définition de l'architecture du système. Néanmoins, il peut être difficile d'analyser l'ensemble des variantes architecturales de par leur grand nombre et leurs potentielles grandes différences. C'est pour cette raison que la phase amont est souvent traitée trop rapidement et trop partiellement. L'objectif qui est fixé dans le cadre de ces travaux est d'aider les concepteurs en phase de pré-conception à parcourir l'ensemble des configurations architecturales et à les analyser de manière automatique, afin de permettre ensuite une phase de conception détaillée sur les concepts les plus prometteurs. Souvent les synthèses architecturales se font sur un concept unique et seuls les paramètres de conception sont explorés. C'est-à-dire que l'on recherche un optimal local et pas un optimum sur l'ensemble des solutions architecturales qui peuvent répondre au cahier des charges. La multitude des solutions, provenant d'une combinatoire sur les technologies disponibles pour instancier un système décrit par des exigences, est rarement analysée. Ceci étant dit, les recherches de solutions architecturales sont souvent menées d'après les expériences passées et sans recherche systématique : l'émergence de solutions innovantes est donc difficile. La recherche sur les architectures prometteuses doit être exhaustive sous certaines conditions de simplicité pour permettre la production d'un nombre fini de solutions pouvant contenir des innovations d'un point de vue architectural.

Pour traiter ce sujet de recherche, une convention industrielle de formation par la recherche (CIFRE) a été montée conjointement par Airbus Helicopters (AH) et le Laboratoire des Sciences du Numérique de Nantes (LS2N). L'objectif de ce partenariat était d'apporter à AH une réponse quant à la faisabilité de développer un outil pouvant aider la synthèse architecturale des ensembles dynamiques des hélicoptères. Les ensembles dynamiques d'hélicoptères sont composés des sous-systèmes permettant la propulsion et la sustentation de la machine à l'aide de voilures tournantes. Le LS2N a été choisi pour des travaux déjà réalisés sur la synthèse assistée par ordinateur. Ces travaux ont été menés, entre autre, par CHENOUARD et collab. [2008], CHRISTOPHE et collab. [2009] et MATAR [2013].
Le domaine d'application présenté en fil rouge dans ce manuscrit concerne les ensembles dynamiques d'aéronefs à voilures tournantes, mais la méthodologie proposée par cette thèse se veut générale et peut être appliquée à d'autres domaines d'application. C'est en analysant cette branche de l'aéronautique qu'est venue l'idée de proposer un logiciel automatisé qui serait en mesure de générer des solutions aussi diverses que celles qui ont été proposées depuis plus d'une centaine d'années, comme présenté, certainement partiellement, sur l'Annexe A.1. Il est légitime de penser que l'architecture actuelle des hélicoptères avec un rotor principal et un rotor anticouple sur la queue de la machine est l'architecture optimale étant donné que c'est elle qui s'est imposée de nos jours. Mais avec la réapparition de certaines architectures plus anciennes ou de nouvelles architectures de drones, le débat est réouvert. Au lieu de se focaliser directement sur des architectures particulières, il peut être intéressant d'étudier plusieurs solutions maximisant certains paramètres clés. C'est du moins le parti pris dans ce manuscrit et la raison de son existence.

Une méthodologie globale, de la définition du besoin à l'étude architecturale physique,

est présentée au cours de ce manuscrit. Elle est découpée en sous-processus mettant en jeu des modèles différents, mais se doit d'être cohérente dans son ensemble. La synthèse architecturale présentée ici est une synthèse assistée par ordinateur. Elle permet de générer et d'analyser des architectures logiques et des architectures physiques depuis une spécification textuelle et une base de données de modèles technologiques appelés Objet Architectural, ou *Architectural Object* (AO)s.

Le manuscrit s'articule autour de la méthodologie développée pour répondre à la problématique de recherche. Il est composé de six chapitres. Pour les chapitres présentant un apport scientifique, deux grandes parties seront développées. Une partie de présentation des travaux et une partie applicative.

Le premier chapitre est introductif, il permet de comprendre les objectifs initiaux qui ont conduits à la mise en place de la thèse dans une première partie. Dans une seconde partie, les concepts clés créés ou adaptés de la bibliographie lors des travaux de thèse sont présentés. Cela permet au lecteur d'avoir une première vue sur les objets et les processus qui seront décrits au court du manuscrit.

Dans le second chapitre, l'état de l'art est traité en suivant le même principe que ce manuscrit. Il s'appuie sur la méthodologie développée et les références bibliographiques sont triées par rapport à leur apport dans les sous-processus de la méthodologie de synthèse développée. Il s'agit donc d'une mise en perspective des travaux par rapport à l'état de l'art. Plusieurs questions de recherche sont énoncées. Des éléments de réponses seront donnés tout au long du manuscrit.

Le troisième chapitre correspond à la première étape de la méthodologie de synthèse. Il traite de la capture du besoin et de la transformation de ce besoin en exigences auxquelles le système à concevoir va devoir répondre. Les concepts clés de ce chapitre sont : besoin, exigence fonctionnelle, exigence de performance, exigence de contrainte, boîte noire et situation de vie.

Le quatrième chapitre correspond à la deuxième étape de la méthodologie de synthèse. Il traite de la transformation des exigences en architectures logiques pouvant être rangées en plusieurs concepts architecturaux aussi appelés méta-architectures. C'est à cette étape que le cœur de la synthèse est développé. Les concepts clés de ce chapitre sont : vue logique d'un AO, mode de fonctionnement, occurrence, architecture logique, méta-architecture et satisfaction de problèmes sous contraintes avec des variables entières.

Le cinquième chapitre correspond à la troisième étape de la méthodologie de synthèse. Il traite de l'analyse physique et donc du pré-dimensionnement des architectures logiques précédemment synthétisées. Les concepts clés de ce chapitre sont : vue physique d'un AO, architecture physique, satisfaction de problèmes sous contraintes avec des variables réelles et optimisation mono-objectif.

Le sixième chapitre est une conclusion qui présente l'implémentation de la méthodologie et qui récapitule les apports des travaux effectués et pointe les perspectives à venir.

1.2 Définition des concepts importants utilisés dans le manuscrit

Avant de se lancer à proprement dit dans la synthèse des différents courants bibliographiques qui se rapprochent de notre problématique, il est préférable de donner quelques définitions sur des termes clés du sujet qui nous importe.

FAISANDIER [2015b] apporte une spécification de la notion de système d'un point de vue structurel, qui correspond à notre besoin de définition. « Un système est caractérisé par un ensemble de constituants (matériels technologiques, logiciels, opérateurs humains, matériaux, procédures, services) ; les constituants sont en forte interaction, et échangent des flux de matière, d'énergie et d'information dans un environnement ou contexte donné. Cet ensemble satisfait des besoins, des attentes ; il accomplit une mission assortie d'objectifs prescrits permettant de répondre à une finalité.» Les systèmes complexes conçus de nos jours se meuvent dans différents environnements et sous plusieurs contextes tout au long de leur cycle de vie. Il est important de garder ce point en tête pour bien comprendre que chaque situation de vie peut conduire à un comportement souhaité différent.
Comme le fait remarquer FAISANDIER [2015b], un système de systèmes correspond à un simple système. La différence se fait au niveau de la granularité de ses constituants qui sont à présent des systèmes. Pour définir un système de systèmes, il faut néanmoins que chaque sous-système soit indépendant des autres.

CRAWLEY et collab. [2004] donnent une définition, traduite ci-dessous de l'anglais, de l'architecture d'un système que nous partageons. « L'architecture d'un système est une description abstraite de ses entités et de leurs interconnexions. [...]. L'architecture d'un système a une forte influence sur son comportement global. Chaque système a une architecture. Des architectures peuvent émerger, par exemple, du processus de conception d'un système, de l'évolution d'un ancien système à cause de fortes contraintes légales, pour obéir aux nouvelles régulations, aux standards et aux protocoles, par l'agrégation de plus petits sous-systèmes avec leur propre architecture ou par l'exploration des contraintes comportementales ou géométriques via un dialogue entre les clients et les architectes.» Cette définition est en accord avec les concepts utilisés dans ce manuscrit. En effet, dans notre processus de synthèse assistée, nous prenons en compte l'ajout d'exigences de contrainte qui correspondent à de la régulation ou à l'application de standards, ou encore à la réutilisation de concepts architecturaux existants. De même, le système à concevoir est spécifié principalement par le comportement global attendu.

Les définitions suivantes sont un apport du présent manuscrit. Elles sont issues du travail présenté tout au long des chapitres suivants mais sont énoncées ici car l'ensemble du manuscrit y fait référence et la plupart des sources bibliographiques sont introduites par rapport à leur apport vis-à-vis du travail de thèse. Elles sont accompagnées de méta-modèles issus des travaux présentés tout au long du manuscrit.

L'architecture, ou le système à concevoir, est le but à atteindre. Il est défini par ses interfaces extérieures et par sa composition interne. Un système peut être vu comme un AO et peut être utilisé comme un composant d'un système plus imposant. Lors de sa construction, ses attributs diffèrent légèrement de ceux d'un AO. L'architecture peut être vue sous plusieurs vues : la vue logique, la vue physique et la vue géométrique. Cela permet d'étudier un système à travers différents prismes de plus en plus complexes. La Figure 1.1 représente un diagramme de classe Langage de Modélisation Unifié, ou *Unified Modeling Language* (UML) définissant la classe «Architecture» telle que définie dans ces travaux. Les deux premiers attributs en noir sont utilisés dans toutes les vues et les deux suivants en mauve définissent la vue physique.

L'AO est un modèle représentant une technologie existante à l'aide de plusieurs vues. La vue logique permet de représenter le système modélisé par une boîte noire, de la même

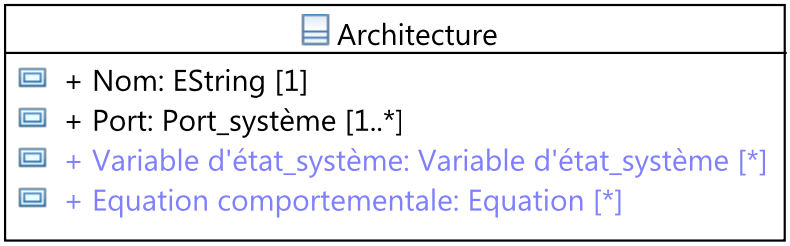


FIGURE 1.1 – Implémentation de la classe «Architecture» dans le manuscrit.

façon que cela est fait pour les exigences fonctionnelles de PAHL et collab. [2007]. Cette vue définit l'AO comme un objet contenant un ensemble de ports et un nom unique. La vue physique permet de modéliser le système à l'aide de variables d'échange et d'état, liées entre elles par des équations d'état et de comportement. Cette vue permet le dimensionnement du sous-système. La dernière vue est géométrique et permet la représentation et le placement des sous-systèmes dans l'espace. Elle permet donc d'obtenir des résultats sur les performances du système global avec une confiance plus grande que précédemment. La Figure 1.2 représente le diagramme de classe UML définissant la classe « AO » telle que définie dans ces travaux. Les deux premiers attributs en noir sont utilisés dans toutes les vues et les trois suivants en mauve définissent la vue physique de l'AO.

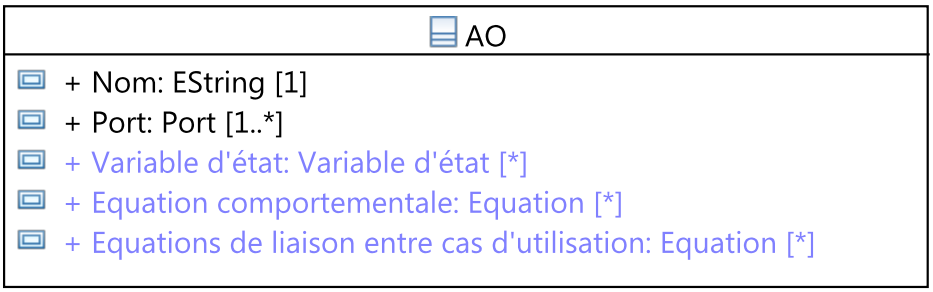


FIGURE 1.2 – Implémentation de la classe «AO» dans le manuscrit.

Un port est un objet capable de recevoir ou de donner un flux d'un type défini pouvant appartenir à trois familles : matière, énergie, signal. Ces types sont utilisés couramment et sont, bien sûr, la base des travaux de HIRTZ et collab. [2002]. De leur côté, PAILHÈS et collab. [2009] utilisent les flux d'énergie pour décomposer les fonctions en conception préliminaire. L'ontologie utilisée dans ce papier se focalise sur les flux énergétique et propose un ensemble de termes à utiliser pour écrire les fonctions textuelles qui est réduit par rapport à celui de HIRTZ et collab. [2002]. Un port est typé avec une direction qui peut être soit d'entrée, soit de sortie mais qui ne peut, en aucun cas, être d'entrée/sortie dans notre modélisation. Ce choix de modélisation est fortement contraint par la méthode de résolution du problème de synthèse aboutissant à la génération des architectures logiques. Un port appartient à un AO.
Lors du processus de synthèse, les ports sont connectés entre eux en respectant certaines règles qui seront décrites plus tard dans le manuscrit. La Figure 1.3 représente le diagramme de classe UML définissant la classe « port » telle que définie dans ces travaux. Le premier attribut en noir est utilisé dans toutes les vues, les quatre attributs suivants sont utilisés dans la vue logique et le dernier en mauve définit la vue physique du port. De la même manière qu'un objet différent est défini pour l'architecture vis-à-vis de l'AO, il en est de même pour les ports, qui sont légèrement différents dans la classe «Architecture».

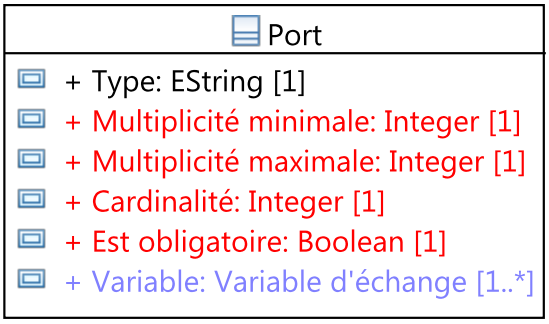


FIGURE 1.3 – Définition de la classe «port».

Un flux représente un transfert de matière, d'énergie ou d'un signal entre deux ports appartenant à des objets différents. Comme dit précédemment, ces flux sont typés avec le même type que le port qui les émet ou les reçoit. Dans la modélisation qui a été choisie et pour des raisons de simplification, le concept de flux est fusionné avec le concept du port.

Même si le langage de modélisation système Langage de Modélisation de Systèmes, ou *Systems Modeling Language* (SysML) n'est pas utilisé pour représenter les architectures que nous synthétisons, pour différentes raisons exprimées tout au long de ce manuscrit, il nous a semblé intéressant de présenter nos modèles à l'aide de diagrammes de classe de l'UML. Ainsi la Figure 1.4 décrit les relations d'héritage et les liens entre les différentes classes utilisées pour construire nos solutions architecturales. Une classe abstraite « Elément architectural » est créée. De cette dernière, héritent deux classes « port » et « AO ». La classe « AO » contient des références à des objets de type « port », tandis que la classe « Architecture » est définie comme héritant de la classe « AO ». Elle possède plusieurs objets de cette même classe et des objets de type « Connexion » liant deux objets de type « port ». Ce dernier point est une traduction directe du concept même de système défini par FAISANDIER [2015b], où un système peut être considéré comme un sous-système, tout dépend de la granularité considérée. La classe « Architecture » sera, tout au long de ce manuscrit, confondue avec les termes «système», «solution architecturale» ou, bien entendu, «architecture». Cet objet sera représenté sous plusieurs vues, sous la forme d'une spécification, d'une vue dite logique, d'une vue dite physique et d'une vue dite géométrique. Ces différentes vues et leurs moyens d'obtention seront décrits tout au long de ce document.

Les architectures logiques représentent des systèmes composés d'AO connectés entre eux par des flux. Ces architectures peuvent être représentées par des graphes Entrées multiples Sorties Multiples, ou *Multiple Inputs Multiple Outputs* (MIMO) pouvant être eux-mêmes représentés graphiquement, à l'aide d'un langage comme SysML, ou mathématiquement par une Matrice d'interconnexion, ou *Design Structure Matrix* (DSM) orientée. C'est-à-dire qu'en ligne, les connexions des ports de sortie sont visibles. Sur les colonnes, les connexions des ports d'entrée sont visibles. Les DSM sont des matrices carrées et sont composées en lignes et en colonnes par des ports contenus dans les AO de la base de données considérée pour le problème de synthèse. Concrètement, si on regarde la ligne correspondant à un port d'entrée, cette dernière ne sera composée que de zéros, même s'il est connecté dans l'architecture. Par contre, si on regarde la colonne correspondant à ce port, elle contiendra aussi des uns. Ce qui est l'inverse pour les ports de sortie. Une illustration est donnée sur la Figure 1.5 : la première ligne correspond à un port d'entrée. On voit qu'elle ne contient que des zéros, alors que la première colonne correspondant au

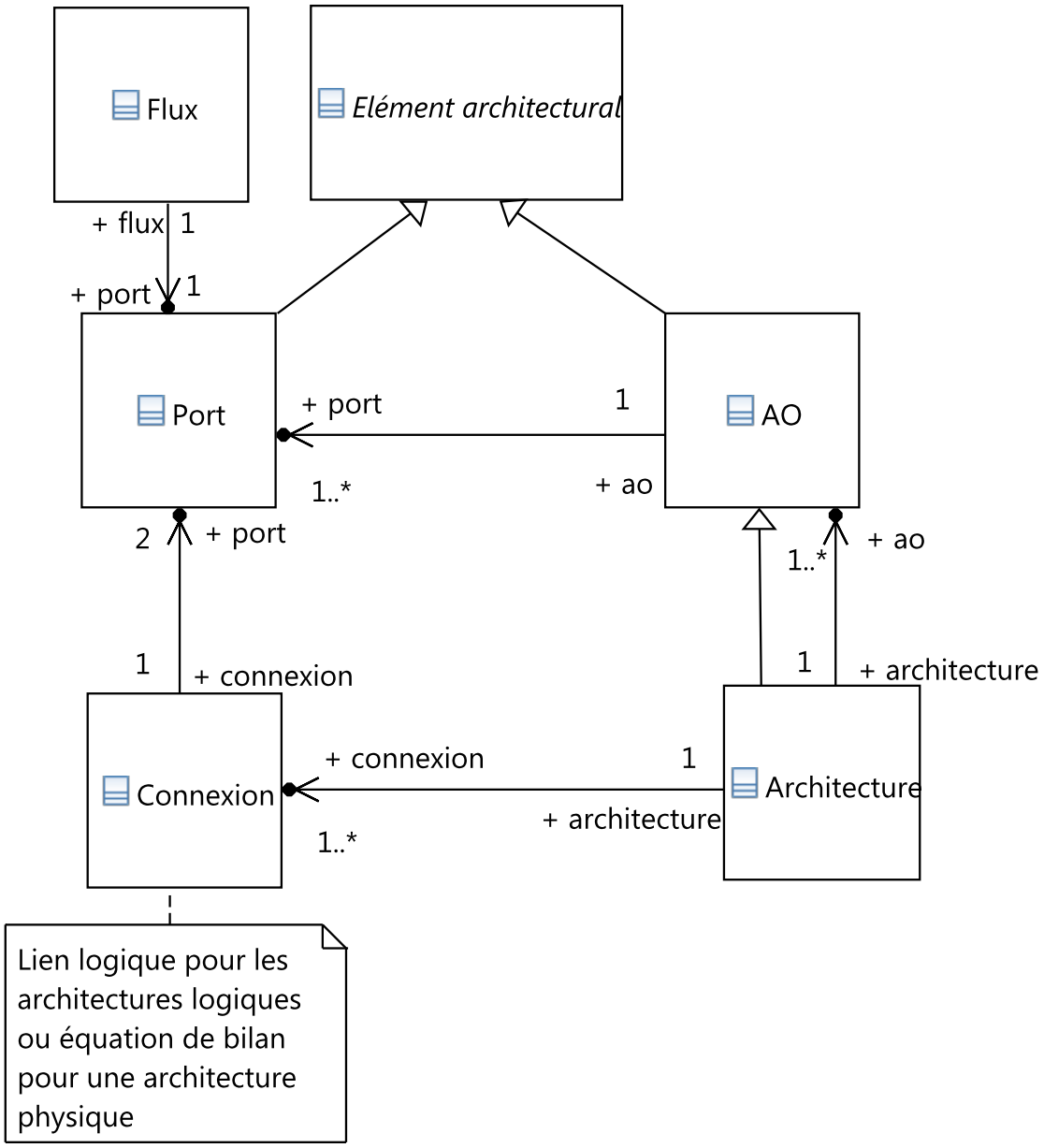


FIGURE 1.4 – Métamodèle de la synthèse architecturale d'après les travaux de ce manuscrit.

même port contient un chiffre un.
Les DSM permettent la visualisation des interfaces entre les différents AO et sont, surtout, le premier niveau de représentation d'un système à concevoir dans notre méthodologie de synthèse assistée par ordinateur.

$$\begin{pmatrix} 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 1 & \cdots & 1 & 0 & 0 \end{pmatrix}$$

FIGURE 1.5 – DSM orientée.

Les architectures physiques sont des modèles mathématiques composés d'équations non différentielles. Ces équations lient des variables d'état et d'échange contenues dans les différents AO composant le système. Elles permettent le pré-dimensionnement du système. La Figure 1.6 représente le diagramme de classe UML définissant les classes de variables implémentées dans les codes. Tous les attributs sont utilisés uniquement dans la vue physique, étant donné que la vue logique ne contient pas d'objet « variable ».

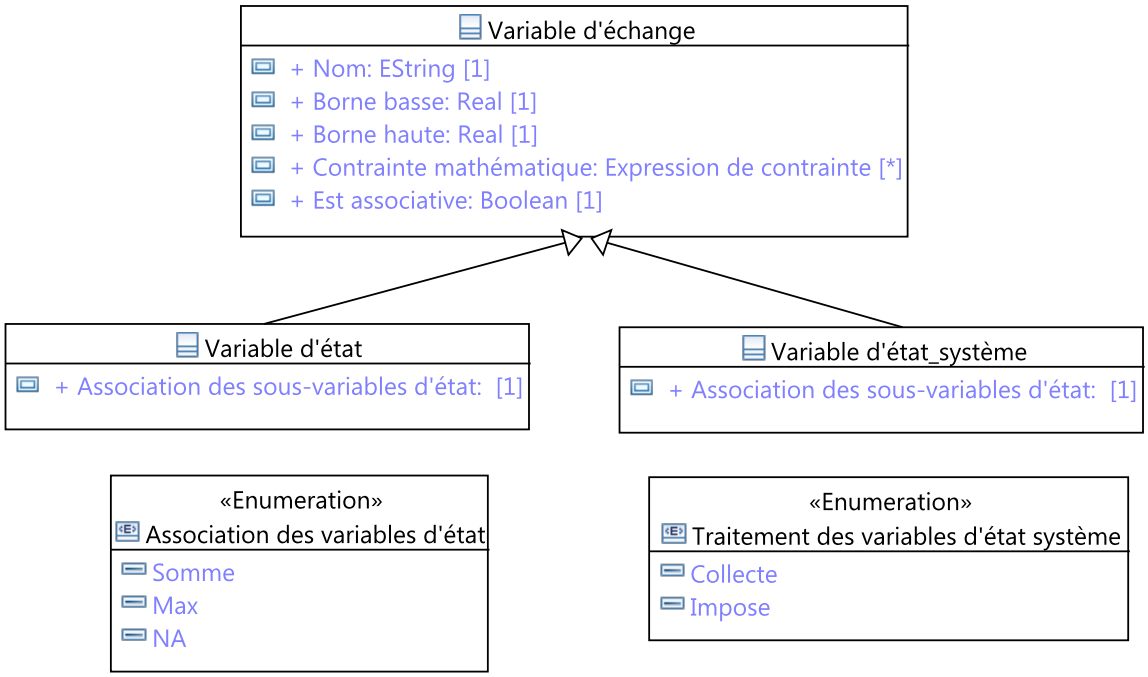


FIGURE 1.6 – Implémentation des classes de variables dans le manuscrit.

L'architecture géométrique est une représentation géométrique simplifiée du système à concevoir. Elle permet de faire des études de placement des différents AO et donc de calculer les premiers centres de masse du système complet. Le présent manuscrit n'a traité ce sujet que de manière partielle et uniquement théorique car il s'agit d'une étape qui n'a pas pu être traitée durant la thèse de doctorat. Néanmoins, il est possible d'imaginer un

couplage avec OpenVSP, un logiciel libre de modélisation géométrique et paramétrique de la NASA. GALLAHER [2017] utilise ce logiciel pour estimer des propriétés de masses et d'inerties sur de nouvelles architectures d'hélicoptères, en utilisant un algorithme de placement automatique des éléments à l'aide d'une base de données des technologies existantes. C'est-à-dire que si, habituellement, une boîte de transmission hélicoptère est positionnée au centre du fuselage, cet algorithme placera la boîte de transmission de cette nouvelle architecture, si elle en contient une, au milieu du fuselage.

La méthodologie de synthèse assistée par ordinateur développée dans ce manuscrit et illustrée sur la Figure 1.7 répond aux attentes décrites par CAGAN et collab. [2005] en reprenant les quatre grands axes développés dans le papier. La Figure 1.7 est un des apports principaux de la thèse, il est le résultat des choix qui ont été faits au regard de l'analyse bibliographique et des travaux de thèse. La méthode est découpée en trois parties avec la capture des exigences s'appliquant sur le système à concevoir, la génération des solutions et leur représentation. La partie génération est, elle-même, décomposée en trois avec une partie de génération logique qui est le cœur du travail, suivie d'une étape d'analyse physique des solutions ou encore de dimensionnement, conclue par un placement géométrique des éléments dans l'espace avec une génération de 3D.

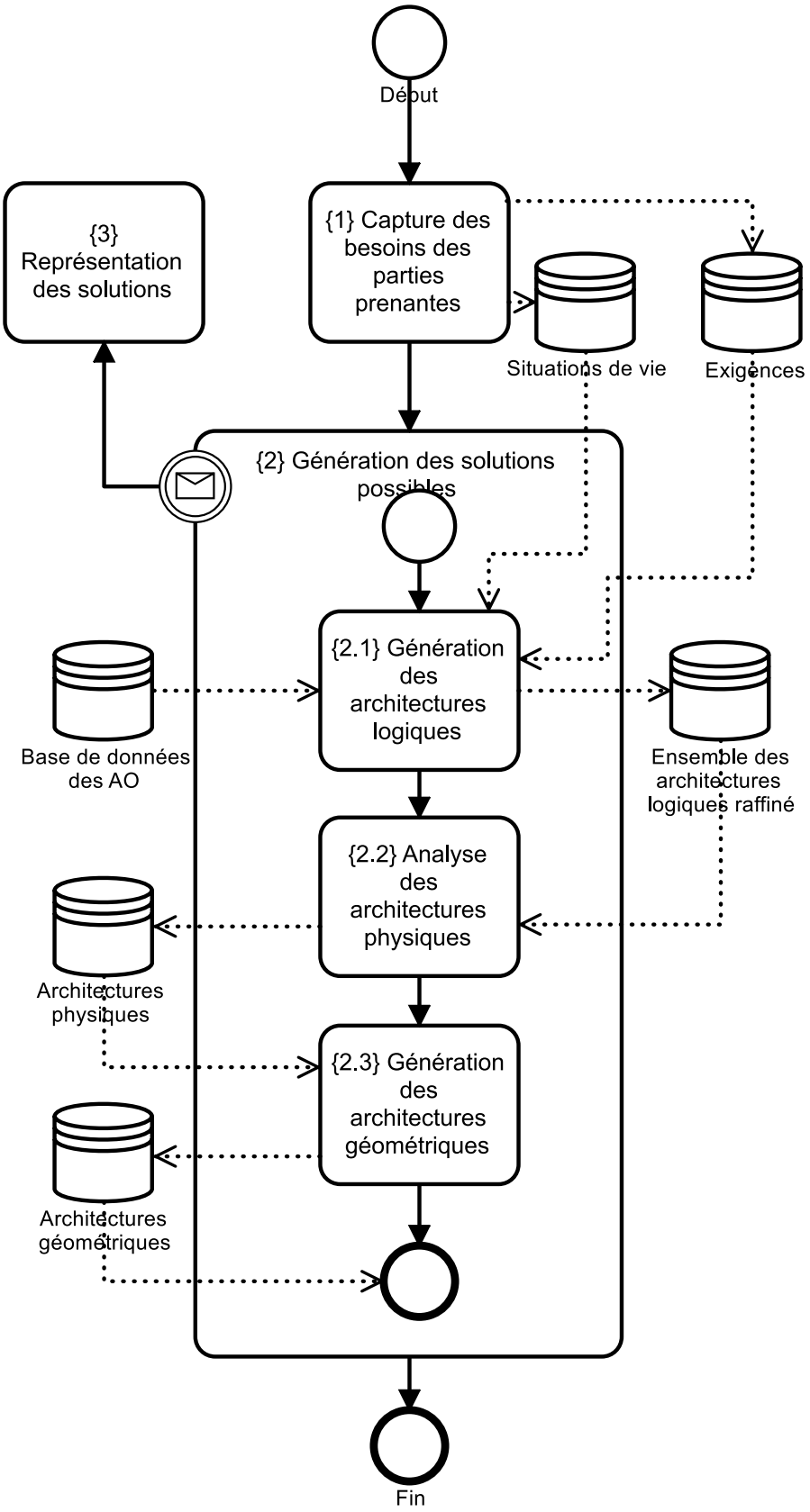


FIGURE 1.7 – Vue globale de la méthodologie provenant des travaux de thèse présentés.

Chapitre 2

Mise en perspective des travaux par rapport à l'état de l'art

« *L'abeille confond, par la structure de ses cellules de cire, l'habileté de plus d'un architecte. Mais ce qui distingue dès l'abord le plus mauvais architecte de l'abeille la plus experte, c'est qu'il a construit la cellule dans sa tête avant de la construire dans la ruche...* »

Marx, Le Capital, p. 728 de l'Édition Pléiade

Sommaire

2.1 Revue bibliographique

Une multitude d'approches ont déjà été développées dans la littérature pour faciliter les étapes de conception préliminaires. Quatre types de méthode de résolution de problèmes en conception sont présentés par KRYSSANOV et collab. [1999] :

— Essai-erreur
— Règles empiriques
— Analyse scientifique
— Méthodes de synthèse

La méthode d'essai-erreur appliquée à la génération d'architecture automatique prend beaucoup de temps de calcul car elle parcourt l'ensemble de l'espace des solutions. Ce type de recherche est impossible à mettre en place dès que la taille des bases de données devient trop importante. En supposant que toutes les solutions émanant des bases de données puissent être trouvées, elles ne permettront pas toutes de répondre aux besoins des parties prenantes étant donné qu'elles n'auront pas été générées à partir de ces besoins.
La méthode d'application de règles empiriques sur une base de données dans le but de générer des architectures candidates correspond à une méthode d'essai-erreur appliquée à un espace de conception réduit par les règles.
Les méthodes d'analyse scientifique sont des méthodes mathématiques reposant sur la logique, les statistiques ou encore la théorie des ensembles.
Les méthodes de synthèse sont une façon de traduire les besoins des parties prenantes en exigences sur le système à concevoir puis en solutions technologiques. Ces méthodes permettent aux architectes d'explorer de manière organisée l'espace des solutions en fermant au plus tôt des branches d'alternatives.

Les travaux présentés dans ce manuscrit portent sur l'application d'une méthode de synthèse car nous nous efforçons de transformer en alternatives architecturales un ensemble de besoins exprimés par des parties prenantes. Chaque besoin est transformé en exigences qui permettent de réduire l'espace de recherche. Une fois que l'espace de recherche est réduit à son minimum, alors une combinatoire sous contraintes est utilisée.

La Figure 2.1 présente, par une analyse systématique des termes composant la problématique de thèse, les champs de connaissance et les méthodes les plus pertinents vis-à-vis des travaux présentés dans ce manuscrit.

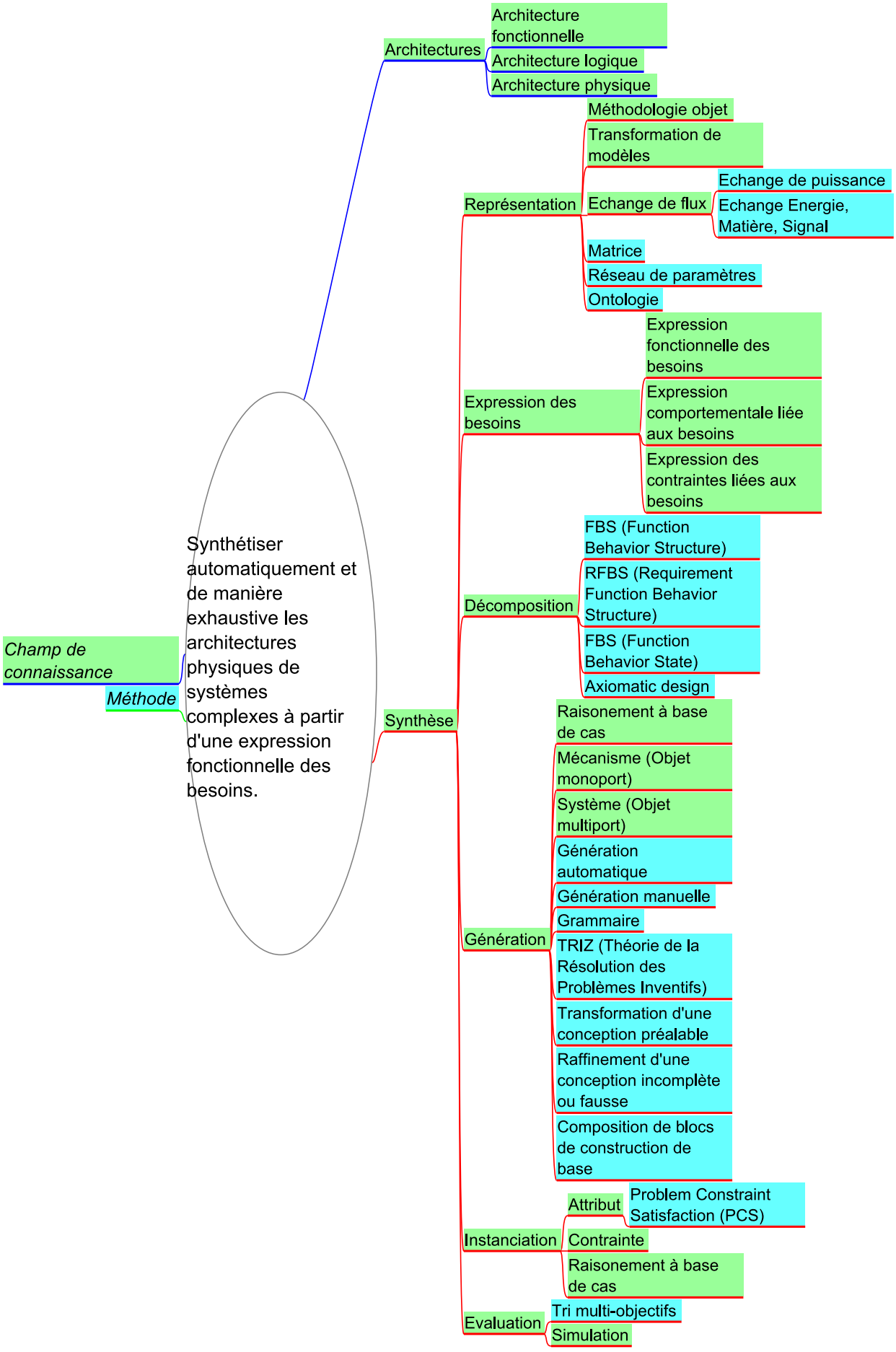


FIGURE 2.1 – Cartographie des champs bibliographiques nécessaires à la résolution du sujet.

On peut classer les approches de synthèse en deux grandes familles, la synthèse pour la conception de mécanismes et la synthèse pour la conception de systèmes.
La première approche a pour objectif la résolution d'une cinématique voulue à l'aide d'éléments cinématiques de base tels qu'une liaison pivot glissant ou une liaison rotule, comme dans l'exemple traité par LI et collab. [1999]. Les échanges entre les sous-systèmes sont uniquement de type mécanique, les blocs de construction peuvent être vu comme des fonctions de transfert. Elles prennent un mouvement donné en entrée et fournissent un mouvement transformé en sortie.
La deuxième approche permet la résolution d'un système d'après une liste d'exigences fonctionnelles et une liste de technologies disponibles. Les échanges entre les sous-systèmes sont de types hétérogènes. Notre ambition étant d'assister la conception architecturale de systèmes complexes, nos travaux se rangent dans la famille des approches de synthèse pour la conception de systèmes, comme ceux de UMEDA et collab. [1996] et de CHRISTOPHE et collab. [2009].

HAN et LEE [2006] décrivent trois stratégies de synthèse en conception qui sont : la stratégie de transformation, qui modifie une conception existante, la stratégie de raffinement, qui part d'une conception incomplète ou même fausse et la stratégie de composition, qui permet de partir de spécifications et d'assembler des blocs de construction de base. Dans ce papier, le concept de «générateur de fonction virtuelle» permet de ramener au même niveau de granularité des assemblages et des blocs de construction de base. Par exemple, cela peut permettre la prise en compte de deux systèmes de niveau de détail différent, tel qu'un moteur thermique, qui est un système complexe, et un réservoir de carburant, qui est un système beaucoup plus simple. Ensuite, un mécanisme est décrit par un assemblage de «générateurs de fonctions virtuelles».

CAGAN et collab. [2005] définissent les différentes étapes clés d'une synthèse architecturale sur la Figure 2.2. Quatre grandes étapes ressortent de cette figure : la définition du problème de conception, la génération des solutions, la représentation des solutions et l'évaluation des solutions. Après cette étape de synthèse, les architectes peuvent tirer des conséquences des solutions architecturales obtenues et redéfinir les objectifs si nécessaire, ou passer à une conception plus détaillée sur les architectures ainsi générées et validées. La synthèse architecturale peut être automatisée lorsque beaucoup d'alternatives doivent être générées et évaluées, ce qui est le cas de la plupart des systèmes actuels, qui sont des systèmes complexes et qui peuvent être composés de beaucoup d'éléments hétérogènes.

Dans la suite de ce chapitre les références bibliographiques sur lesquelles se basent les travaux de thèse sont présentées. L'analyse de ces références suit le processus de synthèse développé et illustré dans le chapitre 1 sur la Figure 1.7.

2.1.1 La représentation ou modélisation

D'après CAGAN et collab. [2005], lors d'un processus de synthèse, chaque élément constituant le système à concevoir doit être représenté : les fonctions, les objets technologiques, les connexions internes à l'architecture, etc... L'utilisation de modèles revient de facto à adresser le problème de représentation soulevé précédemment. Un modèle est un concept très souvent utilisé, mais sa définition générale est difficile à exprimer de manière non ambiguë. La meilleure façon de définir un modèle est de comprendre dans quel but il est utilisé. En effet, un système à concevoir peut être représenté par plusieurs modèles, avec pour objectif la capture de ses différents aspects. C'est pour cela que la première chose à faire lors de la définition d'un modèle est de préciser les aspects qui doivent être

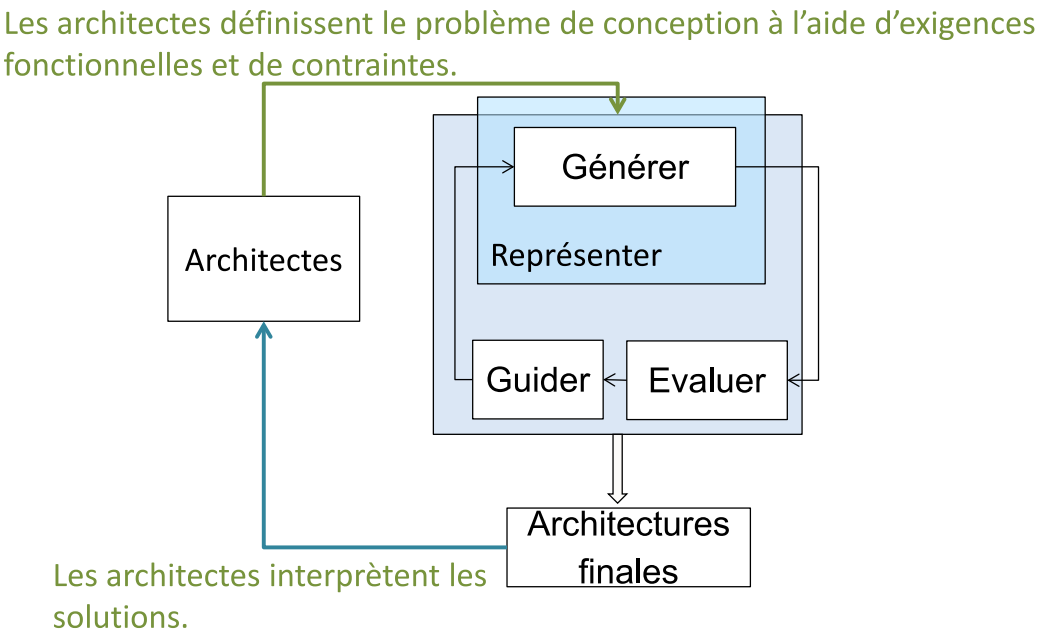


FIGURE 2.2 – Méthodologie de synthèse architecturale d'après CAGAN et collab. [2005].

étudiés. Un modèle peut être plus ou moins formel mais doit toujours être non ambiguë dans sa définition. Pour ce faire, un langage de modélisation bien défini doit être utilisé pour décrire le modèle. Ce point de vue est partagé avec FAISANDIER [2015b]. Le guide de l'Ingénierie Système, ou *The Guide to the Systems Engineering Body of Knowledge* (SEBoK) définit deux types de modèles : descriptifs et analytiques. Un modèle descriptif détaille les relations logiques entre différents objets qui peuvent être hétérogènes, tandis qu'un modèle analytique décrit des relations mathématiques entre des variables d'un système. Un autre type de modèle, appelé hybride, peut aussi être décrit dans le but de définir des relations logiques et mathématiques dans une même perspective.

Passons maintenant en revue quelques outils ou méthodes de modélisation qui nous ont aidés à définir nos propres modèles architecturaux.

ZHENG et collab. [2016] décrivent un modèle d'interfaces pour la représentation des connexions entre les différents objets présents dans une architecture. Une interface est une classe décrite par quatre attributs : son nom, son type, sa configuration et sa désirabilité. Le nom assure l'unicité de chaque interface. Le type décrit une interface, comme par exemple une interface géométrique, un échange d'énergie, un échange de données ou une interface de contrôle. La configuration indique entre quels types d'objets une interface s'applique : en effet, on parle toujours d'une interface entre deux objets. Ces configurations sont : interface entre deux AO, interface entre un AO et un environnement, interface entre un AO et une interface, interface entre deux interfaces, interface entre une interface et un environnement. Les configurations présentées ci-dessus permettent la définition de trois types de ports qui sont : le port d'un composant, le port d'un environnement et le port d'une interface. Des règles de compatibilité permettent de vérifier que les connexions entre les différents objets sont possibles.

De leur côté, afin de gérer les interfaces, BONEV et collab. [2015] étendent la représentation DSM avec la définition de plusieurs types de connexions entre les objets composant le système. Les objets composant le système sont de types hétérogènes et sont décrits

comme étant : «Composant», «Type de composant», «Attribut» et «Contrainte». Ces objets sont liés par des liens de différents types décrits comme : lien entre composants, lien entre types de composants, collaboration entre composants, lien entre des contraintes et lien entre des attributs. Une collaboration entre deux composants existe quand des attributs de ces composants sont reliés par une contrainte.
Cette représentation prend le parti de créer autant de types de connexions que de types d'objets existants. Dans le présent manuscrit, les connexions ne sont autorisées que sur les ports. Les autres objets constituant nos modèles sont connectés entre eux car il s'agit d'attributs des ports connectés.

CHIKHAOUI [2013] présente d'autres outils de modélisation pour représenter énergétiquement l'hélicoptère. Cette modélisation doit ensuite servir à analyser le comportement de l'hélicoptère d'un point de vue énergétique. Les langages de modélisation sélectionnés dans la thèse de CHIKHAOUI [2013] sont le multibond graph issu du bond graph décrit par OULD BOUAMAMA et DAUPHINTANGUY [2006] et la Représentation Energétique Macroscopique (REM). Ces deux langages sont principalement utilisés pour la modélisation et la simulation physique des systèmes. Dans notre étude, comme cela sera développé plus tard, nous ne traitons pas de simulation, mais de choix architecturaux. C'est-à-dire que les caractéristiques physiques du système ne sont pas des constantes, mais bien des variables d'état qui doivent être déterminées non pas à l'aide de simulations, mais à l'aide de règles de pré-dimensionnement. Lorsqu'on parle de simulation, le modèle à simuler est carré, ses paramètres dimensionnants sont déjà déterminés. Lors d'un pré-dimensionnement ou d'une optimisation, tous les paramètres dimenssionants ne sont pas déterminés. L'utilisation des bond graphs, ou encore de la REM, ne s'avère peut être pas pertinent car ces deux outils sont plus utilisés pour de la simulation que pour du pré-dimensionnement.

La méthodologie bond graph permet la modélisation de systèmes mécatroniques du point de vue des échanges énergétiques. Chaque lien entre deux blocs est modélisé par une variable de potentiel et une variable de flux. Le produit de ces deux variables doit correspondre à une puissance.
De la même manière, le bond graph à mots, présenté par DAUPHIN-TANGUY [2010], représente le niveau technologique de la modélisation, avec chaque bloc qui représente une technologie simple. Ce niveau de représentation est intéressant car il permet une modélisation logique d'un système. C'est à dire se focalisant sur les sous-systèmes et leurs interconnexions.

D'un autre côté, les approches Ingénierie des Modèles, ou *Model Based System Engineering* (MBSE) mettent au centre des activités de conception les modèles et non plus la documentation. Durant les différentes phases de conception, il est nécessaire d'enrichir les modèles du système à concevoir. Chaque métier conçoit son propre modèle du système, mais, à la fin, un modèle global doit être construit pour simuler dans sa globalité le système à concevoir. BARBEDIENNE et collab. [2015] mettent en exergue l'apparition de langages spécialisés dans la transformation de modèle du type Modèle à Modèle, ou *Model to Model* (M2M) et Modèle à Texte, ou *Model to Text* (M2T). Ces langages permettent l'interopérabilité entre les modèles et la rédaction automatique de documentations pour faciliter les échanges entre les experts qui ont conçu les différents modèles. D'autres utilisations des M2M sont possibles, par exemple l'écriture d'un unique modèle devant être résolu par plusieurs solveurs différents. CHENOUARD et collab. [2008] utilisent un langage de modélisation nommé s-COMMA qui permet de créer des systèmes de contraintes à résoudre. Ces systèmes peuvent ensuite être résolu avec plusieurs solveurs sans aucun effort de réécriture

car la plateforme s-COMMA se charge de réinterpréter les systèmes différemment suivant le solveur qui est utilisé. BARBEDIENNE et collab. [2015] présentent trois approches pour la conception d'une architecture au travers d'échanges d'informations entre un modèle logique, un modèle géométrique et un modèle physique. Les approches abordées dans ce papier peuvent sembler intéressantes mais les modèles ne sont malheureusement pas détaillés, ce qui limite la contribution de cet article.

2.1.2 Les éléments bibliographiques qui supportent la formulation du besoin

L'extraction du besoin des parties prenantes est une tâche importante de la conception d'un produit au sens large pour laquelle des méthodes ont déjà été développées comme le fait remarquer HAMIDA et collab. [2015] dans une analyse non-exhaustive des méthodes de modélisation fonctionnelle. La tâche de capture des besoins des clients est traitée grâce à des outils de l'ingénierie système, tels que des bêtes à cornes pour la méthodologie APTE, des formulaires pour aider à la formulation du besoin KROB [2014] et des études sur l'ensemble des cycles de vie que le système aura à supporter, ainsi que sur l'ensemble des parties prenantes auxquelles il aura affaire durant tous les cycles de vie. Ce dernier point peut être illustré par une méthode décrite par un regroupement entre l'Association Française pour l'Ingénierie Système (AFIS), l'Atelier Inter-Etablissements de Productique et Pôle de Ressources Informatiques pour la MECAnique (aip primeca) et un groupement d'enseignants de l'Education Nationale Française.
Récemment, beaucoup de nouvelles méthodologies de conception ont émergé de la communauté, comme celle de MHENNI et collab. [2014]. Cela est dû à l'apparition de nouveaux langages comme le UML et le SysML qui permettent de représenter des concepts complexes d'après les nouveaux standards de l'*Object Management Group* (OMG) et du Conseil International sur l'Ingénierie Système, ou *International Council on Systems Engineering* (INCOSE). Cet environnement favorable, même s'il n'est pas parfait comme discuté par HAMPSON [2015], permet aux groupes de recherche de présenter des processus de conception innovants qui prennent en compte, ou pas, l'extraction des besoins comme CHAPON et BOUCHEZ [2009] ou MATAR [2013].

Le problème de l'analyse fonctionnelle a été régulièrement traité et différents outils peuvent être utilisés pour capturer, transformer et décomposer les besoins et les contraintes en fonctions indépendantes de toute solution technologique. Néanmoins, il nous semble que les représentations fonctionnelles développées jusqu'à présent ne permettent pas une modélisation suffisamment normée, qui pourrait être utilisée pour de la génération automatique.
En effet, l'expression du besoin fonctionnel d'un produit à concevoir et, surtout, l'intégration des fonctions dans un processus de traitement automatisé est un axe de travail difficile. Il est nécessaire de transformer des connaissances subjectives en éléments objectifs qui seront analysés et transformés par un ordinateur, comme cela est fait dans la thèse de HELMS [2012].

CHAKRABARTI et BLESSING [1996] présentent les résultats de plusieurs groupes de recherche travaillant sur la représentation et l'usage des fonctions dans la phase de conception d'un produit. Il en ressort trois représentations fonctionnelles : la paire verbe-nom qui est la plus ancienne forme encore utilisée, la paire vecteur de ports d'entrée-vecteur de ports de sortie où les ports de sortie peuvent être de type énergie, matière ou information

et la paire situation à l'entrée-situation à la sortie. Pour comprendre la différence entre ces trois représentations fonctionnelles, un exemple simple est proposé :

— Paire verbe-nom : Transformer de l'énergie électrique en énergie mécanique,

— Paire vecteur de ports d'entrée-vecteur de ports de sortie : Port électrique en entrée et port mécanique en sortie et

— Paire situation à l'entrée-situation à la sortie : Reçoit de l'électricité, donne de l'énergie mécanique.

Ces représentations doivent permettre de trouver facilement des correspondances entre les fonctions et les composants.

UMEDA et TOMIYAMA [1997] rappellent quelques modèles de représentations fonctionnelles dont le Langage de Représentation Fonctionnelle Causal, ou *Causal Functional Representation Language* (CFRL), présenté en détail par IWASAKI et collab. [1993]. Ce langage permet les modélisations fonctionnelle et comportementale en parallèle. Chaque comportement est décrit par une Description de Processus Causal, ou *Causal Process Description* (CPD), qui représente une séquence d'évènements.
Cette définition fonctionnelle est complexe et ne correspond pas à ce que l'on attend d'une fonction, c'est-à-dire être décorrélée de tout objet. En effet, une fonction est définie par le triplet DF, CF, GF où :

— DF représente le composant technique qui a pour fonction F.

— CF représente le contexte dans lequel le composant doit fonctionner.

— GF décrit le but que la fonction doit atteindre.

Dans la modélisation que nous souhaitons utiliser, nous voulons totalement décorréler la solution technologique, de la fonction. C'est-à-dire que nous ne souhaitons pas utiliser une liste de technologies qui seraient liées à des fonctions et ensuite appelées dans des solutions pour répondre à une liste d'exigences représentant le système à concevoir. Nous souhaitons laisser libre tout le champ des solutions possibles, sans pré-orienter les solutions vers des architectures connues.

La méthode de modélisation des fonctions à l'aide des bond graphs, utilisée par l'outil Schemebuilder présenté par BRACEWELL et collab. [1992], est une méthode adaptée à la simulation énergétique, mais n'est pas autant adaptée à des systèmes échangeant des flux d'informations. Dans le présent manuscrit, nous aurons besoin de deux modèles architecturaux, l'un pour représenter les architectures logiques des solutions et l'autre pour faire l'analyse physique de ces dernières. La méthodologie bond graph est habituellement utilisée pour faire de la simulation multiphysique sur des systèmes causaux. Un système est dit causal quand ses sorties ne peuvent être déterminées avant ses entrées. Ce n'est donc pas l'outil idéal pour modéliser les deux vues qui nous intéressent, même si les éléments de langage illustrés par la figure 2.3 pourraient servir à décrire nos AO.
STONE et WOOD [1999] entreprennent avant HIRTZ et collab. [2002] une réduction de l'espace fonctionnel habituel en restreignant l'usage des mots et verbes anglais, pour éviter les ambiguïtés et la redondance des fonctions. La taxonomie ainsi décrite est issue de l'étude de plusieurs autres travaux afin de former une liste exhaustive mais minimaliste de fonctions. Cette réduction du vocabulaire peut permettre la transformation d'exigences textuelles en exigences logiques de manière systématique, surtout si elle est couplée à l'utilisation d'un canevas à respecter dans la construction des phrases, comme cela est préconisé par KROB [2014]. Ce canevas est le suivant :

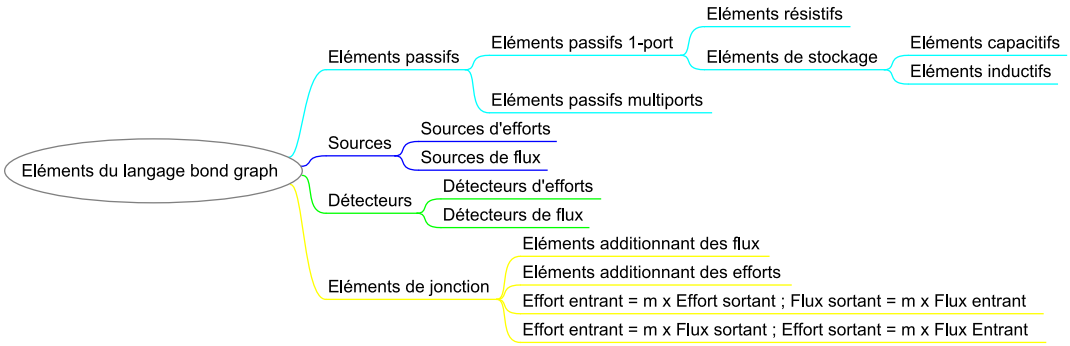


FIGURE 2.3 – Eléments du langage bond graph d'après DAUPHIN-TANGUY [1999].

«Le système à concevoir» «doit faire quelque chose» «sous une certaine performance» «dans un certain mode de fonctionnement».
A l'aide de règles, des boîtes fonctionnelles peuvent être créées directement depuis des exigences textuelles portant sur de la transformation de flux, dans le même esprit que CHRISTOPHE et collab. [2009]. La différence dans cette publication est que les auteurs utilisent un dictionnaire de synonymes du Centre National de la Recherche Scientifique (CNRS) dans le but de modifier des fonctions qui ne seraient pas énoncées avec les verbes et flux déclarés dans la taxonomie de Hirtz. De par cette transformation, un ingénieur peut converser sans utiliser un langage dédié, grâce à un programme informatique, puisque les entrées non conformes au dictionnaire de Hirtz seront modifiées avant d'être traitées. Concrètement, cela revient à utiliser un logiciel qui réécrit automatiquement les fonctions renseignées par l'utilisateur à l'aide d'un dictionnaire de synonymes pour se rapporter à des fonctions connues par le logiciel de synthèse architecturale. KANG et TUCKER [2015] utilisent aussi la reconnaissance du texte pour associer une fonction à un système. Pour éviter le bruit dans la détection des fonctions, le texte descriptif des systèmes est nettoyé automatiquement en supprimant tous les termes de liaison. L'objectif est de quantifier les interactions fonctionnelles entre des modules réels représentant des sous-systèmes, à l'aide de la description des modules sous forme de textes. Cela permet de voir quels sont les sous-systèmes étant liés fonctionnellement.

BURGE [2013] explicite une méthode de modélisation fonctionnelle. Il traite de la modélisation de diagrammes de contexte, de la modélisation des échanges de flux entre fonctions, de la décomposition fonctionnelle classique et de l'utilisation d'un dictionnaire sur les ports. Ce dictionnaire, contrairement à celui proposé par HIRTZ et collab. [2002], définit en plus une grammaire, de manière à pouvoir composer de nouveaux ports à partir de ceux existants et de règles logiques. Le résultat de la décomposition fonctionnelle est un diagramme de flux entre des fonctions. Le diagramme de contexte permet de connecter, à l'aide de flux, le système à concevoir avec les éléments extérieurs, aussi appelés les terminaux. Il s'agit soit de sources, soit de récepteurs externes au système.
Une fonction est représentée à l'aide d'un langage basé sur une liste de verbes, de mots et de mots logiques et de règles de combinaisons possibles.

Après avoir été exprimées à l'aide d'un certain formalisme, les exigences fonctionnelles sont souvent décomposées en sous-fonctions pour être ensuite allouées à des solutions

technologiques.
Une des méthodes de décomposition et d'allocation des exigences, fonctionnelles ou non, la plus utilisée est la méthode Functional Analysis Systems Technique (FAST) présentée par WIXSON [1999]. Cette méthode date des années 1960 et est largement diffusée dans la formation des ingénieurs français. Le FAST raffine une fonction principale en plusieurs sous fonctions jusqu'à ce que les fonctions de plus bas niveau ne puissent plus être décomposées.
Certains groupes de recherche ont rajouté un étage dans cette décomposition fonctionnelle avec l'ajout d'une étape se focalisant sur les comportements du système. Ces méthodes de décomposition prennent l'acronyme Function Behaviour Structure (FBS) pour GERO [1990] ou Function Behaviour State (FBS) pour UMEDA et TOMIYAMA [1997].
Il est à noter, comme le rappellent UMEDA et TOMIYAMA [1997], que les deux méthodes sont indépendantes et que le FBS (Gero) de GERO [1990] ne prend pas en compte la notion d'état, nécessaire à la description des transitions entre deux états.
La méthode Requirement Function Behaviour Structure (RFBS) de CHRISTOPHE et collab. [2010] est une déclinaison du FBS (Gero), où une phase de passage entre les besoins et les fonctions est rajoutée. Les comportements de la structure instanciée sont comparés avec les comportements requis par les besoins exprimés. Si les deux ensembles de comportements ne sont pas assez proches, la structure est rejetée.

L'Axiomatic Design décrit par PARK [2007], et créé par Suh dans les années 1990, est une méthode qui repose principalement sur deux axiomes.
Le premier est l'axiome de l'indépendance. Il annonce qu'une conception optimale s'efforce de maintenir l'indépendance des fonctions. Cela veut dire que chaque fonction doit être réalisée par un système qui n'affecte pas la réalisation des autres fonctions.
Le deuxième axiome déclare que les flux d'information échangés entre les différents systèmes doivent être minimisés. Cet axiome concernant les échanges minimaux entre deux sous-systèmes est une recommandation récurrente, qui apparait aussi dans RECHTIN [1991].
La méthode de « zigzagging », introduite dans l'Axiomatic Design, doit permettre une décomposition simultanée des fonctions et des solutions physiques. Cette décomposition oblige le concepteur à trouver pour chaque fonction de la décomposition fonctionnelle un système physique qui la satisfera. FAISANDIER [2015a] fait le même constat : à savoir que l'architecture fonctionnelle et l'architecture physique se construisent conjointement. Il est en effet clair que la concrétisation d'une fonction en un AO peut introduire un nouveau besoin fonctionnel lié à ce nouvel objet. Ce nouveau besoin induit doit être exprimé et peut conduire à la nécessité d'un AO supplémentaire afin de satisfaire la nouvelle situation.
Par exemple, JANTHONG et collab. [2010] utilisent le concept de « zigzagging » dans le cadre du raisonnement à partir de cas (Case Based Reasoning (CBR)), où des conceptions fructueuses sont réutilisées et adaptées par rapport aux nouveaux besoins. Les auteurs développent un processus d'adaptation en trois étapes : proposer, évaluer et modifier. Ce processus peut faire penser à la méthodologie RFBS de CHRISTOPHE et collab. [2010] où le comportement désiré est tout d'abord comparé avec le comportement souhaité et où, ensuite, des modifications sont apportées au système à concevoir si les deux comportements sont trop éloignés.

YUAN et collab. [2016] proposent une méthode de décomposition fonctionnelle automatique, basée sur la décomposition de la fonction principale du système en effets fonctionnels, qui sont ensuite décomposés en effets physiques. Les fonctions sont repré-

sentées par des boîtes noires transformant des flux d'entrée en flux de sortie.
Les effets fonctionnels décrivent des changements qui s'opèrent sur les ports d'entrées, il y en a donc autant que de changements sur les ports d'entrées. Deux types de changements sont définis, les changements de type de flux et les changements de la valeur d'un flux. Par exemple, un changement de type de flux peut permettre la transformation d'un flux d'énergie électrique en un flux d'énergie mécanique et un changement de valeur d'un flux peut permettre une augmentation d'énergie mécanique.
Les effets physiques sont des principes de solution possibles et ils proviennent de la décomposition des effets fonctionnels. Certains effets physiques induisent l'apparition de fonctions de support dans la décomposition fonctionnelle. Par exemple, l'effet physique de convection induit l'obligation d'avoir une source de chaleur à disposition.

GIAMPÀ et collab. [2004] utilisent une base de données générée grâce à l'étude des éléments standards de l'INTERNATIONAL ORGANIZATION FOR STANDARDIZATION [2005]. Tous les éléments ont été étudiés et les fonctions élémentaires correspondantes ont été groupées en neuf classes fonctionnelles qui sont : bloquer, placer, contenir, convoyer (de la matière ou du signal), dissiper, fournir, transformer, transmettre (de l'énergie) et employer. L'ingénieur d'étude sélectionne ensuite dans la base de données les fonctions qu'il souhaite accomplir. L'objectif est de développer une grammaire fonctionnelle où des fonctions peuvent être reliées entre elles pour former un modèle fonctionnel. Ces fonctions sont représentées par une boîte noire qui comporte des ports de types énergie, force, matière et signal. Ces travaux sont intéressants car ils répondent à notre besoin de modélisation des exigences et des architectures logiques. Néanmoins, chaque fonction reste liée à certaines technologies, ce qui veut dire que, dès qu'une fonction sera demandée par l'utilisateur, une des technologies liées sera forcément présente dans la solution finale. Il ne pourra donc pas y avoir de combinaisons de technologies pour répondre à une fonction demandée si cela n'a pas été précédemment défini dans l'outil d'aide à la conception. Les travaux que nous présenterons dans le reste du manuscrit répondent à cette problématique de combinaison de technologies pour répondre à une fonction, sans pour autant avoir défini directement un lien entre l'AO et la fonction élémentaire.
CHIOU et SRIDHAR [1999] présentent la même méthode que GIAMPÀ et collab. [2004] quant à l'étude des éléments standards que l'on peut trouver dans la littérature, mais uniquement pour des blocs de construction de type « mécanisme ». Cela revient à dire que les seules données d'entrée et de sortie du problème à résoudre seront géométriques et cinématiques. Ces travaux aboutissent à la création de quarante-trois blocs de construction physiques de base, qui peuvent ensuite être combinés entre eux pour répondre à un besoin exprimé à partir d'un bloc de construction fonctionnel correspondant à une transformation de mouvement requise. CHIOU et SRIDHAR [1999] définissent trois types de mouvement : la translation, la rotation et le mouvement hélicoïdal qui est un mélange des deux précédents. Les blocs fonctionnels de base sont alors définis au nombre de cinq. Il s'agit des duets : translation/translation, rotation/rotation, translation/rotation, hélicoïdal/translation et hélicoïdal/rotation.
Trois niveaux de représentation sont introduits pour les blocs de construction. Le premier niveau permet la représentation du type de transformation de mouvement, ainsi que la différence d'orientation entre les axes du mouvement d'entrée et de celui de sortie. Le second niveau permet aussi la représentation du type de transformation de mouvement, ainsi que l'orientation des axes d'entrée et sortie des mouvements, mais de manière plus précise. De même, des informations sur la continuité, la linéarité, l'interchangeabilité et la direction des mouvements sont rajoutées à la représentation dans le second niveau. Le

troisième niveau rajoute des informations sur les relations géométriques entre les axes d'entrée et de sortie et une équation de mouvement mathématique décrivant le comportement des paramètres d'entrée et de sortie. La résolution d'un problème de conception de mécanisme est traitée en six étapes, qui permettent la décomposition du mécanisme requis en mécanismes de base existants, ainsi que la vérification des contraintes inhérentes aux mécanismes choisis lors de la décomposition. Après ces étapes, une liste d'assemblages possibles de mécanismes peut être exploitée pour de la simulation à l'aide du troisième niveau de représentation présenté plus précédemment.

2.1.3 Les éléments bibliographiques qui supportent la définition des architectures

L'architecture logique correspond à un assemblage d'AO qui sont des abstractions de systèmes réels. Une méthodologie doit permettre le passage des fonctions formatées exprimant le besoin à des alternatives d'architectures logiques répondant au besoin. La génération de l'architecture logique revient à la réalisation de deux tâches que l'on peut traiter de front ou distinguer. Il s'agit de la sélection des AO et de leur interconnexion. L'assemblage des AO peut être manuel, semi-manuel ou totalement automatisé à partir de données d'entrée judicieusement choisies.

Avant de pouvoir sélectionner des AO et, à fortiori, de pouvoir les connecter, il est nécessaire de les créer et de les trier par classes. PAILHÈS et collab. [2009] font le listing des méta-classes de composants existants. Ce travail est important si l'on souhaite travailler avec une méthodologie objet et profiter du concept d'héritage. Pour plus d'information sur l'héritage, merci de vous référer à cet écrit de SIMONS [2003]. Les classes mises en exergue sont les convertisseurs, les transmetteurs, les composants supports, les composants de liaison et les éléments de contrôle. Néanmoins, dans le présent manuscrit, les AO ne seront pas rangés par classes. Un rangement par classes permettrait uniquement de présenter la base de données de manière organisée, mais cela n'aurait aucun impact sur la résolution du problème de synthèse tel qu'il est défini et sera présenté par la suite.

Une possibilité pour créer l'architecture logique serait maintenant de faire une allocation directe de fonctions de base sur ces AO. Cela permettrait, d'après une spécification donnée, de chercher la correspondance avec les fonctions de base. Ensuite, chaque fonction de base étant liée à des AO, l'allocation serait directe. GERO [1990] affirme qu'un passage direct d'une fonction à une structure physique ne correspond pas à un travail de conception, mais à un simple choix sur catalogue. Ce choix sur catalogue peut faire penser aux méthodes de « case based reasoning ». Chaque prototype possible peut être rangé dans une base de données et indexé par les fonctions qu'il satisfait. Un prototype, d'après GERO [1990], est composé d'un ensemble de fonctions, de comportements, d'une structure et d'un ensemble d'équations qui forment les connaissances que l'on a du prototype et du contexte dans lequel il est utilisé. Un prototype, contrairement à une classe parente, dans la littérature de la programmation objet de SIMONS [2003], est un super conteneur qui contient plus d'information que ses instanciations. Cela permet de travailler à différents niveaux de granularité. Une classe fille est toujours plus détaillée que sa classe parente et donc, si l'on souhaite instancier un objet qui contient moins d'information que dans la définition de sa classe, il est nécessaire de volontairement laisser de côté les attributs qui ne nous intéressent pas. C'est exactement le même principe que pour le super conteneur de SIMONS [2003]. GERO [1990] expose différents types de conception, la conception

routinière, la conception innovante et la conception créative. La conception routinière est basée sur la réutilisation des prototypes dans leur mode de fonctionnement normal. La conception innovante est aussi basée sur la réutilisation des prototypes, mais dans un mode de fonctionnement inhabituel. La conception créative correspond à l'ajout de nouveaux prototypes ou la modification des prototypes habituels. Cette analyse ne semble pas prendre en compte les configurations innovantes, mais uniquement des modifications internes aux objets qui composent un système conçu. Or, il nous semble qu'une architecture peut être considérée comme innovante si elle montre une topologie différente de celles habituellement observées comme le faisait déjà remarquer HENDERSON et CLARK [1990].

Une autre possibilité pour créer l'architecture logique serait d'allouer manuellement les fonctions émanant de la spécification aux différents AO de la base de données. C'est l'objectif de BRACEWELL et collab. [1992], qui offrent « une approche neutre technologiquement » qui se concentre principalement sur les échanges de flux d'information et de puissance électrique et mécanique, à l'aide de la méthode bond graph. Le logiciel n'est pas une solution automatisée de génération d'architectures, mais il permet aux architectes de construire le modèle à la main, en cherchant les composants dans une bibliothèque triée par les types de ports. Sans cette génération automatique de toutes les configurations architecturales possibles, il n'est pas possible d'affirmer que la solution choisie est la meilleure.

DAVID F. WYATT [2011] oppose deux méthodes de génération architecturale. D'un côté, les méthodes informelles du type «brainstorming», qui ne peuvent pas être informatisés, et de l'autre, les méthodes formelles qui sont souvent en partie automatisées. Avec les méthodes formelles, des étapes de formalisation, de synthèse et d'interprétation sont à rajouter aux méthodes informelles. Le but est de donner à l'algorithme de synthèse une entrée formelle qu'il puisse comprendre, et ensuite de traduire sa sortie formelle en configurations compréhensibles pour les architectes. Notons que BONEV et collab. [2015] partagent la même vision que DAVID F. WYATT [2011] concernant les méthodes formelles et informelles. DAVID F. WYATT [2011] propose une méthodologie pour la conception architecturale. L'espace de conception est délimité à l'aide de contraintes exprimées sur le graphe architectural. La méthodologie prend en entrée une architecture générale qui est développée pour finir sur plusieurs configurations instanciées. Il est très difficile d'appréhender à la main une telle architecture généralisée. Pour vérifier sa consistance, il est suffisant de vérifier que toutes les architectures possibles peuvent être dérivées de l'architecture généralisée. Le problème avec cette affirmation est qu'il faut connaitre à l'avance toutes les possibilités architecturales que l'on souhaite obtenir. Cela peut avoir un sens si l'on veut seulement effectuer une analyse automatique et un tri entre toutes les architectures que l'on connait déjà et que l'on souhaite étudier. Autrement, si on souhaite générer des architectures nouvelles, l'exhaustivité n'est pas assurée. Un autre problème avec le processus décrit par DAVID F. WYATT [2011] est que la phase de conception ne commence pas par une expression fonctionnelle du besoin, mais directement par une architecture technologique.

Dans le monde des méthodes de génération formelles, CAGAN et collab. [2005] font la liste de quelques méthodes pour l'émergence d'architectures concurrentes. La génération des configurations architecturales peut se faire à l'aide de techniques de recherche optimisées. C'est-à-dire qu'à chaque ajout de composant, le meilleur composant est sé-

lectionné. Le problème de cette technique est qu'il n'est pas acquis que tous les meilleurs composants mis ensemble peuvent fournir la meilleure solution. De même, la sélection d'un meilleur composant se fait par rapport à un unique critère. Il est possible que, pour le même composant, un autre critère soit très défavorable et cause un comportement global hors de la spécification. Une synthèse à partir d'un arbre de décision permet d'obtenir toutes les solutions possibles à un problème. Le point faible de cette méthode est qu'il faut attendre d'atteindre la dernière feuille de l'arbre avant de pouvoir comparer deux solutions voisines. Une autre possibilité est de mettre en place des systèmes experts qui vont prendre des décisions à la manière des humains en fonction de la situation.
CORTELLESSA et FRITTELLA [2007], dans cet état d'esprit, introduisent le concept d'«antipattern», en opposition au concept de «design pattern». Un «antipattern» est un assemblage de composants architecturaux qui n'est pas souhaitable dans une architecture en opposition au «design pattern», qui est un assemblage habituellement mis en place dans l'état de l'art. L'utilisation de ce type d'antipattern permet de ne pas contraindre une architecture avec des solutions éprouvées, mais d'assurer que les mauvaises conceptions passées ne se reproduisent plus.

Autrement, MAKKONEN et PERSSON [1994] proposent une méthodologie de synthèse à l'aide de la théorie des graphes et d'une grammaire. On peut parler de langage dans le cadre de la synthèse architecturale comme dans le cadre des langages humains. Les objets techniques élémentaires sont l'équivalent des mots, qui peuvent former des phrases, qui peuvent former des textes. De la même façon on peut former des chaînes à partir d'objets techniques, des arbres à partir des chaînes et aussi des forêts depuis les arbres. Ces assemblages d'objets techniques sont réalisés à partir d'un ensemble de règles fixes.

Une possibilité séduisante est de traiter les problèmes de synthèse à l'aide de contraintes qui pourraient être traduites mathématiquement. GADEYNE et collab. [2014] décrivent un espace de conception d'une synthèse assistée par ordinateur à l'aide de SysML et de contraintes écrites en Langage de Contraintes Objet, ou *Object Constraint Language* (OCL). Le modèle contient un objet «Système à concevoir» formé d'une agrégation d'objets «Composant». Les contraintes peuvent être de type architectural, avec des contraintes topologiques qui limitent les connexions entre composants et des contraintes dites mixtes car elles s'appliquent non pas à un composant, mais à l'assemblage, comme les contraintes d'interférences géométriques. D'autres contraintes prises en compte dans le modèle sont les règles de conception, ainsi que des contraintes dites statiques qui ne sont pas fonction de l'instanciation du système à concevoir. Par exemple, la taille globale du système peut être spécifiée comme une contrainte statique. Le modèle décrit dans ce papier permet aussi de définir des contraintes par scénario d'utilisation. Par exemple il est possible de spécifier une consommation de carburant différente pour chaque scénario. La dernière pièce du modèle présentée est la fonction objectif, sur laquelle sera joué l'algorithme d'optimisation. Le modèle ainsi décrit semble cohérent, mais il y manque une vue fonctionnelle. En effet, les contraintes décrites sont bien des exigences de performance ou de géométrie ou encore de pures contraintes, mais il n'y a pas de trace d'exigences fonctionnelles. Les composants étudiés sont uniquement de type mécanique car l'exemple porte sur des boîtes de transmission simples et non instrumentées. De plus, même si le modèle semble cohérent, il n'est pas supporté par un algorithme de synthèse et il manque l'exigence fonctionnelle pour balayer tous les aspects de l'espace des solutions. Dans ce manuscrit, nous décrirons des modèles dans le même état d'esprit que ceux présentés dans ce papier, mais aussi une méthodologie outillée qui permettra de prouver la véracité de nos modèles.

AL-HAKIM et collab. [2000] proposent une méthode de modélisation de systèmes mécaniques dans différentes configurations à l'aide de graphes logiques. Les boîtes de transmission à plusieurs rapports sont prises en exemple dans la publication. La représentation présentée n'est pas adaptée à des systèmes mécatroniques car il n'y a qu'un seul type de flux possible entre deux objets. Or, un objet complexe possède différents types de ports et peut être connecté plusieurs fois. Les solutions logiques présentées sont uniquement des graphes Entrées uniques Sorties uniques, ou *Single Inputs Single Outputs* (SISO) et pas des graphes MIMO, requis dans la méthodologie présentée dans ce manuscrit. Néanmoins, la représentation de plusieurs configurations permet d'analyser une phase de « pause » du système ou de déconnexion d'un sous-système du reste du système. Il semble important, et même indispensable, de permettre l'étude des différents situations de vie du système à concevoir dans une méthodologie de synthèse architecturale. En effet, le comportement attendu du système est différent à chaque situation de vie et cela a un impact sur la sélection des sous-systèmes qui vont le composer.

VAN BEEK et collab. [2010] utilisent conjointement une décomposition FBS (Umeda) de UMEDA et collab. [1996] et une matrice DSM dans le but de faciliter la détection d'interactions entre des composants d'un même système et de définir des sous-systèmes. Faire cet exercice à la main est souvent difficile, voire impossible, car le risque est fort d'introduire des erreurs lors d'un remplissage manuel d'une matrice de grande taille. Repartir de la décomposition fonctionnelle permet de démarrer d'un état de modélisation avec peu d'éléments et donc de mieux gérer la complexité croissante lors de la décomposition du problème.

Dans le monde des méthodes informelles, des outils ont été imaginés pour guider les architectes. L'objectif est alors d'analyser l'existant pour proposer des solutions à divers problèmes de conception. YANG et collab. [2015] proposent un travail détaillé concernant la création d'une méthodologie d'aide à la conception basée sur la réutilisation de connaissances sur les fonctions, les comportements et la structure de systèmes existants. Trois processus de conception reprennent treize flux de passage d'un espace à un autre. Il s'agit de la variation d'une conception, de l'adaptation d'une conception ou d'une toute nouvelle conception. Aucune méthodologie de synthèse automatisée n'est proposée, il ne s'agit que d'un système d'aide aux idées. La représentation des modèles est un mélange de la méthode IDEF0 du NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY [1993] et de la représentation boîte noire avec les entrées/sorties de type matière, énergie et signal définies par PAHL et collab. [2007].

Dans le même état d'esprit, en synthèse architecturale, il peut être possible d'utiliser la méthode résolution des problèmes inventifs, ou *Teoriya Resheniya Izobretatelskikh Zadach* (TRIZ) dans le but de résoudre des contradictions techniques. En effet, en partant d'une conception existante à améliorer, il est possible de trouver deux caractéristiques du système qui sont liées et semblent aller chacune dans une direction. C'est-à-dire que l'optimisation de l'une va dés-optimiser l'autre. La méthode TRIZ permet de surmonter les contradictions en se basant sur des solutions techniques mises en œuvre dans d'autres champs techniques, comme présenté par DEWULF et MANN [2014]. Ce principe ne sera pas du tout utilisé dans ce manuscrit, mais il présente un champ bibliographique important.

L'architecture physique correspond à une instanciation de l'architecture logique. Chaque

AO répond à des fonctions, réalise des échanges de flux à l'aide de lois déterminées et possède des attributs qui peuvent être calculés. L'objectif de l'instanciation est de déterminer toutes les valeurs des attributs, à l'exception des paramètres géométriques. Après avoir généré différentes solutions, une analyse peut permettre de guider les architectes dans leurs choix ou bien de guider le processus de génération. L'analyse des solutions générées est une étape délicate d'après CAGAN et collab. [2005]. En effet, une analyse complète de chaque scénario peut prendre plusieurs milliers d'heures. De même, lors de l'analyse, le meilleur comportement ne repose pas forcément sur un seul objectif. Dans ce cas, il faut gérer une comparaison des configurations architecturales sur plusieurs objectifs.

Dans la littérature actuelle, le terme «espace de conception» est utilisé pour des concepts de granularités différentes. L'espace de conception complet doit être composé de plusieurs modules indispensables : un module qui regroupe les besoins et les contraintes client, un module de modélisation fonctionnelle, un module de synthèse d'architectures logiques, un module d'analyse et d'optimisation et un module de génération de modèles géométriques. Néanmoins, on peut trouver dans la littérature des travaux qui considèrent l'espace de conception comme l'espace des valeurs qui peuvent être prises par les paramètres clés de la conception. Avec cette vision des choses, on ne réalise pas une synthèse sur les configurations possibles, mais une synthèse sur les instances d'une seule configuration, comme présenté par GANE et HAYMAKER [2012]. Comme exemple, le lecteur peut se rapporter au logiciel ModelCenter® de la société PHOENIX® INTEGRATION, utilisé pour de l'optimisation sur une architecture de drone quad-rotor par BEN MESSAOUD et collab. [2014].

LO [2013] présente différentes méthodes pour trier différentes architectures candidates sur plusieurs critères à la fois. Une de ses contributions est la création d'un outil qui vérifie la traçabilité des critères d'évaluation et qui va donc permettre de détecter si deux critères sont liés, c'est-à-dire s'ils émanent d'une même exigence ou d'une même fonction.

L'architecture géométrique correspond à une instanciation de l'architecture physique et, à fortiori, d'une architecture topologique. Il s'agit d'une représentation souvent simplifiée du système à concevoir, qui permet de visualiser et modifier le placement dans l'espace des objets architecturaux.

L'étape de cotation fonctionnelle se fait souvent une fois la pièce conçue sur des plans en 2D, sans prise en compte des contraintes additionnelles liées aux interfaces. Dans la méthodologie de BALLU et collab. [2006], les informations sur le tolérancement arrivent le plus tôt possible lors de la décomposition structurelle, puis sur le squelette et ensuite sur les volumes de l'étape de conception détaillée. Si une synthèse architecturale allait jusqu'à la définition des interfaces sur un modèle géométrique, des cotes fonctionnelles pourraient y être associées. De même sur une architecture logique, suivant le type de port mis en jeu, des contraintes sur la cotation pourraient être introduites.

DEMOLY [2010] et DEMOLY et collab. [2011] présentent une méthode de conception orientée assemblage. Dans les travaux liés à cette thèse, un chapitre retient principalement notre attention. Ce dernier permet la génération d'un squelette d'assemblage à partir d'un graphe d'assemblage. Cette méthodologie semble intéressante pour gérer le passage d'une représentation topologique à une représentation 3D filaire. En effet, dans les processus de synthèse, il est important de comprendre que, dans la littérature, deux visions existent : la

synthèse architecturale qui parcourt l'espace des solutions en faisant varier des variables de l'architecture, comme présenté par CHENOUARD [2007], et la synthèse architecturale qui crée des concepts différents. Dans le premier cas, le squelette géométrique reste le même dans sa forme, et seules ses grandeurs varient entre deux solutions. Dans le second cas, les squelettes géométriques ont des formes différentes. L'apport de DEMOLY et collab. [2011] est très important pour nos propres travaux car cela permettrait de créer des squelettes différents pour chaque architecture logique.
Notons que GADEYNE et collab. [2014] présentent une méthodologie pour la prise en compte des contraintes géométriques lors de la phase de conception architecturale. Ces contraintes sont exprimées à partir du langage OCL et accrochées dans une représentation SysML.

2.1.4 Des processus de synthèse de l'expression des besoins à la sélection des solutions architecturales

La plupart des groupes de recherche travaillant sur la synthèse en conception essayent de proposer des processus complets offrant dans un même environnement des visions multiples du système à concevoir. Le but est de prendre en compte les besoins des clients pour, au final, fournir une conception détaillée.

Des processus de synthèse manuels

Les processus de synthèse architecturale les plus courants restent ceux qui sont manuels et qui passent par une allocation des exigences aux sous-systèmes par des analyses manuelles. L'ingénierie système est une approche permettant la conception de systèmes complexes tout en assurant une complétude vis-à-vis des besoins et des contraintes induits par l'environnement. BAHILL et GISSING [1998] présentaient, il y a une vingtaine d'année, plusieurs processus se réclamant de l'ingénierie système. A la lecture de ce papier, et comparé aux processus décrits ensuite, le lecteur peut se rendre compte que les modèles et les processus utilisés se sont beaucoup complexifiés.

WEILKIENS [2006] présente une méthodologie d'ingénierie système supportée par SysML de la même sorte que le processus présenté ci-dessous, mais en moins accessible. L'Education Nationale, l'AFIS et l'aip primeca ont décrit un processus pour utiliser les concepts de l'ingénierie système avec SysML. Un système est défini comme un ensemble de composants interagissant et organisés dans le but de réaliser des objectifs prédéterminés (ISO 15288 :2015). Deux domaines sont définis : le domaine du problème, qui contient les besoins initiaux et les spécifications techniques et le domaine de la solution, qui contient l'architecture fonctionnelle et l'architecture physique.
La méthode de conception architecturale est définie par les actions suivantes :

- **Définition des besoins des parties prenantes**
  - Définition de la mission principale du système [Requirements Diagram]
  - Définition du contexte dans lequel le système va évoluer [Block Definition Diagram]
  - Définition des cas d'utilisation du système [Use Cases Diagram]
  - Description des scénarios d'utilisation [Use Cases Diagram]
  - Définition des besoins des parties prenantes [Requirements Diagram]

  - Vérification des besoins des parties prenantes [Needs/Needs matrix ; Needs/Use case matrix]
  - Validation des besoins des parties prenantes
  - Documentation des besoins des parties prenantes
- **Analyse des exigences**
  - Analyse du périmètre du système [Use Cases Diagram]
  - Définition des concepts du système [Block Definition Diagram]
  - Description des missions du système [State Machine Diagram ; Sequence Diagram]
  - Définition des exigences du système [Requirements Diagram]
  - Validation des exigences définies [Requirements Diagram]
  - Validation de la traçabilité sur les exigences du système [Requirements/Needs traceability matrix]
  - Vérification des exigences du système
  - Validation des exigences du système
  - Documentation des exigences du système
- **Conception de l'architecture**
  - Identification des opérations du système [Sequence Diagram]
  - Définition de la vue topologique du système [Block Definition Diagram]
  - Association des états aux opérations [State Machine Diagram]
  - Vérification de l'architecture topologique [Sequence Diagram ; Operations/System requirements matrix]
  - Analyse des architectures candidates [Block Definition Diagram]
  - Allocation des sous-systèmes aux opérations [Block Definition Diagram]
  - Définition des échanges entre les sous-systèmes [Sequence Diagram]
  - Définition des vues internes du système [Internal Block Diagram]
  - Vérification de l'architecture physique [Requirements Diagram ; Components/-Requirements]
  - Validation de l'architecture [Block Definition Diagram ; Internal Block Diagram ; Sequence Diagram ; State Machine Diagram ; Operations/Requirement matrix ; Components/Requirements matrix]
  - Documentation de l'architecture

Toutes les étapes présentées ci-dessus sont déjà définies par l'AFIS et elles peuvent être utilisées pour la construction et la représentation des architectures. Néanmoins, toutes les étapes définies dans ces méthodologies ne doivent pas forcément être utilisées lors de chaque conception d'architectures. Dans ce manuscrit, par exemple, le besoin des parties prenantes et les fonctions que le produit va devoir supporter sont déjà définis. Seulement quelques nouvelles fonctions seront ajoutées à la description fonctionnelle du produit à concevoir. Ce point de vue est partagé avec DENG et LU [2009]. C'est pour cela que nous devons garder à l'esprit le fameux heuristique KISS (Keep It Simple Stupid) de RECHTIN [1991], en essayant de n'utiliser que les représentations qui sont utiles à notre méthodologie.

Dans la méthodologie de CESAMES de KROB [2014], un système est défini par un ensemble d'entrées, de sorties et d'états internes. Les entrées sont fonction du temps, les états sont fonction du temps et des entrées et les sorties sont fonction du temps, des entrées et des états du système. Cette méthodologie CESAMES décrit trois visions architecturales qui sont : la vision opérationnelle (de l'extérieur au système), la vision fonctionnelle et la vision organique ou physique. La dynamique associée à cette méthodologie est illustrée à l'aide de la figure 2.4. Notons qu'une distinction est faite entre les fonctions et les exigences fonctionnelles. Une fonction est une simple combinaison d'un verbe d'action plus un nom, alors qu'une exigence fonctionnelle est une phrase permettant de spécifier la fonction. Cela veut dire que chaque fonction a pour attributs plusieurs exigences fonctionnelles, elles-mêmes dérivées des besoins exprimés par les parties prenantes.
Les visions opérationnelle et fonctionnelle permettent aux architectes la définition des besoins, des contraintes et du comportement attendu du système. En rajoutant la vision organique, ces trois visions sont obligatoires pour avoir une vue globale et complète du système, qui est souvent difficile à obtenir à cause de la composition des comportements. En effet, chaque composant possède ses propres comportements, mais un ensemble de systèmes interconnectés peut avoir un comportement différent de la somme des comportements de ses sous-systèmes. Notons que cette méthodologie est similaire à celle développée par Thalès au nom d'Arcadia. Cette méthodologie présentée par BONNET [2016], est supportée par l'outil Capella. Le grand avantage de cette méthodologie outillée est la possibilité de faire réellement du MBSE et de conserver les liens entre tous les objets. En effet, cette capacité est très importante lors de la spécification et de l'implémentation de systèmes complexes.

BEN HAMIDA et collab. [2015] présentent les différentes étapes que devrait posséder un cadre de synthèse. Ces étapes sont au nombre de sept, il s'agit :

— d'une étape de capture du besoin et de spécification,
— d'une étape de sélection des technologies disponibles,
— d'une étape de génération des architectures,
— d'une étape de sélection des alternatives,
— d'une étape d'exploration de l'espace de conception de chaque concept architectural,
— d'une étape de simulation des architectures pré-dimensionnées et
— d'une étape de définition des variantes possibles.

Il est a noter que quelques activités transverses sont aussi décrites, il s'agit de l'estimation des risques et des coûts, de la politique sur la prise de marges dans la conception, de la gestion des changements dans les objectifs et de la visualisation des informations.
La plupart de ces étapes sont en accord avec les préconisations de CAGAN et collab. [2005] sur la synthèse automatique. Néanmoins lors d'une synthèse assistée par ordinateur, le nombre de concepts architecturaux pouvant être analysés en simultané est bien supérieur au nombre de concepts habituellement sélectionnés par des concepteurs humains. En effet, d'après MILLER [1956], le cerveau humain peut gérer des informations rangées dans sept plus ou moins deux cases. Au maximum neuf concepts architecturaux pourraient être analysés sans une gestion informatisée.
Il est très intéressant de voir que dans le cadre de synthèse de BEN HAMIDA et collab. [2015] la phase d'exploration de l'espace de conception de chaque concept est découplée de la phase d'analyse. En effet, il est impossible de simuler un modèle comportementale sans que les variables d'état des AO aient été précédemment définies.

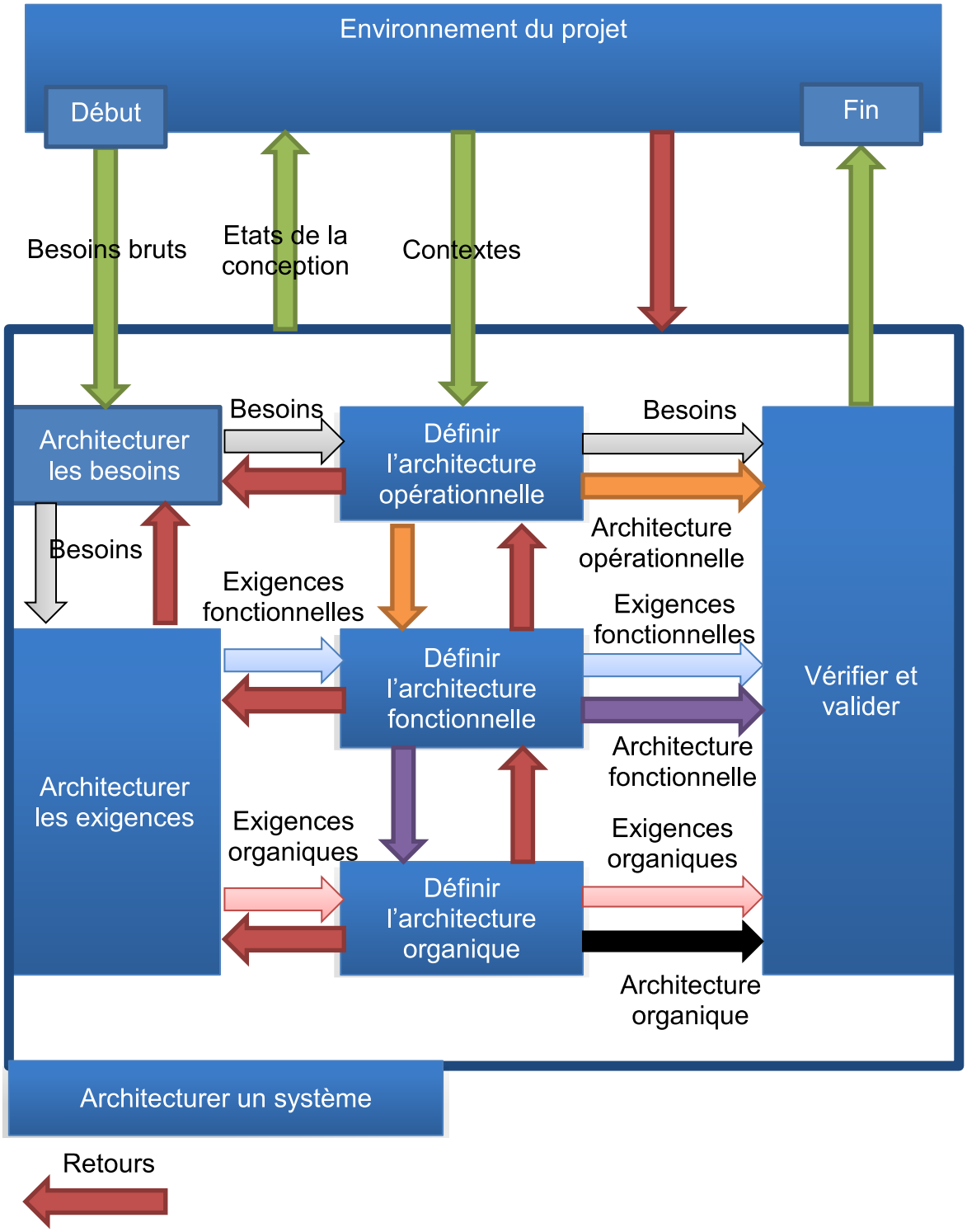


FIGURE 2.4 – Description de la dynamique de conception architecturale d'après KROB [2014].

Des processus de synthèse automatisés

Les processus de synthèse architecturale automatisés sont les plus proches de la méthode proposée dans la suite de ce manuscrit. Ils permettent de transformer plus ou moins automatiquement les exigences s'appliquant au système et l'allocation aux sous-systèmes. Les challenges que doivent surmonter ces processus sont souvent liés au temps de calcul nécessaire pour parcourir l'espace des solutions.

PEARCE et FRIEDENTHAL [2013] définissent un processus basé sur trois visions architecturales : l'architecture fonctionnelle, l'architecture logique et l'architecture physique. L'architecture fonctionnelle est une organisation de boîtes noires qui ne contiennent qu'une paire verbe-nom pour décrire l'opération effectuée. L'architecture logique raffine l'architecture fonctionnelle et introduit les performances requises en tant qu'attributs et les interfaces requises en tant que ports. Les architectures physiques sont les instanciations de l'architecture logique. Des composants sont sélectionnés dans la base des données et des valeurs sont appliquées aux propriétés. Dans ce dernier modèle, l'expression des comportements est laissée de côté intentionnellement dans le but de le simplifier et d'éviter un temps de calcul trop long. La traçabilité des besoins est aussi un point clé de la méthodologie et permet aux architectes d'être certains que tous les besoins sont remplis une seule fois.

SCARAVETTI [2004] présente une méthodologie pour la formulation d'un problème de conception. Cette démarche reste assez proche des méthodes habituelles de conception, mais permet en fin de processus une analyse systématique, à l'aide de la résolution d'un problème de satisfaction de contraintes, ou *Constraints Satisfaction Problem* (CSP), couplée à un tri sur les solutions. Les étapes mises en jeu sont : une analyse du besoin, une approche fonctionnelle, une approche organique et une approche physique, qui permettent de décrire les variables des différentes solutions. Le processus global de conception préliminaire décrit par SCARAVETTI [2004] reprend des concepts simples de l'ingénierie système. L'objectif derrière l'utilisation de ce processus est de trouver des contraintes physiques sur des variables de conception qui sont liées à des fonctions du système à concevoir. Les variables explicitées sont transformées en contraintes pour une résolution à l'aide d'un solveur CSP. Une fonction objectif est définie, puis, un tri sur les solutions fait ressortir les concepts optimaux qui devraient alors être sélectionnés par les concepteurs. La principale différence avec notre méthodologie est que nous souhaitons automatiser la génération des solutions et nous appuyer ensuite sur le CSP pour faire l'analyse de ces alternatives architecturales à la manière de SCARAVETTI [2004].

Le laboratoire du Prof. Chakrabarti (IISc – Inde) travaille, entre autres, sur un logiciel nommé FuncSION évalué par PAL et collab. [2014]. Avec la méthodologie développée dans ce logiciel, un problème de conception est présenté comme une transformation d'un ensemble de ports d'entrée en ports de sortie, comme cela est illustré sur la figure 2.5. Chacun de ces ports a les caractéristiques suivantes : son type de port, son orientation dans l'espace, son sens, sa position dans l'espace et sa magnitude. La résolution du problème de conception se fait en quatre étapes. Dans un premier temps, l'utilisateur définit ses fonctions à l'aide d'un ensemble de ports d'entrée à transformer en un ensemble de ports de sortie. Ensuite, une synthèse exhaustive permet la création d'un ensemble de solutions dites topologiques à l'aide de blocs fonctionnels. Troisièmement, une synthèse spatiale permet de transformer les solutions topologiques en des configurations respectant les contraintes spatiales énoncées dans les fonctions à satisfaire. Enfin, une synthèse physique permet d'instancier des sous-systèmes correspondant à chaque bloc fonctionnel. Les blocs

fonctionnels utilisés dans les étapes de synthèse sont créés à partir de l'analyse de systèmes existants. Etant donné qu'il y a trois étapes de synthèse dans FuncSION, il y a donc trois types de bloc de construction qui sont les blocs topologiques, spatiaux et physiques. Les blocs de construction au niveau topologique sont modélisés par un nom, un type du port d'entrée et un type du port de sortie. La différence avec notre modélisation est que nous autorisons l'architecte à définir des AO avec plusieurs types de ports. Un AO peut donc être connecté à plusieurs autres objets à la fois dans notre cas. Au niveau spatial, les notions de position, de direction et de sens sont ajoutées à la modélisation des solutions. Au niveau physique, des informations sur la géométrie, l'interface géométrique, le type de mouvement à l'interface, le comportement cinématique des blocs de constructions et le type du bloc (système élémentaire ou addition de systèmes) sont ajoutées.

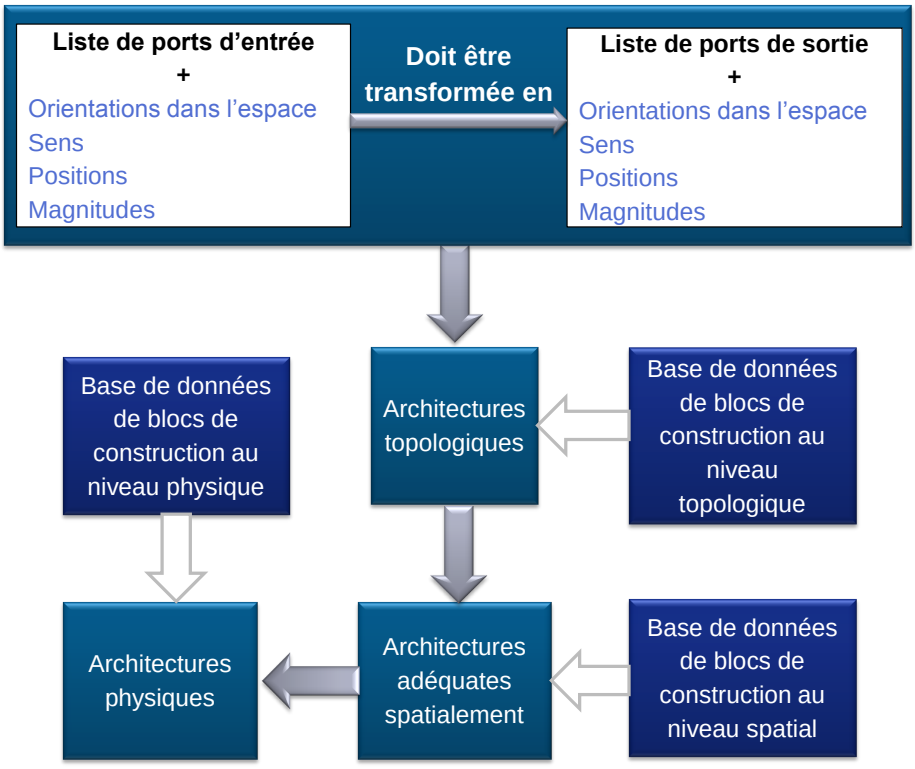


FIGURE 2.5 – Méthode FuncSion par PAL et collab. [2014].

Dr. Helms (TUM – Allemagne) et Prof. Shea (ETH Zurich - Suisse) ont travaillé sur la conception d'un logiciel de synthèse en conception utilisant le principe de la transformation de graphes à l'aide de règles prédéfinies, avec une vision orientée objet dans HELMS et SHEA [2012]. Un graphe est composé de nœuds et de liens. Dans l'exemple présenté sur la synthèse d'architectures de chaînes de puissance hybride pour l'automobile, les noeuds correspondent à des objets technologiques et les liens à des flux énergétiques, informationnels et matériels. Décrire l'architecture d'un système revient à expliciter les dépendances entre les différents sous-systèmes qui le composent. D'après les auteurs, ces dépendances entre objets peuvent être décrites comme un problème mathématique basé sur des contraintes. Ces contraintes représentent des conditions sur les variables des objets, qui doivent être remplies pour que l'architecture soit valide. Les auteurs nous font remarquer que, dans la littérature, deux types de contraintes peuvent être trouvés : les

contraintes paramétriques et les contraintes topologiques décrites dans DAVID F. WYATT [2011]. L'expression de contraintes topologiques sous forme de contraintes mathématiques est une tâche difficile d'après HELMS [2012]. L'utilisation de ports d'interconnexion permet une expression simplifiée des contraintes topologiques. Le concept de port introduit dans cette thèse permet de capitaliser les connaissances sur les différents types d'interfaces, sans tenir compte du niveau d'abstraction souhaité. Il permet aussi d'appliquer la notion d'héritage aux interfaces et, dans le cas de cette thèse, de considérablement diminuer le nombre de règles requises pour la génération des architectures. Les types des ports sont définis à partir de la taxonomie de HIRTZ et collab. [2002].
Dans la thèse de HELMS [2012], deux métamodèles sont présentés dans le but de décrire l'espace des solutions. Le premier métamodèle présenté contient les concepts de Fonction, de Comportement et de Structure, composés de ports. Le concept Structure contient trois arguments : «réalise l'effet physique», «multiplicité» et «peut être utilisé en mode inversé». Le deuxième métamodèle présenté contient, en plus des concepts précédents, le concept de port abstrait.
Le premier métamodèle utilisé possède une vision orientée objet. Les nœuds et les liaisons du modèle instancié sont donc des instanciations des classes nœud et liaison. Il est à noter que des classes abstraites sont définies dans le but de grouper des éléments sous un même type. Pour rappel, une classe abstraite est une classe qui ne peut être instanciée, mais dont des classes filles peuvent en hériter. La plus-value de cette méthode d'abstraction est de posséder des règles qui sont indépendantes du moyen de génération de la solution et du domaine d'application. Le premier métamodèle utilisé s'appuie sur différents niveaux d'abstraction, repris de GERO [1990], qui sont le niveau fonctionnel, le niveau comportemental et le niveau structurel. La définition des fonctions est reprise de PAHL et collab. [2007], et correspond à une description par une paire verbe plus nom et par des ports d'entrée et de sortie. La définition des comportements se fait, dans un premier temps, à l'aide d'équations comportementales, puis, dans un deuxième temps, des ports d'abstraction sont assignés aux effets physiques dans le but de réaliser la liaison entre les fonctions et les comportements. HELMS [2012] déclare que s'il n'y a pas de création de nouveaux composants, il n'est pas nécessaire de passer par le niveau d'abstraction comportemental. Dans la thèse de HELMS [2012], la représentation géométrique de l'architecture n'est pas abordée car l'auteur considère qu'à ce stade du processus de conception, la géométrie joue un rôle mineur. C'est principalement le métamodèle qui contient les spécificités du domaine d'application étudié lors de la synthèse. Les règles s'appliquent sur le concept de port, qui reste le même quel que soit le domaine d'application. La synthèse se fait à l'aide d'une méthodologie de transformation de graphes. Cela revient à appliquer successivement des règles de transformation à un graphe conventionnellement positionné à gauche, pour obtenir un graphe transformé, positionné à droite. Chaque règle contient des paramètres d'application, des conditions négatives d'application et les paramètres transformés à la fin de l'application de la règle. Les conditions négatives d'application correspondent à la recherche de motifs dans le graphe initial de gauche qui annuleraient l'application d'une règle. Le processus de synthèse automatisé commence avec un modèle fonctionnel composé de fonctions de haut niveau. Ce graphe de fonctions de haut niveau est transformé, à l'aide de la règle «initialisation», soit en un problème de décomposition fonctionnelle, soit en un problème d'allocation d'effets physiques, suivant le type de l'élément du graphe de gauche. Soit une sous-fonction est créée, soit un effet physique est créé. Les paramètres de cette règle permettent de décider si l'élément du graphe de gauche doit être décomposé ou instancié par un élément ayant les mêmes ports d'entrée et de sortie ou non. La règle de «création de chaîne» s'applique ensuite et ajoute un nouvel élément autour de l'élément

qui avait été ajouté au graphe lors de l'application de la règle précédente d'«initialisation». Cette règle ne s'applique que si les ports d'entrée ou de sortie de l'élément précédemment ajouté ne correspondent pas aux ports de la sous-fonction, ou de l'effet physique suivant le cas. Les paramètres de cette règle permettent de définir si l'élément à ajouter doit être ajouté à l'entrée ou à la sortie de la chaîne en cours de construction et au bout de combien d'itérations de chaînage un élément peut être introduit une fois de plus dans la chaîne. Certains éléments peuvent être écartés de la génération quand ils ont déjà été utilisés. La règle «concrétisation» permet une transformation du graphe des effets physiques en un graphe de composants à l'aide de l'attribut «réalise», interne aux composants. Cet attribut permet l'instanciation d'un principe physique ou d'un ensemble de principes physiques. Il est possible, lors de l'instanciation, de voir apparaitre des comportements non voulus qui sont contenus dans l'attribut «réalise». La règle « combinaison » permet la recherche de composants présents plusieurs fois dans l'architecture. Le paramètre «multiple» d'un composant indique le nombre de composants du même type qui peuvent être présents dans l'architecture. L'application des règles définit le processus de synthèse et la

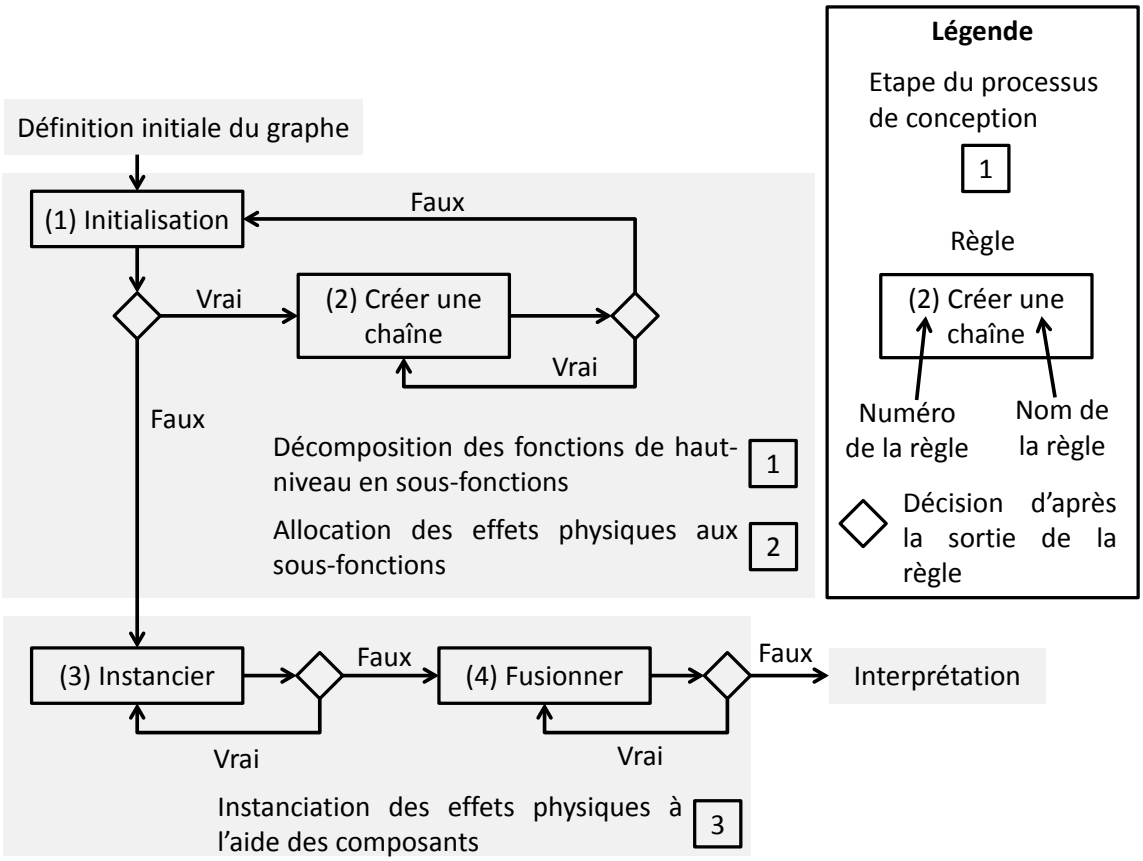


FIGURE 2.6 – La séquence de règles pour la synthèse architecturale à l'aide d'une décomposition FBS (Gero) d'après HELMS [2012].

génération de différentes solutions provient de la sélection aléatoire d'éléments lors de l'application des règles « initialisation » et « création de chaîne », comme illustré sur la Figure 2.6. Ce type de génération basée sur une recherche aléatoire est décrit comme naïve par CAGAN et collab. [2005], mais permettra forcément la découverte d'un nombre fini de solutions si l'espace de recherche est lui-même fini et si la répétition des composants dans une même chaîne est maîtrisée. L'auteur note qu'une fois que toutes les solutions d'un problème donné sont trouvées, la seule façon d'obtenir plus ou moins de solutions

est de modifier le métamodèle, étant donné que l'ensemble de règles est fixe. De même, il remarque qu'une augmentation raisonnée de la taille du métamodèle peut conduire à une forte augmentation de l'espace des solutions. Le premier métamodèle permet une génération des architectures uniquement sur la base des échanges de flux qui existe entre les différents effets physiques pour instancier une architecture fonctionnelle donnée.
Le second métamodèle permet, grâce aux ports abstraits, de relier directement les fonctions aux effets physiques et de réduire alors l'espace des solutions. Les différents types de ports d'abstraction sont calqués sur les composants de la méthodologie bond graph présentée dans DAUPHIN-TANGUY [2010]. La procédure pour relier les ports d'abstraction aux fonctions commence par la détermination de toutes les fonctions qui ne peuvent pas être représentées facilement par la méthodologie bond graph, ces fonctions ne sont pas traitées. Puis, manuellement, chaque fonction restante est analysée suivant la méthodologie bond-graph et des ports d'abstraction lui sont associés. Ensuite, les ports abstraits sont assignés aux effets physiques par l'analyse des équations caractéristiques de ces derniers. Enfin, les fonctions sont instanciées en effets physiques à l'aide des ports d'abstraction qui ont été introduits. Si un port d'abstraction n'est pas connecté lors de l'instanciation des fonctions, cela revient à l'introduction d'un effet indésirable dans l'architecture en cours de conception. Notons que dans les exemples développés dans la thèse, les ports d'abstraction ne sont pas utilisés pour faire la génération. Le concept est seulement introduit formellement, mais n'est pas appliqué. Les ports sont utilisés pour formuler des contraintes topologiques et peuvent imposer des contraintes de multiplicité. La résolution de ces contraintes nécessite, d'après HELMS [2012], l'utilisation d'un solveur de contraintes, mais le travail développé dans sa thèse n'utilise pas ce type d'algorithme. Le travail de la thèse de HELMS [2012] porte sur la modélisation, la synthèse et l'évaluation de topologies, sans prendre en compte les paramètres physiques des composants des architectures synthétisées. De ce fait, tous les composants de modulation ou d'adaptation ne sont pas utilisés lors de la synthèse. Les évolutions prévues après cette thèse ne remettent pas en cause l'utilisation des transformations de graphes. Les CSP ne seraient pas utilisées pour l'étape de synthèse, mais uniquement lors de l'étape d'analyse. Un logiciel nommé booggie a été développé pour mettre en œuvre la méthodologie développée. Ce logiciel utilise GrGen.NET [1] pour transformer les graphes de manière optimisée et Tulip [2] pour l'affichage.

A la suite des travaux conduis dans HELMS [2012], MÜNZER et collab. [2013] reprennent la résolution du problème de conception à l'aide de la méthode de satisfaction des problèmes booléens. Comme HELMS [2012], les auteurs utilisent le concept de port, défini dans plusieurs publications et ouvrages dès la fin du XXème siècle par ULRICH et SEERING [1989] et PAHL et collab. [2007]. Les ports peuvent représenter des flux ou des relations plus abstraites. MÜNZER et collab. [2013] prennent pour exemple un port qui ne représente rien de physique et qui permettrait de connecter un objet A à un objet B dans le but de répondre à un besoin ne répondant pas à un échange de flux. Les entrées du processus de synthèse automatique sont une base de données sur les objets technologiques (Métamodèle sur Figure 2.7) et un ensemble de ports à connecter (Graphe de départ sur Figure 2.7). Ces entrées sont modifiées pour coller au besoin réel de la synthèse dans le but de réduire la taille du problème de conception. Pour ce faire, le métamodèle est réduit en supprimant les composants que l'on ne veut pas voir apparaître dans la synthèse (Métamodèle réduit sur Figure 2.7). De la même manière, certains ports définis dans le métamodèle complet ne sont pas forcément utiles pour la synthèse attendue. Par exemple, il est possible de

1. http://www.info.uni-karlsruhe.de/software/grgen/ dernier accès le 11/10/2017
2. http://tulip.labri.fr/TulipDrupal/ dernier accès le 11/10/2017

supprimer des ports donnant des indications sur la géométrie des connexions si seule une synthèse topologique est attendue (Ports disponibles sur Figure 2.7). L'ensemble des ports à connecter devient le problème à résoudre (Problème à résoudre sur Figure 2.7). Des équations logiques à vérifier sont ensuite développées dans le but de synthétiser

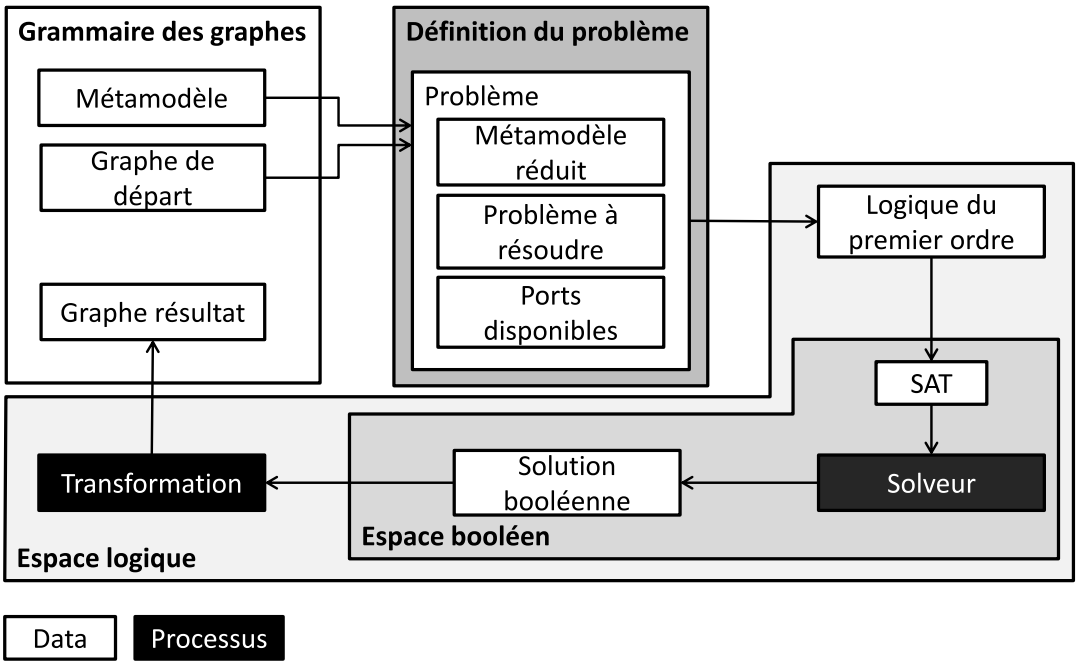


FIGURE 2.7 – Vue d'ensemble de la méthode développée par MÜNZER et collab. [2013].

tous les assemblages possibles entre les composants, à l'aide de deux prédicats logiques, l'un vérifiant l'appartenance du port à un élément et l'autre vérifiant que deux ports ou deux éléments sont identiques. Ces équations logiques peuvent être exprimées en langage courant :

— Tous les éléments doivent faire partie du métamodèle réduit.

— Chaque élément est une instance de son super type.

— Chaque port n'appartient qu'à un seul élément.

— Tous les ports doivent être connectés.

— Deux ports peuvent être connectés s'ils sont de types et de directions compatibles et s'ils ne sont pas identiques.

— Le problème défini (Problème à résoudre sur Figure 2.7) doit être résolu.

Ces équations logiques sont traduites automatiquement pour être utilisées dans un solveur de problème de SATisfaisabilité booléenne, ou *boolean SATisfability problem* (SAT), renvoyant les matrices DSM représentatives des solutions issues de la synthèse. Ces matrices sont ensuite interprétées sous forme de graphes. Pour éviter la génération d'une infinité de solutions, la méthodologie développée s'applique sur des «cadres» prédéfinis qui limitent le nombre de composants et de ports dans les solutions générées. Cette obligation rend la

génération de toutes les solutions impossible.

Le courant initié par Dr. Komoto (AIST – Japon), Prof. Tomiyama (Cranfield University – Royaume-Unis) et Prof. Umeda (Osaka University – Japon) porte sur la création d'un logiciel de « System Architecting CAD ». Ce logiciel est composé d'un outil de décomposition fonctionnelle du type FBS (Umeda), explicité par UMEDA et collab. [1996], d'une représentation des variables interconnectées de l'architecture, d'un modèle comportemental et d'un modèle géométrique. Ce développement est décrit par KOMOTO et TOMIYAMA [2010b], KOMOTO et TOMIYAMA [2010a], KOMOTO et TOMIYAMA [2011] et KOMOTO et TOMIYAMA [2012] et illustré par la Figure 2.8. UMEDA et collab. [1996] distinguent les deux principales

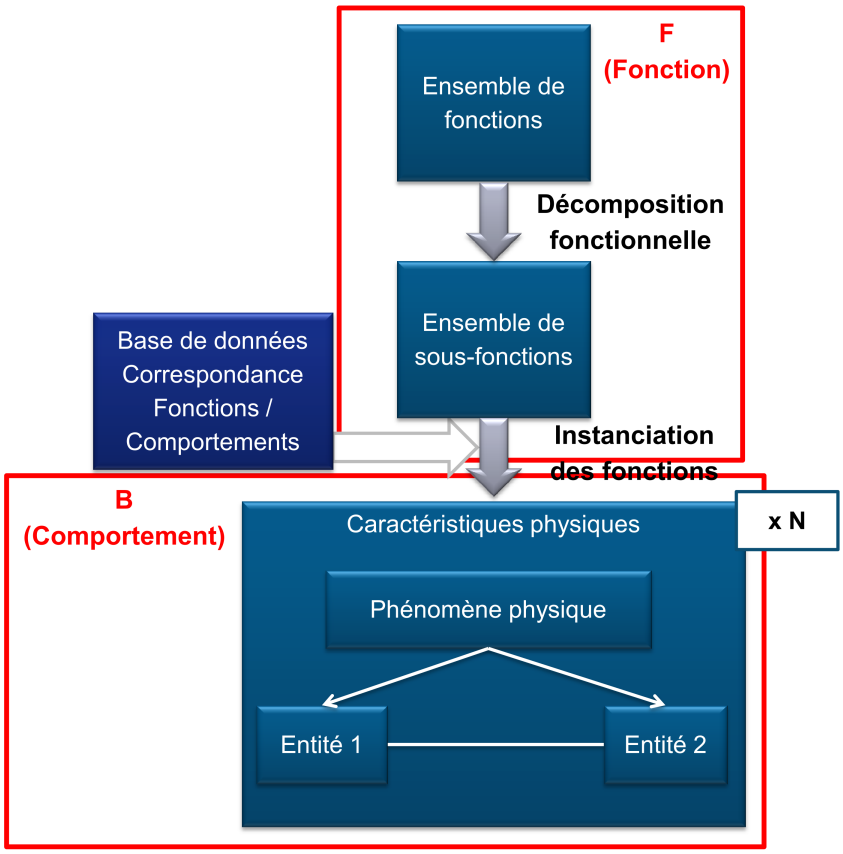


FIGURE 2.8 – Méthode SA-CAD par KOMOTO et TOMIYAMA [2012].

façons de décrire une fonction : la paire de ports entrée/sortie et la paire verbe plus nom. La première représentation n'est valable que pour des transformations de flux et la deuxième n'est pas assez précise et ne permet pas de décrire totalement un comportement voulu pour un système. C'est pour cette raison qu'un ensemble de comportements est ajouté à la paire verbe plus nom dans la méthodologie FBS (Umeda). La méthode de conception définie par ALVAREZ et collab. [2012], KOMOTO et TOMIYAMA [2011] et KOMOTO et TOMIYAMA [2010a] est composée de quatre visions formant le modèle global du système. Le logiciel conçu pour l'application de la méthode est composé de :

— Un modèle FBS (Umeda)
— Un modèle géométrique
— Un modèle du processus
— Un modèle du réseau de paramètres

Notons que ces travaux résultent de M ISHII [1996], qui ne contiennent pas de modèle géométrique, mais dont les premiers résultats ont déjà été obtenus avec *Qualitative Process Abduction System* (QPAS). Dans le modèle QPAS, pour activer un phénomène physique, tous les paramètres d'entrée du phénomène doivent être activés. Si un paramètre n'est pas connecté, QPAS cherche un nouveau composant qui possède le paramètre manquant. Le processus est illustré sur la Figure 2.9. Le modèle FBS est utilisé pour instancier le

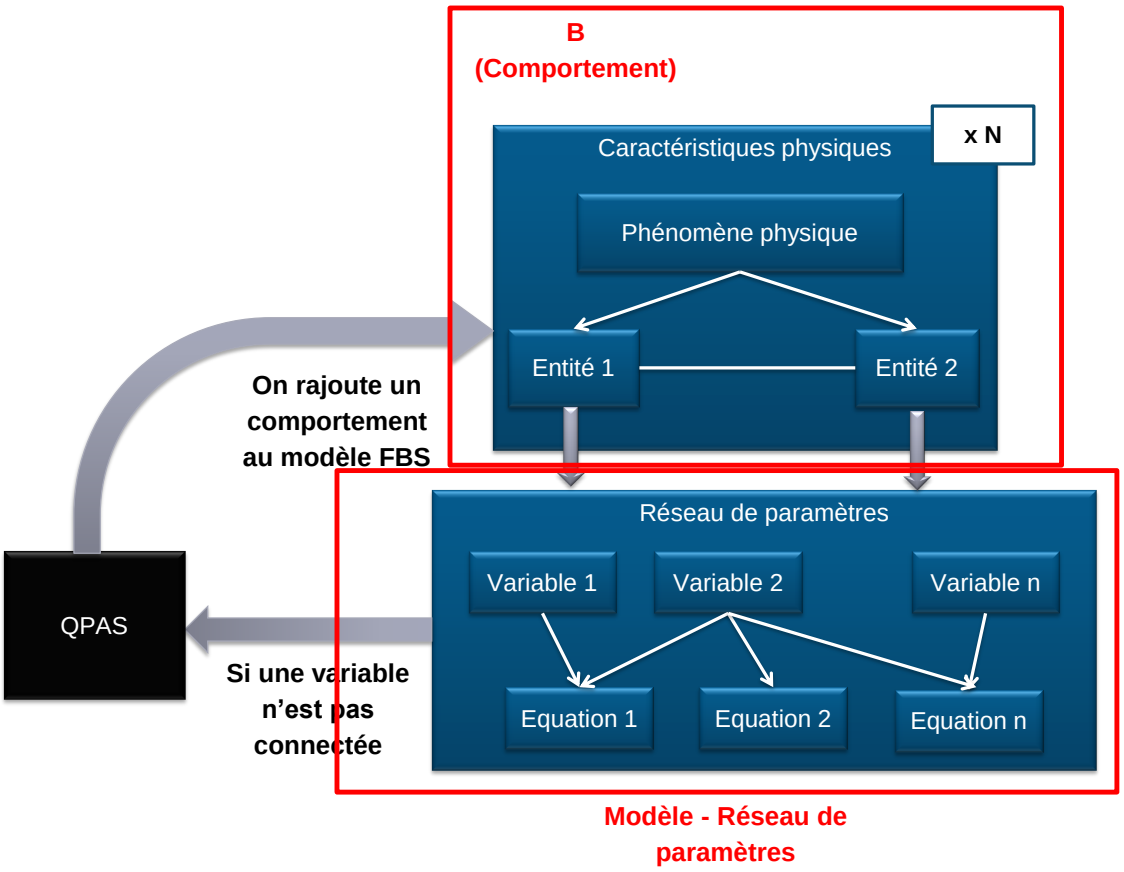


FIGURE 2.9 – Méthode QPAS par M ISHII [1996].

modèle fonctionnel en un modèle contenant les phénomènes physiques mis en jeux, les composants et leurs liaisons. Le modèle du processus décrit les relations temporelles entre les phénomènes physiques. Le modèle géométrique décrit les relations géométriques qu'il peut y avoir entre les différents phénomènes physiques. Le modèle du réseau de paramètres décrit les paramètres des composants et les relations entre les composants grâce aux relations entre leurs paramètres.
Le processus de conception architecturale démarre avec l'expression des besoins et quelques entités représentant l'environnement dans lequel le système va exister. Les besoins introduisent des paramètres qui sont représentatifs des entités présentes. Ces quelques actions sont réalisées manuellement par les architectes. Ensuite, le logiciel cherche et affiche tous les phénomènes physiques qui peuvent influencer les paramètres définis comme cibles pour cette analyse. Les architectes sélectionnent dans cette liste les phénomènes qui doivent être présents dans le modèle en cours de construction. Ces phénomènes physiques sélectionnés sont reliés à d'autres paramètres qui sont donc instanciés avec leurs relations. Si ces paramètres font partie de nouveaux composants, alors ils sont aussi instanciés. Un phénomène physique peut être instancié de différentes façons et donc plusieurs architectures apparaissent à ce moment.

Quand le modèle FBS (Umeda) est complet, les composants et les phénomènes physiques sont copiés dans le modèle géométrique et le modèle du processus. La géométrie des composants est définie, tout comme la temporalité entre les différents effets physiques. Ensuite le modèle du processus est transcrit en un code qui sera utilisé pour vérifier la consistance du système conçu. Quand le modèle est totalement représenté, une matrice DSM est utilisée pour représenter les relations entre composants et pour définir où se fera le découpage entre les sous-systèmes, dans le but de minimiser les échanges d'informations entre eux. KOMOTO et collab. [2014] introduisent l'utilisation de Modelica comme un outil d'analyse sur les architectures développées. Notons que le langage Modelica est utilisé pour de la simulation et non pas pour de l'instanciation car, les solveurs utilisés ont besoin de systèmes d'équations «carrés». Or dans une phase d'instanciation, les systèmes ne sont pas carrés puisque toutes les variables d'état ne sont pas encore sélectionnées. Néanmoins, certains travaux comme ceux de RENIER et CHENOUARD [2011], sous l'impulsion du groupe de travail de l'OMG sur la standardisation de la traduction SysML-Modelica utilisent des transformations M2M. En effet, PAREDIS et collab. [2010] construisent le profil SysML SysML4Modelica pour l'OMG. Dans leurs travaux, une comparaison entre les deux langages est faite et il en ressort que seul les diagrammes de blocs et de blocs internes de SysML ont le niveau de détail nécessaire pour une transformation M2M. Avec le même objectif, JOHNSON et collab. [2012] proposent une méthode de modélisation avec SysML qui permet ensuite de simuler les modèles sous Modelica.

Un autre axe de recherche concernant la synthèse architecturale est l'utilisation de réseaux bayésiens. Il s'agit de modèles probabilistes permettant le calcul de probabilités jointes et aboutissant à la découverte de toutes les possibilités d'instanciation de ses attributs. Dans ce cas les attributs du réseau bayésien sont les variables de conception du problème de synthèse architectural. Les travaux de MOULLEC et collab. [2013] utilisent les concepts des réseaux bayésiens pour de la synthèse architecturale. Quelques limitations sont à noter : l'architecture fonctionnelle est prédéterminée et tous les domaines de variation sont discrets. En plus de ces deux inconvénients il est à noter que l'utilisation directe de réseaux bayésiens n'est pas forcément intuitive pour des non-initiés et que la construction des modèles semble moins intuitive que ce qui se fait avec des outils utilisés en ingénierie système avec SysML et Modelica.

Des processus de transformation d'architectures existantes

Les processus de transformation partent habituellement d'architectures existantes et, par une application de règles, permettent leur transformation et le parcours d'une partie de l'espace des solutions architecturales. Ces processus s'inscrivent plus dans un cadre de modification d'architectures que dans un cadre de synthèse de nouvelles architectures.

KÖNIGSEDER et SHEA [2015] présentent une méthode pour synthétiser les solutions possibles liées à la modification d'un système initial valide ou non. La méthode repose sur deux points clés : l'algorithme « d'explosion » et les règles qu'il applique pour modifier le système initial lors de la recherche de nouvelles solutions. Ce papier propose une comparaison entre différentes stratégies pour l'application de règles topologiques et paramétriques lors de la synthèse. La topologie d'un système revient à décrire sa structuration, tandis que les paramètres décrivent ses caractéristiques physiques. Une règle topologique revient donc à une règle d'assemblage et une règle paramétrique à une règle de dimensionnement.

La méthodologie Mal'in de NADEAU et LEDOUX [2014] s'applique sur des architectures déjà définies. Mal'in forme des graphes qui permettent de visualiser les relations fonctionnelles entre les différents composants du système. Ces liens fonctionnels sont labellisés «utiles», «nuisibles» ou «insuffisants». Dans le cas de Mal'in, on va rechercher des contradictions sur une architecture déjà définie : contradiction physique, contradiction technique ...

L'approche GAIA, développée par FALGARONE et CHEVASSUS [2006], est une approche multi-vue qui permet d'étudier les liens de traçabilité présents dans un assemblage multi-métiers. Ces vues correspondent à :

— une décomposition de type FAST avec, au plus bas niveau, des alternatives sur les solutions technologiques qui concrétisent les fonctions,

— un regroupement des solutions technologiques sous la forme de systèmes et de sous-systèmes liés entre eux (Ces liens sont créés à l'aide du FAST. Si des fonctions sont liées, alors les solutions technologiques qui en découlent le sont aussi.),

— une vision géométrique des assemblages,

— une vision paramétrique où l'on peut relier des variables appartenant à des solutions technologiques pour visualiser directement l'impact d'un changement.

Aucune information n'est générée automatiquement, elles sont toutes rentrées par l'utilisateur, mais ensuite elles sont cascadées dans toutes les représentations. Cette approche est développée pour permettre une traçabilité sur les interfaces dans un assemblage mécanique. Une interface est définie par un couple de matières et des surfaces fonctionnelles avec des exigences sur le tolérancement. L'objectif de cette traçabilité est de permettre, entre autres, aux concepteurs de comprendre l'utilité des exigences de tolérancement qu'ils sont en train de mettre en place. Car, en effet, le tolérancement d'une surface fonctionnelle n'a aucun sens s'il n'est pas replacé dans le contexte d'un assemblage.

2.1.5 Conclusion du chapitre et questions de recherche

De nombreux travaux ont déjà été conduits par la communauté scientifique sur l'analyse du besoin et l'allocation des exigences aux sous-systèmes. Certaines méthodologies sont totalement manuelles, tandis que d'autres essayent d'aider l'utilisateur soit dans la décomposition des exigences, soit dans leurs allocations. Néanmoins, aucune méthodologie ou aucun logiciel n'est encore accepté par l'industrie. Est-ce parce que les méthodologies sont trop difficiles à appliquer ? Ou alors que les résultats ne sont pas assez probants ?

Au vu des références bibliographiques présentées dans ce chapitre, des questions de recherche émergent et des esquisses de réponses peuvent être données :

— **Question 1 :** Quels outils sont les plus aptes à l'extraction du besoin ?
Les méthodes développées par l'INCOSE, l'AFIS et l'aip primeca, ainsi que celle proposée par CESAMES pour l'extraction du besoin fonctionnel sont tout à fait satisfaisantes et éprouvées. Les concepts trouvés dans ces écrits seront donc réutilisés dans la partie de spécification des exigences de ce manuscrit.

— **Question 2 :** Comment formaliser les fonctions auxquelles le système doit répondre pour qu'un algorithme puisse les traiter ?
Même si différentes modélisations fonctionnelles existent, il semble que la modélisation d'une fonction sous la forme d'une boîte noire est celle qui se prête le plus à un traitement automatique. En effet, la transformation des fonctions textuelles

en fonctions interprétables par un ordinateur semble plus difficile à réaliser que la modélisation directe sous forme de boîtes noires.

— **Question 3 :** Quelle méthode utiliser pour interconnecter automatiquement des objets architecturaux technologiques ?
Des règles de grammaire, des chaînages de blocs fonctionnels et l'utilisation d'un solveur SAT pour résoudre des problèmes d'interconnexion ont été présentés dans cette bibliographie. La méthode qui semble la plus simple d'utilisation pour l'utilisateur final, c'est-à-dire l'architecte, est la résolution automatisée à l'aide de contraintes renseignées à l'avance dans un modèle d'interconnexion.

— **Question 4 :** Quels outils sont les plus adaptés à l'instanciation des architectures logiques et l'analyse des architectures physiques ?
Le langage Modelica, les bond graphs et les CSP ont été présentés succinctement. Etant donné la nature des équations qui vont être traitées dans ce manuscrit, l'utilisation de CSP semble plus indiquée pour du pré-dimensionnement.

— **Question 5 :** Comment évaluer les architectures concurrentes ?

— **Question 6 :** Comment représenter nos modèles architecturaux ?
SysML, UML et Modelica sont autant de langages qui peuvent être utilisés pour représenter d'une manière compréhensible les architectures qui doivent être générées par l'outil développé. Ces langages demandent néanmoins une certaine connaissance pour être interprétés. De plus les modèles architecturaux vont être générés automatiquement et le placement en 2D des blocs de construction encore appelés AO doit se faire automatiquement. Aucune plateforme ne permet de créer automatiquement des diagrammes avec ces trois langages en évitant la superposition des objets de modélisation graphiques. Des choix et un travail de mise en forme seront donc présentés dans le manuscrit pour adresser ce point.

L'objectif des travaux qui vont qui vont être décrits dans la suite du manuscrit est de transférer la complexité de la synthèse architecturale dans un processus automatisé, invisible pour l'utilisateur.

Tout au long de ce manuscrit, un exemple fil rouge sera considéré. Il s'agit d'un système de drone volant permettant de transporter des charges et de faire de la surveillance de zones à distance. La méthodologie présentée dans ce manuscrit sera illustrée de la capture des besoins des parties prenantes, à l'analyse des architectures physiques, en passant par la génération des architectures logiques.

Chapitre 3

Capture du besoin et expression des exigences

« *Essentially, all models are wrong, but some are useful.* »

George Box

Sommaire

3.1 Introduction du chapitre

Le présent chapitre a pour but de décrire précisément la méthode d'extraction du besoin et la spécification du système à concevoir. Les sujets qui y sont traités vont de l'identification des parties prenantes et des situations de vie, en passant par l'expression des besoins jusqu'à leur transformation en exigences.
Le challenge est d'exprimer la spécification du système à concevoir sous une forme interprétable, et par l'utilisateur de la méthodologie, et par un ordinateur qui fera la synthèse architecturale automatiquement.

Cette étape de la méthodologie de synthèse assistée par ordinateur est purement manuelle. Il s'agit d'une étape d'émergence et de transformation des données d'entrée de la génération automatique. L'objectif, ici, est de décrire le système à concevoir, dans un premier temps, de manière textuelle et enfin, dans un formalisme compréhensible par les humains et par la machine qui va faire la résolution du problème de synthèse. La vue globale de cette étape est illustrée sur la figure 3.1 et par HARTMANN et collab. [2017]. Les différentes étapes de modélisation et d'extraction du besoin sont reprises de méthodologies éprouvées dans l'industrie par Thalès avec la méthodologie présentée par BONNET [2016], par CESAMES fondée par KROB [2014] et par la NASA [2007]. Ces processus s'appliquent habituellement de manière récursive au système à concevoir puis à ses sous-systèmes, jusqu'à arriver à la spécification physique des composants. Le processus décrit dans ce manuscrit ne contiendra pas une telle récursivité car seul le système est spécifié avant la sélection des composants réalisée par la partie automatisée. En effet, une des difficultés lors des phases de conception amont est d'identifier des alternatives prometteuses. Les méthodologies présentées ci-dessus donnent un cadre d'émergence des solutions à l'aide de différents diagrammes guidant la réflexion de l'architecte. Néanmoins, toutes les solutions ne peuvent être détectées de manière systématique, car même si des modélisations opérationnelle, fonctionnelle et physique sont conduites, les architectes iront plus facilement vers des solutions éprouvées pour faire leur choix.
La modélisation opérationnelle permet de répondre à la question «pourquoi ?». C'est une modélisation qui considère le système comme une boîte noire et qui s'attaque à la représentation des échanges du système vers l'extérieur dans tous les cas opérationnels qu'il va rencontrer. Cette étape de modélisation ne doit pas faire apparaître de choix technologiques pour le système. La recherche exhaustive des parties prenantes est fondamentale pour éviter de contraindre le système à concevoir avec des choix technologiques induits. Un exemple est nécessaire pour bien comprendre le raisonnement. Imaginons la spécification d'une voiture, qui aurait comme partie prenante le réseau électrique, mais pas le réseau de stations essences. Dans ce cas, cette voiture, une fois conçue, serait forcément 100% électrique, car impossible de faire le plein, étant donné que la partie prenante «station essence» a été oubliée. L'exemple est volontairement gros, mais, dans la méthodologie, un oubli de ce type aurait ce type de conséquence.
A la fin du processus d'architecture, les différents diagrammes auront évolué et seules les parties prenantes réellement utilisées seront gardées. Toutes les parties prenantes non conservées devraient néanmoins être documentées pour comprendre pourquoi elles ont été retirées des diagrammes.
La modélisation fonctionnelle permet de répondre à la question «quoi ?». C'est une modélisation qui considère le système comme une boîte blanche et qui considère aussi les échanges de flux entre les fonctions et vers l'extérieur. Dans un premier temps, les fonctions doivent être libres de tout choix technologique, mais en les décomposant, certaines architectures alternatives émergent. Prenons l'exemple d'une fonction «Transformer de

l'énergie en énergie mécanique de rotation» qui est vague et de très haut niveau. Si cette fonction est raffinée en «Transformer une énergie électrique en énergie mécanique de rotation», cela oriente déjà vers des solutions. Dans une méthode classique de décomposition fonctionnelle, chaque branche doit être traitée jusqu'à être abandonnée pour des raisons à documenter. Dans la méthodologie présentée dans ce manuscrit, la décomposition fonctionnelle se doit de rester de haut niveau pour ne pas surspécifier le système. Néanmoins, si l'on sait que le système va s'insérer dans un environnement où seule de l'énergie électrique sera disponible, alors il faut forcément avoir une fonction «Transformer de l'énergie électrique en énergie mécanique de rotation».

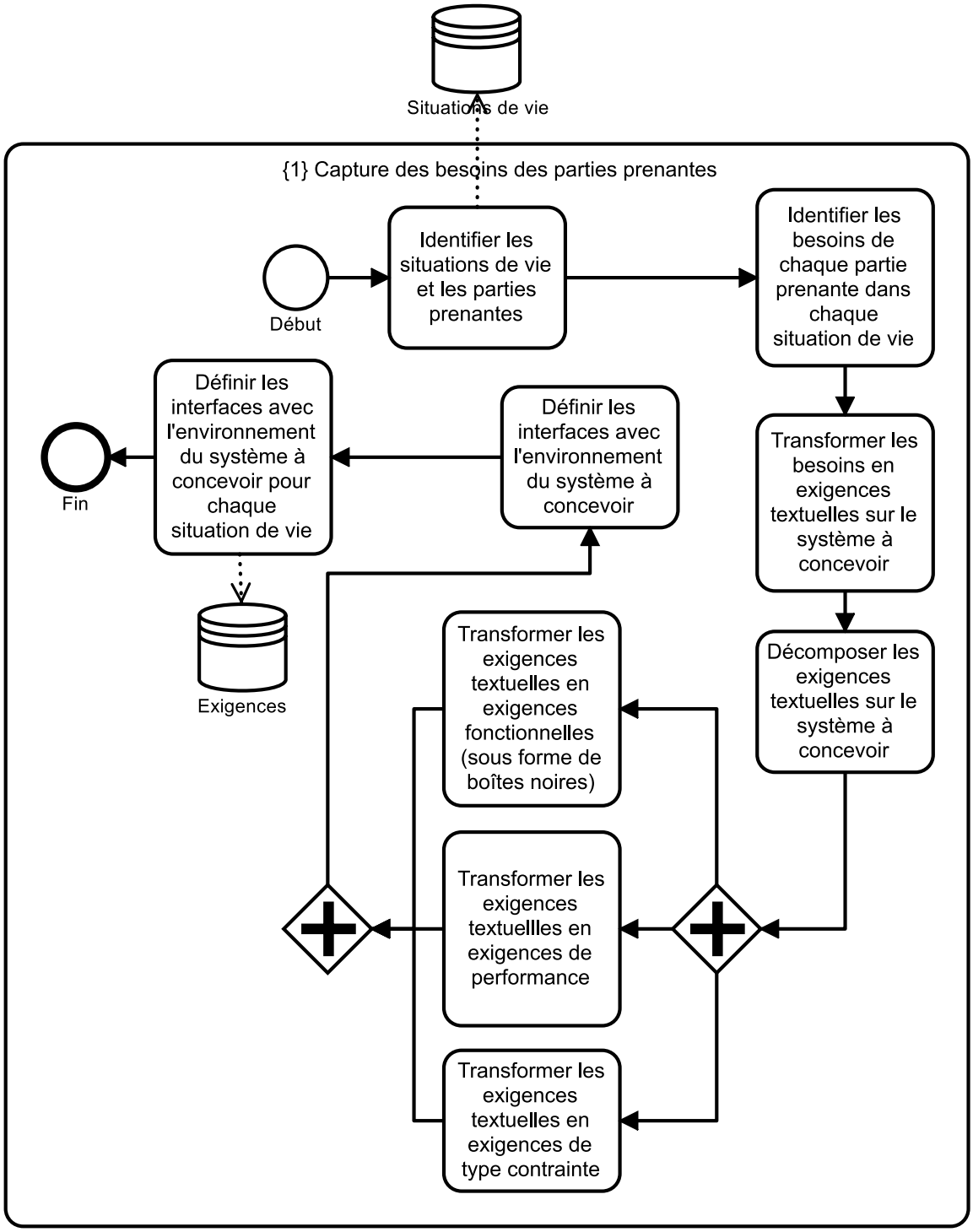


FIGURE 3.1 – Émergence et transformation des exigences.

3.2 Expression du besoin des parties prenantes

3.2.1 Identification des situations de vie et des parties prenantes

Dans un premier temps, l'objectif est de trouver toutes les parties prenantes dans toutes les situations de vie du système à concevoir. Pour ce faire, il est nécessaire de lister toutes les parties prenantes et les situations de vie qui viennent à l'esprit avant de les prendre une par une pour voir si une partie prenante n'introduit pas une nouvelle situation de vie et inversement. Cela revient à créer plusieurs diagrammes pieuvres de la méthodologie APTE ou des diagrammes d'environnement de la méthodologie CESAMES de KROB [2014]. Ces diagrammes sont utilisés pour étudier l'environnement du système à concevoir dans les différentes situations de vie pertinentes. Lors de la phase d'expression des besoins des parties prenantes, les fonctions du système ne sont pas encore définies. En effet, l'objectif est de capter le besoin pour ensuite pouvoir exprimer les fonctions et les exigences fonctionnelles ou non. Le diagramme pieuvre permet la représentation des parties prenantes avec la fonction principale ou la fonction contrainte qui la relie au système à concevoir. Le diagramme d'environnement permet la représentation des parties prenantes avec le flux qui le relie au système à concevoir. Il semble donc plus approprié d'utiliser des diagrammes d'environnement que des diagrammes pieuvres dans cette phase de la synthèse.

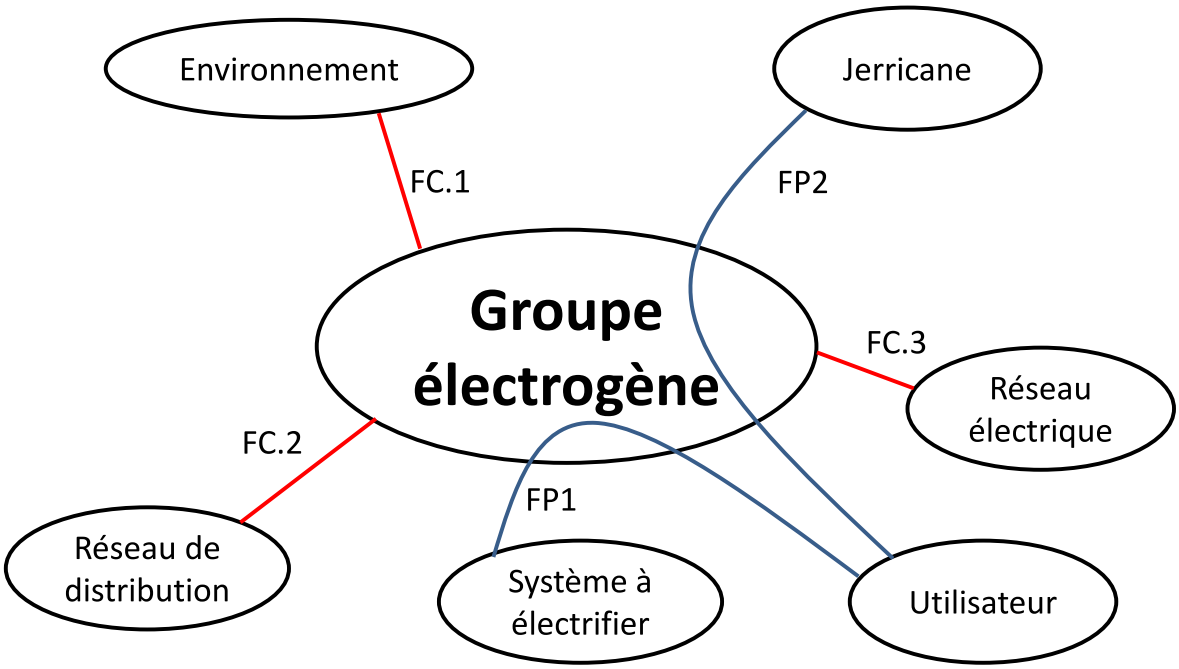


FIGURE 3.2 – Exemple d'un diagramme pieuvre de la méthodologie APTE.

Les premières listes des parties prenantes et des situations de vie se font directement sur une feuille blanche. L'utilisation, dans un second temps, d'un cycle de vie sur lequel sont représentés les passages d'une situation de vie à l'autre est une bonne façon de détecter des manquants. Ce cycle de vie peut être décrit avec le langage *Business Process Model and Notation* (BPMN) ou avec un diagramme d'états/transitions du langage SysML. Le BPMN est utilisé habituellement pour décrire l'enchaînement de processus métier, mais il peut aussi convenir pour décrire les cas opérationnels présents dans un cycle de vie, ainsi que les transitions qui permettent de passer d'une situation de vie à une autre.

La Figure 3.3 illustre l'utilisation des situations de vie dans la méthodologie proposée.

Un AO dispose de plusieurs situations de vie appelée mode de fonctionnement qui se modélise sous la forme d'un ensemble de ports actifs ou non. Cela permet, en prenant en compte tous les modes de fonctionnement de chaque AO, de vérifier que le système a concevoir pourra bien fonctionner dans toutes les situations de vie. A noter que la situation de vie est associée au système à concevoir alors que le mode de fonctionnement est associé à l'AO aussi appelé le sous-système du système à concevoir. Ces définitions sont identiques mais elles permettent de différencier si l'on parle du système à concevoir ou des AO qui le composent.

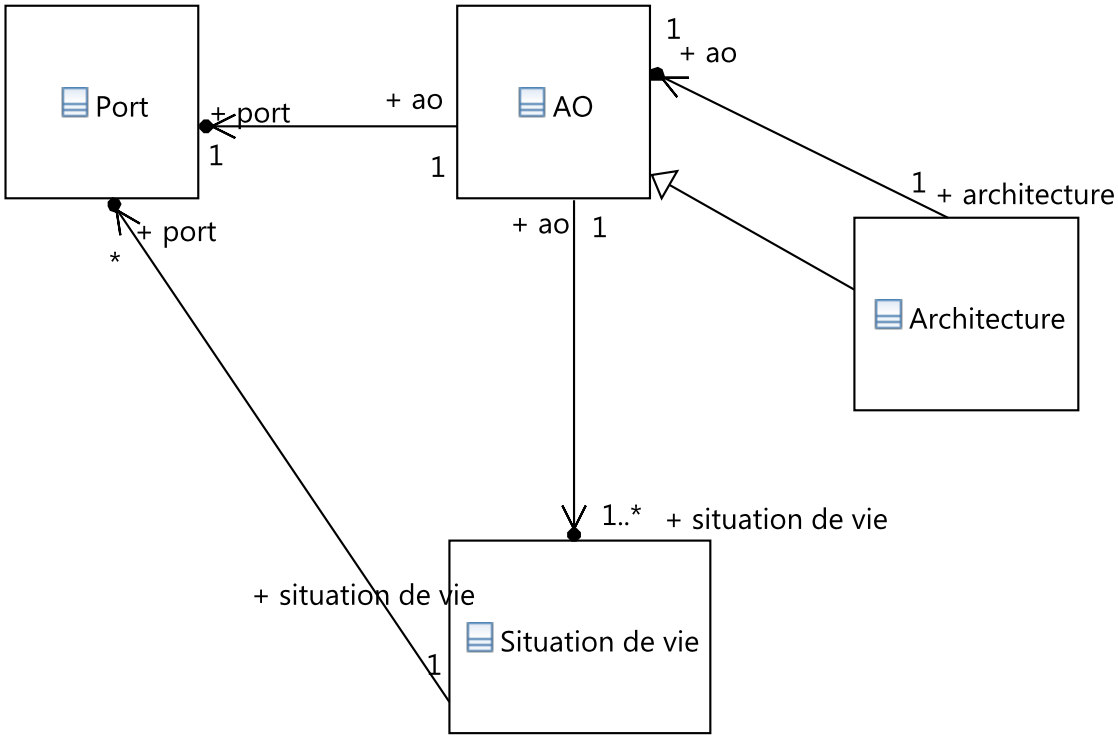


FIGURE 3.3 – Le méta-modèle des situations de vie.

3.2.2 Expression du besoin

Une fois que toutes les parties prenantes et les situations de vie sont décrites, il est nécessaire d'exprimer le besoin de chaque partie prenante dans chaque situation de vie qui la concerne. Le besoin est exprimé textuellement du point de vue des parties prenantes. Lors de cette étape, de nouvelles situations de vie peuvent émerger de l'écriture des besoins d'une partie prenante. Dans ce cas, l'architecte doit retourner à l'étape précédente pour harmoniser la liste des situations de vie. L'ajout d'une nouvelle situation de vie pourrait, à son tour, faire émerger de nouvelles parties prenantes et donc de nouveaux besoins. L'analyse de la liste des besoins avec la méthode PESTEL, introduite pour la première fois par AGUILAR [1967], pour Politique, Economique, Sociologique, Technologique, Ecologique et Légal permet la détection d'un champ de connaissances non pris en compte lors de la phase d'émergence des besoins. Par exemple, si l'architecte se rend compte qu'il n'a aucune exigence économique, c'est qu'il a oublié des parties prenantes et des situations de vie qu'il doit rajouter.
Il est très important de tracer les besoins dans une méthodologie de conception préliminaire. Si chaque besoin est bien lié à une partie prenante et à une situation de vie, il

est validé et peut être dérivé en exigences. Les exigences ainsi écrites vont, elles-mêmes, être liées à des choix technologiques. Si cette chaîne est bien conservée, il sera possible de justifier chaque choix technologique vis-à-vis d'un besoin d'une partie prenante. De ce fait, si jamais une partie prenante n'est plus à prendre en compte, alors une analyse de traçabilité permettra directement la détection des besoins, des exigences et des choix techniques qui en découlent et qui seront donc obsolètes.
Lors de la phase de capture des besoins, il est demandé à l'utilisateur d'exprimer chaque besoin par une phrase écrite en anglais, qui doit suivre le modèle suivant : «The stakeholder shall be able to do something with a defined level of performance in a defined life situation.»
Dans le cadre de ce manuscrit, rédigé en français, la phrase suivante sera utilisée : «La partie prenante doit être capable de faire quelque chose avec un certain niveau de performance dans une certaine situation de vie.».

3.3 Transformation des besoins en exigences

Une fois que tous les besoins pour chaque situation de vie sont exprimés, il est nécessaire de passer du monde des parties prenantes au monde du système à concevoir. Le canevas de KROB [2014] est utilisé pour cadrer la définition des exigences et faire en sorte que les utilisateurs de la méthodologie présentée décrivent bel et bien des fonctions du système. Ces exigences textuelles doivent ensuite être décomposées et transformées en exigences compréhensibles pour un ordinateur. Il a été décidé de ne pas utiliser l'analyse d'exigences textuelles par ordinateur proposée par FRANÇOIS CHRISTOPHE [2014]. Dans cette analyse, une spécification en langage naturel est transformée en une liste d'exigences formatées, adaptée à un traitement informatisé. Dans notre cas, il est demandé à l'utilisateur de mettre en forme, lui-même, les exigences qui vont servir à la génération automatique à l'aide d'un cadre bien détaillé.
Il est, dans un premier temps, demandé à l'utilisateur de réexprimer chaque besoin en une exigence écrite en anglais, qui doit suivre le modèle suivant : «The system shall do something with a defined level of performance in a defined life situation.».
Ensuite, toutes les exigences sont décomposées à la façon d'un diagramme FAST, mais sans aller jusqu'au choix des solutions. C'est-à-dire qu'aucune exigence décrite ne doit venir d'un choix technologique répondant à une autre exigence. En effet, comme le précise très justement FAISANDIER [2015a], habituellement, la liste des exigences se complète au fur et à mesure de l'allocation des exigences à des sous-systèmes. Quand un choix technologique est fait, de nouvelles exigences liées à ce choix apparaissent dans la spécification.
Dans la méthodologie présentée ici, la définition du système à concevoir doit rester totalement indépendante des sous-systèmes qui seront utilisés pour répondre aux exigences, car l'allocation sera faite de manière automatique et systématique par le logiciel.
Une fois la décomposition terminée, chaque exigence est triée dans trois catégories, comme cela est présenté dans NASA [2007] : exigences fonctionnelles, exigences de performance et exigences de contrainte.

3.3.1 Exigences fonctionnelles

Les exigences fonctionnelles sont des exigences mettant en œuvre des actions sur des flux. On peut se référer aux travaux de GIAMPÀ et collab. [2004] qui définissent neuf classes élémentaires de fonctions qui sont, pour rappel : bloquer, placer, contenir, convoyer (de la matière ou du signal), dissiper, fournir, transformer, transmettre (de l'énergie) et employer.

Néanmoins, nous laissons libre le choix des exigences fonctionnelles aux architectes tant qu'ils utilisent le canevas proposé précédemment.
Ces exigences fonctionnelles peuvent être associées à des «environment-centric functions» qui considère la fonction par rapport à ses effets par opposition aux «device-centric functions» qui considère la fonction par rapport à ses paramètres internes. Ces deux points de vue fonctionnels ont été identifiés par CHANDRASEKARAN et JOSEPHSON [2000]. Dans la méthodologie proposée dans ce manuscrit, tout le côté spécification s'applique uniquement sur le système au global.
Les exigences fonctionnelles textuelles doivent être transformées sous la forme de boîtes noires, définies par PAHL et collab. [2007], et d'exigences de performance, dans le but d'être traitées de manière automatique par le logiciel développé. Ces exigences portent un nom qui leur est propre et ont pour attribut une liste de ports pris dans la base de données définie par l'architecte, qui sera explicitée plus loin dans ce document. Cette forme de boîte noire est décrite par KROB [2014] comme une fonction, et non pas comme une exigence fonctionnelle, car elle ne contient plus de spécifications sur ses performances sous cette forme. Les exigences fonctionnelles transformées sous la forme d'une boîte noire présentées dans ce manuscrit sont en réalité plus complexes que de simples blocs sans attributs. En effet, ces objets fonctionnels sont en fait très proches des AO car ils sont instanciés par un ensemble d'AO. Autrement dit, une boite noire issue d'une exigence fonctionnelle correspond à une sorte de vue fonctionnelle d'un glsao, si un AO répond à l'exigence. Une exigence fonctionnelle est donc composée de ports, définis à l'aide d'un diagramme de classe sur la Figure 1.3, et permettant la représentation des transformations attendues de la fonction. Les ports sont définis par un nom et par un attribut de direction qui est, soit d'entrée, soit de sortie. C'est à dire que soit le port émet un flux ou reçoit un flux. Les multiplicités des ports de la boîte noire fonctionnelle doivent aussi être définies à ce moment. Notons que, même sous cette représentation, une exigence fonctionnelle reste associée à une situation de vie du système qu'elle spécifie. Cela veut dire qu'une boîte noire ne sera valide que dans une situation de vie ou un ensemble de situations de vie bien définis. Précédemment, la vue physique des AO a été abordée, elle sera étudiée plus en détail dans ce chapitre, mais l'important est de se rendre compte que la partie exigence de performance, contenue dans l'exigence fonctionnelle textuelle, est utile lors de la spécification physique du système car elle a un impact sur la magnitude des flux échangés et non pas sur leur type. Pour plus de simplicité le type d'un port et donc d'un flux est son nom.

3.3.2 Exigences de performance

Les exigences de performance proviennent de deux sources : directement de certains besoins qui n'ont pas été traduits à l'aide de fonctions et d'exigences fonctionnelles transformées en boîtes noires. En effet, comme vu juste au-dessus, lors de la transformation d'une exigence fonctionnelle en boîte noire, sans création d'exigences de performance, une partie de l'information serait perdue. Chaque performance émanant d'un besoin non fonctionnel, comme une masse maximale par exemple, est rattachée à des variables dites d'état du système. L'objectif est d'écrire des égalités ou inégalités sur les variables d'état du système à concevoir. La prise en compte de ces performances permet à l'architecte de savoir quels sont les paramètres clés du système à concevoir. En effet, lors de la définition des AO, il saura que tous ces paramètres clés doivent être présents dans les variables d'état de ces composants. Ce point sera étudié plus en détail dans un chapitre suivant sur les architectures physiques. Notons que, dans plusieurs ouvrages, ces besoins ne pouvant

être traduits en exigences fonctionnelles sont plutôt appelés des contraintes. Dans ce manuscrit, ce terme n'est pas utilisé dans ce cas-là, mais uniquement pour des exigences de contrainte ne s'appliquant que sur l'architecture logique, comme expliqué ci-dessous. De même, les variables d'échange qui définissent les flux attendus aux bornes du système peuvent aussi être spécifiées à l'aide d'exigences de performance qui vont contraindre le dimensionnement des architectures physiques. Ces exigences de performance peuvent aider l'architecte à définir les types de flux et donc de ports vis-à-vis des variables d'échange qui les caractérisent. Si par exemple, une exigence fonctionnelle textuelle était la suivante : «Le système doit fournir une puissance hydraulique de 1 kW en phase nominale», alors le port «Puissance hydraulique» se devrait de contenir une variable d'échange pouvant recevoir cette exigence de performance de 1 kW.
Il est bon de préciser que ces exigences de performance peuvent être associées à une phase de vie du système à concevoir ou porter sur ses caractéristiques intrinsèques. Cette distinction doit être faite par l'architecte lors de la spécification du système. Une exigence de performance émanant d'une exigence fonctionnelle textuelle est forcément liée à une situation de vie car l'exigence fonctionnelle textuelle l'était déjà. Une exigence de performance venant d'un besoin non fonctionnel peut être, au choix, appliquée à l'ensemble des situations de vie, ou, avec des valeurs différentes pour chaque cas opérationnel.
Pour rappel, l'objectif de nos travaux est de proposer de manière automatique et systématique des architectures innovantes. Ces architectures pourront donc être non conventionnelles et avoir des caractéristiques architecturales très différentes des systèmes existants.

3.3.3 Exigences de contrainte

Les exigences de contrainte, comme leur nom l'indique, contraignent la conception de l'architecture. Il s'agit d'imposer au logiciel de génération des sous-systèmes à utiliser dans les solutions ou d'imposer des nombres d'occurrences sur les technologies. Car en effet, le processus de recherche automatique de solution minimise le nombre d'occurrences des AO utilisés. La recherche est exhaustive, mais uniquement pour les solutions les plus simples, c'est-à-dire, celles qui limitent le nombre de composants. Il peut néanmoins être nécessaire de forcer l'utilisation de plusieurs occurrences d'un même composant pour faire des études sur des solutions multirotors pour un drone aérien, par exemple, en augmentant le nombre d'occurrences maximal de quatre à seize. De même, il est possible de forcer l'utilisation d'un composant dans toutes les solutions en fixant son nombre d'occurrences minimum à un. Par contre, pour interdire l'utilisation d'un AO, il est préférable de le supprimer de la base de données plutôt que de fixer son nombre maximal d'occurrences à zéro car, comme cela sera vu plus tard dans le manuscrit, dans les premières phases de la synthèse, le composant sera quand même considéré.
Les exigences sur le niveau de sécurité évoquées précédemment peuvent aussi être traitées à l'aide des exigences de contrainte. En effet, en utilisant un retour d'expérience sur les technologies disponibles dans la société, il est possible de savoir le nombre d'occurrences nécessaires d'un sous-système pour être en accord avec les exigences sur le niveau de sécurité. Par exemple, en connaissant le taux de panne d'une pompe hydraulique, on peut savoir qu'il est nécessaire d'en avoir au moins trois dans chaque solution pour obtenir le niveau de sécurité nécessaire.

3.4 Définition des ports du système

A ce point du processus de synthèse, toutes les exigences de plus bas niveau sont mises en forme. Si la méthodologie a été bien suivie, tous les besoins ont été traités et les trois catégories d'exigences sont exhaustives et décrivent bien toutes les phases de vie du système.
Dans une méthodologie non assistée par ordinateur, l'allocation des exigences aux sous-systèmes est faite à ce moment et comme dit précédemment, cette allocation va créer de nouvelles exigences.
Dans notre méthodologie assistée par ordinateur, il n'y a pas d'allocation manuelle. Toutes les fonctions internes du système, c'est-à-dire qui ne communiquent nullement avec l'environnement, ne sont pas modélisées. En effet, très souvent la décomposition des fonctions de haut niveau est le fruit de choix technologiques. Dans la méthodologie proposée, aucune des fonctions uniquement internes, c'est-à-dire qui n'auront aucun impact sur l'environnement, n'est modélisée. Mais par contre, les AO modélisés sous la vue logique sont des objets technologiques définis comme des fonctions sous forme de boîtes noires. Tous les ports non désirés d'AO intégrés dans la solution vont induire des comportements non voulus et devront être traités par d'autres AO. De fil en aiguille, cela va permettre de créer l'architecture sans avoir prédéfini les fonctions internes au système, qui sont le fruit de choix technologiques.

3.4.1 Définition des ports du système au global

Toutes les exigences fonctionnelles sous forme de boîtes noires et qui ne sont pas strictement internes au système, sont agrégées pour fournir une représentation globale du système à concevoir. Cette représentation élicite toutes les interfaces avec l'environnement. Elle est logiquement formée par un ensemble de ports car les exigences fonctionnelles étaient déjà définies par des ports venant de la base de données. Les exigences de performance exprimées depuis les besoins fonctionnels des parties prenantes se trouvent forcément aux interfaces du système. Elles vont donc caractériser tous les ports restant aux interfaces. Les exigences de performance émanant de besoins non fonctionnels spécifient l'intérieur du système, mais pas ses interfaces internes, uniquement ses paramètres globaux, appelés variables d'état dans ce manuscrit. Il ne s'agit pas à proprement parler d'interfaces vers l'environnement, mais les contraintes, qui vont s'appliquer sur les variables d'état du système. Elles vont aussi avoir un impact sur les variables d'échange, à travers les équations comportementales qui seront traitées en détail dans le chapitre 5, page 111.

Concernant la représentation boîte noire du système à concevoir, la vue physique se compose de :

- une liste de variables d'état, chaque variable ayant elle-même plusieurs attributs qui sont :
  - une unité par variable d'état,
  - une borne basse pour la valeur que pourra prendre la variable d'état,
  - une borne haute pour la valeur que pourra prendre la variable d'état,
  - une liste de contraintes associées à chaque variable d'état pour chaque situation de vie,
  - et une variable de choix pour savoir si la variable d'état de la fonction est imposée à toutes les variables d'état identiques des AO de la base de données

dans toutes les situations de vie ou si cette variable est le résultat de la somme de toutes les variables d'état identiques des AO de la base de données,

- une liste d'équations comportementales pour chaque mode de fonctionnement,
- et une liste de ports ayant eux-mêmes leur propre vue physique composée d'une liste de variables d'échange ayant pour attributs :
  - une unité par variable d'échange,
  - une borne basse pour la valeur que pourra prendre la variable d'échange,
  - une borne haute pour la valeur que pourra prendre la variable d'échange,
  - une liste de contraintes associées à chaque variable d'échange pour chaque mode de fonctionnement,
  - et un booléen pour savoir si la variable d'échange est associative ou non.

L'utilisation des objets définis ici, dans la vue physique du système à concevoir, sera explicitée en détail dans le chapitre 5, page 111 lorsque les architectures physiques seront étudiées.

3.4.2 Définition des ports du système dans chaque situation de vie

Le système à concevoir évolue dans plusieurs situations de vie. On peut donc déterminer ses interfaces avec l'environnement en fonction des cas opérationnels qu'il va rencontrer. Lors de la génération, cela permet de sortir des solutions qui peuvent répondre aux exigences dans chaque situation de vie. Lors de l'étape de transformation des besoins en exigences, les situations de vie avaient déjà été considérées. Pour définir les ports du système à concevoir dans une situation de vie donnée, les exigences fonctionnelles sous forme de boîtes noires à considérer ne sont que celles correspondant à la même situation de vie. Forcément, les exigences de performance associées sont aussi relatives aux situations de vie et doivent être associées aux ports actifs dans le cas opérationnel. Par exemple, un système mécatronique doit être en mesure de pouvoir démarrer, il ne faut pas uniquement considérer son fonctionnement nominal. La définition des interfaces nécessaires à la situation de vie "démarrage", permet donc de s'assurer que le système sera en mesure de démarrer, en plus de fournir les flux sortants pour la situation nominale. De même, une même architecture peut avoir des fonctionnements différents dans une même situation de vie, suivant les modes de fonctionnement choisis pour chaque AO.
Prenons l'exemple d'une architecture d'un groupe électrogène avec deux moteurs thermiques et deux alternateurs. Cette architecture permet au groupe électrogène de fonctionner dans une configuration économe si la demande en courant électrique est faible ou dans une configuration haute puissance si la demande est élevée. Une autre possibilité pourrait être de ne jamais éteindre un des moteurs, même si la consommation électrique venait à baisser, pour éviter de rallumer ledit moteur. De ce fait, une même architecture physique peut être différente au regard de son comportement dynamique. C'est pour cela qu'il est nécessaire de considérer chaque AO avec ses différents modes de fonctionnement.

3.5 Exemple fil rouge

L'exemple fil rouge de ce manuscrit est un système drone volant, devant permettre le transport de charges et la surveillance de zones éloignées. Toutes les étapes de la synthèse assistée par ordinateur vont être décrites tout au long de ce manuscrit dans les sections d'exemple.

3.5.1 Identification des situations de vie et des parties prenantes

Premièrement, une liste de parties prenantes et une liste de situations de vie sont établies sans méthodologie particulière. Dans notre cas, en première approche, ces deux listes ont émergé :
Parties prenantes :

- Environnement extérieur
- Charge utile
- Sol
- Réseau électrique [1]
- Régulation
- Pilote

Situations de vie :

- Phase de transport de la charge utile
- Phase d'observation
- Phase de rechargement

Dans un second temps, pour chaque situation de vie, un diagramme d'environnement mettant en œuvre les parties prenantes est réalisé. Lors de cette étape, de nouvelles parties prenantes peuvent émerger. La phase de transport de la charge utile est une des deux phases de vie principales de notre système à concevoir. Elle met en jeu l'environnement, le pilote et la charge utile. Le diagramme d'environnement associé est visible sur la Figure 3.4.
La phase d'observation est l'autre phase de vie principale de notre système à concevoir.

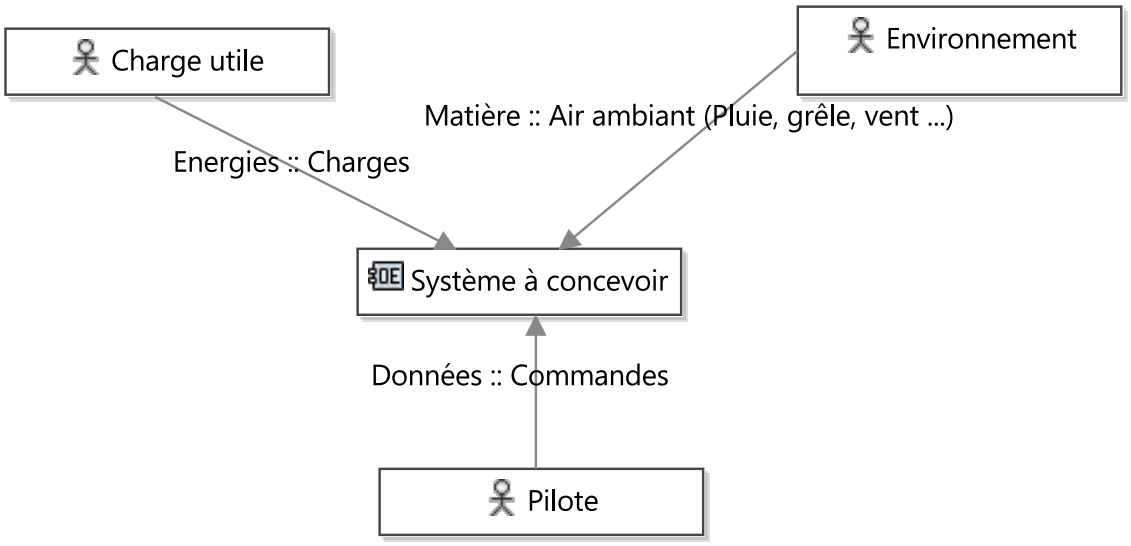


FIGURE 3.4 – Diagramme d'environnement pour la phase de transport de la charge utile.

Elle met en jeu l'environnement et le pilote. Le diagramme d'environnement associé est visible sur la Figure 3.5. La phase de rechargement représentée sur la Figure 3.6 fait apparaître l'équipage au sol. Ensuite, un diagramme représentant le cycle de vie du système à concevoir est tracé. Cela permet de détecter des situations de vie qui ont été oubliées. Ce diagramme est représenté sur la Figure 3.7. Plusieurs phases qui ne

1. Même si le réseau électrique est placé en tant que partie prenante, le système drone n'est pas forcément électrique car il peut être constitué d'un système carburant en interaction avec le pilote. Le réseau électrique peut être utilisé pour recharger une petite batterie pour les phases de démarrage ou alors de grosses batteries qui seront la seule source d'énergie du système.

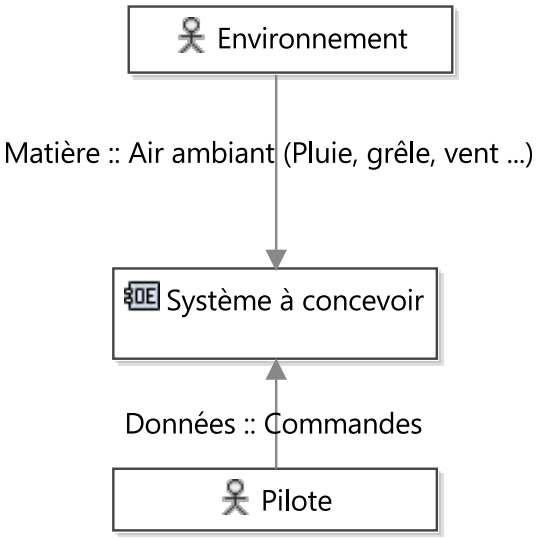


FIGURE 3.5 – Diagramme d'environnement pour la phase d'observation.

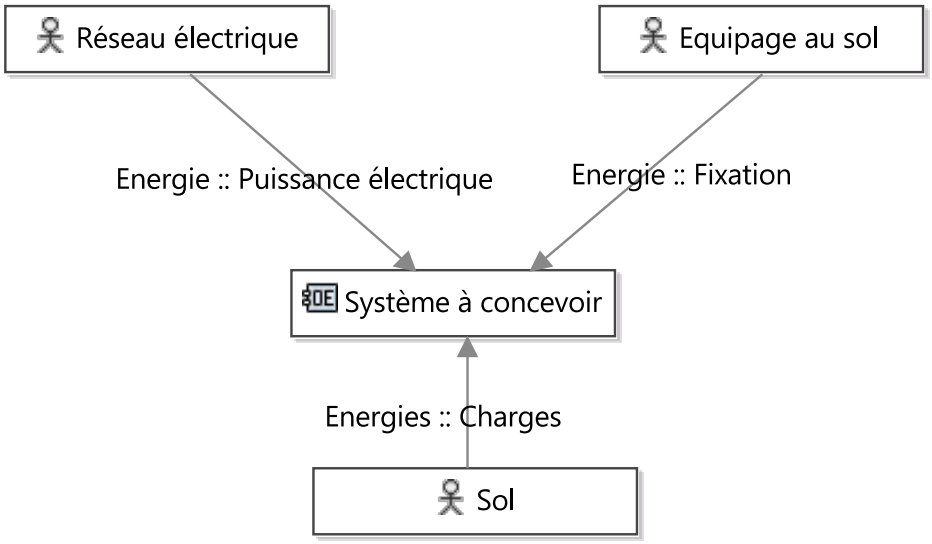


FIGURE 3.6 – Diagramme d'environnement pour la phase de rechargement.

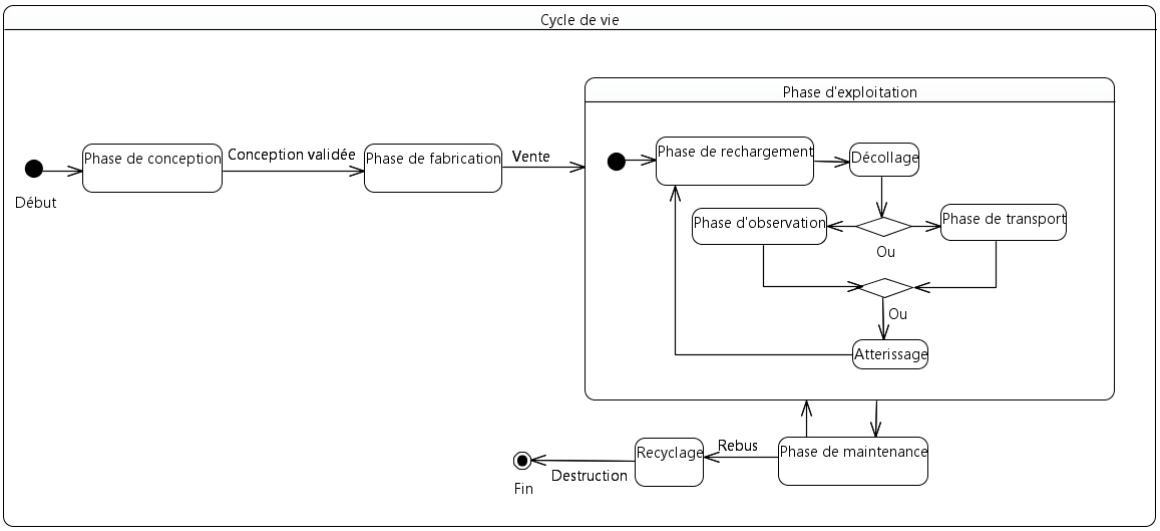


FIGURE 3.7 – Cycle de vie.

correspondent pas aux phases de vie principales sont ajoutées et cela permet ensuite d'identifier de nouvelles parties prenantes. Au final, il est possible de décrire un diagramme d'environnement prenant en compte toutes les parties prenantes. Ce dernier est visible sur la Figure 3.8.

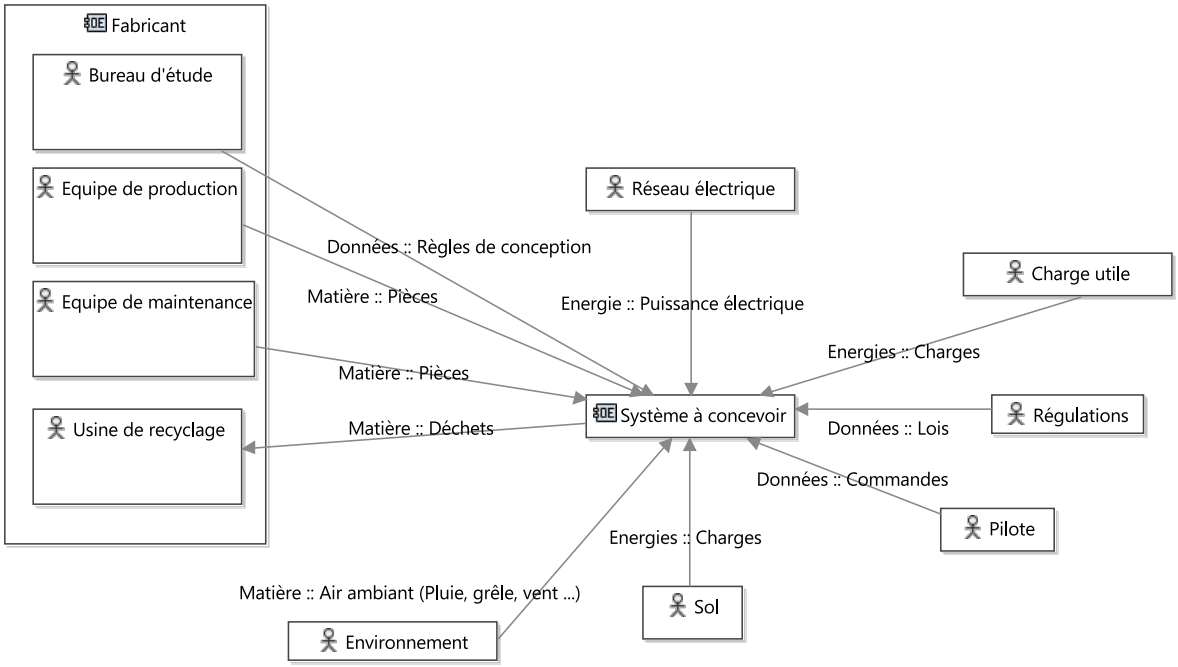


FIGURE 3.8 – Diagramme d'environnement global.

3.5.2 Expression du besoin

Une fois cette première étude terminée, quand toutes les parties prenantes et les situations de vie importantes ont été déterminées, un ensemble de besoins peuvent être exprimés. En suivant le canevas décrit précédemment, pour chaque partie prenante dans chaque situation de vie dans laquelle elle est impliquée, une liste de besoins est extraite. Dans ce manuscrit, les besoins, exigences fonctionnelles, exigences de performance et exigences de contrainte seront exprimés en français. Les axes principaux du système drone seront nommés suivant le système d'axe classique aéronautique rappelé sur la Figure 3.9. Commençons cet exemple avec l'expression des besoins du pilote dans toutes les situations de vie dans lesquelles il est partie prenante. Le pilote est partie prenante des phases de décollage, d'observation, de transport de la charge utile et de l'atterrissage. Ces quatre phases de vie font partie d'une même grande phase de vie du système qui est la phase d'exploitation. Les besoins communs à ces situations de vie ne seront exprimés qu'une seule fois.

— Le pilote doit pouvoir commander le déplacement de la charge en temps réel en phase d'exploitation.

— Le pilote doit pouvoir localiser le système drone avec une précision de 20 cm en phase d'exploitation.

— Le pilote doit pouvoir observer fixement un point avec des déplacements parasites inférieurs à 20 cm en phase d'observation.

— Le pilote doit être averti avant de sortir du domaine de vol d'un point de vue de l'altitude 10 m avant le franchissement en phase d'exploitation.

FIGURE 3.9 – Axes principaux des aéronefs.

— Le pilote doit pouvoir transporter une charge à une vitesse max de 100 km/h en phase de transport.

Ensuite, attardons-nous sur les besoins de la charge à transporter vis-à-vis du système drone dans toutes les situations de vie dans lesquelles elle est partie prenante. La charge est partie prenante dans la phase de décollage, la phase de transport de la charge utile et la phase d'atterrissage.

— La charge utile doit être transportée sans dommages en phase d'exploitation.

— La charge utile doit être transportée sans subir les intempéries en phase d'exploitation.

Le réseau électrique n'a pas vraiment de besoins à proprement parler, mais il peut quand même permettre la spécification du système à concevoir.

— Le réseau électrique doit pouvoir se connecter à l'aide de ses connecteurs normalisés au système à concevoir en phase de rechargement.

Le sol est une partie prenante qui peut aussi exprimer plus une contrainte qu'un besoin, mais qui va avoir un impact sur le système.

— Le sol doit pouvoir soutenir le système sans déformations supérieures à 5 cm à l'atterrissage.

L'équipe de production a forcément des besoins vis-à-vis du processus de fabrication et d'assemblage du système, ces besoins sont exprimés, même si le système ne pourra pas répondre directement aux besoins. De nouveau, il s'agit plus de contraintes que de besoins.

— L'équipe de production doit pouvoir assembler le système en moins de trois semaines dans la phase de fabrication.

— L'équipe de production doit pouvoir porter manuellement tous les éléments à l'exception des ensembles pré-montés dans la phase de fabrication.

Ce dernier besoin est partagé avec l'équipe de maintenance. Aucun autre besoin de l'équipe de maintenance ne sera remonté dans le cadre de cet exemple. Une phase pourrait avoir

son importance, il s'agit d'un auto-diagnostique. Il s'agit d'une phase ou les sous-systèmes du système drone remontent eux-mêmes les défaillances sous forme d'échanges de données. Dans le cadre de cet exemple nous allons rester sur des exigences de base et nous ne traiterons pas le cas d'auto-diagnostique.
Concernant les parties prenantes restantes, c'est-à-dire le bureau d'étude, l'usine de recyclage, l'environnement et la régulation, aucun nouveau besoin ne sera exprimé dans ce manuscrit, même s'il est certain que plusieurs besoins manquent à l'appel. L'objectif ici est d'illustrer le propos et non pas de donner un exemple industriel complet.

3.5.3 Transformation des besoins en exigences

Une fois cette seconde étude terminée avec une liste des parties prenantes exhaustive, une liste des situations de vie, ou cas opérationnels, exhaustive et une liste des besoins, suivant le canevas présenté précédemment, l'écriture des exigences peut commencer. L'objectif ici est de transformer les besoins des parties prenantes, qui ont été capturés à l'aide d'une vision extérieure au système, en exigences avec une vision interne au système. Chaque exigence textuelle est ensuite traitée dans le but d'identifier les parties fonctionnelles, les parties touchant à la performance et celles contraignant le système.

TABLEAU 3.1 – Transformation des besoins en exigences

Besoins	**Exigences**
B.1 Le pilote doit pouvoir commander le déplacement de la charge en temps réel en phase d'exploitation.	E.1.1 Le système drone doit recevoir les commandes provenant du pilote en temps réel en phase d'exploitation. E.1.2 Le système drone doit interpréter les commandes provenant du pilote en temps réel en phase d'exploitation.
B.2 Le pilote doit pouvoir localiser le système drone avec une précision de 20 cm en phase d'exploitation.	E.2.1 Le système drone doit acquérir sa position avec une précision de 20 cm en phase d'exploitation. E.2.2 Le système drone doit envoyer sa position en temps réel en phase d'exploitation.
B.3 Le pilote doit pouvoir observer fixement un point avec des déplacements parasites inférieurs à 20 cm en phase d'observation.	E.3.1 Le système drone doit acquérir une vidéo de son environnement dans une direction en phase d'observation. E.3.2 Le système drone doit transférer en temps réel l'information de visualisation en phase d'exploitation. E.3.3 Le système drone doit générer un couple suivant l'axe de lacet pour diriger la direction d'acquisition lors de la phase d'observation.
B.4 Le pilote doit être averti avant de sortir du domaine de vol d'un point de vue de l'altitude 10 m avant le franchissement en phase d'exploitation.	E.4.1 Le système drone doit acquérir l'altitude de vol en temps réel dès le décollage.

continue sur la page suivante

continue depuis la page précédente

Besoins	**Exigences**
	E.4.2 Le système drone doit transférer la valeur réelle de l'altitude au pilote en temps réel après le décollage.
B.5 Le pilote doit pouvoir transporter une charge à une vitesse max de 100 km/h en phase de transport.	E.5.1 Le système drone doit générer une poussée supérieure à sa masse maximale chargée dès le décollage. E.5.2 Le système drone doit générer une force de propulsion permettant le déplacement à la masse maximale à 100 km/h en exploitation. E.5.3 Le système drone doit avoir un coefficient de traînée inférieur à 0.5.
B.6 La charge utile doit être transportée sans dommages en phase d'exploitation.	E.6.1 Le système drone doit protéger la charge utile des chocs en phase de transport.
B.7 La charge utile doit être transportée sans subir les intempéries en phase d'exploitation.	E.7.1 Le système drone doit protéger la charge utile des agressions extérieures en phase de transport.
B.8 Le réseau électrique doit pouvoir se connecter à l'aide de ses connecteurs normalisés au système à concevoir en phase de rechargement.	E.8.1 Le système drone doit contenir une prise électrique normalisée pour se connecter au réseau électrique pendant la phase de rechargement.
B.9 Le sol doit pouvoir soutenir le système sans déformations supérieures à 5 cm à l'atterrissage.	E.9.1 Le système drone doit appliquer une pression sur le sol de moins de 3MPa à l'atterrissage.
B.10 L'équipe de production doit pouvoir assembler le système en moins de trois semaines dans la phase de fabrication.	E.10.1 Le système drone doit contenir uniquement de la visserie normalisée. E.10.2 Le système drone doit être composé de moins de 20 assemblages.
B.11 L'équipe de production doit pouvoir porter manuellement tous les éléments à l'exception des ensembles pré-montés dans la phase de fabrication.	E.11.1 Chaque pièce ou sous-assemblage du système drone doit peser moins de 20 kg.

Une fois que les exigences ont émergé des besoins des parties prenantes, elles peuvent être classées pour être transformées, dans le but d'être traitées de manière informatique.

TABLEAU 3.2 – Tri des exigences

	Exigence fonctionnelle	Exigence de performance	Exigence de contrainte
E.1.1 Le système drone doit recevoir les commandes provenant du pilote en temps réel en phase d'exploitation.	**X**		
E.1.2 Le système drone doit interpréter les commandes provenant du pilote en temps réel en phase d'exploitation.	**X**		
E.2.1 Le système drone doit acquérir sa position avec une précision de 20 cm en phase d'exploitation.	**X**	**X**	
E.2.2 Le système drone doit envoyer sa position en temps réel en phase d'exploitation.	**X**		
E.3.1 Le système drone doit acquérir une vidéo de son environnement dans une direction en phase d'observation.	**X**		
E.3.2 Le système drone doit transférer en temps réel l'information de visualisation en phase d'exploitation.	**X**		
E.3.3 Le système drone doit générer un couple suivant l'axe de lacet pour diriger la direction d'acquisition lors de la phase d'observation.	**X**		
E.4.1 Le système drone doit acquérir l'altitude de vol en temps réel dès le décollage.	**X**		
E.4.2 Le système drone doit transférer, au pilote, la valeur réelle de l'altitude en temps réel après le décollage.	**X**		

continue sur la page suivante

continue depuis la page précédente			
	Exigence fonctionnelle	**Exigence de performance**	**Exigence de contrainte**
E.5.1 Le système drone doit générer une poussée supérieure à sa masse maximale chargée dès le décollage.	**X**	**X**	
E.5.2 Le système drone doit générer une force de propulsion permettant le déplacement à la masse maximale à 100 km/h en exploitation.	**X**	**X**	
E.5.3 Le système drone doit avoir un coefficient de traînée inférieur à 0.5.		**X**	
E.6.1 Le système drone doit protéger la charge utile des chocs en phase de transport.			**X**
E.7.1 Le système drone doit protéger la charge utile des agressions extérieures en phase de transport.			**X**
E.8.1 Le système drone doit contenir une prise électrique normalisée pour se connecter au réseau électrique pendant la phase de rechargement.	**X**		
E.9.1 Le drone doit appliquer une pression sur le sol de moins de $3\ \mathrm{MP}a$ **à l'atterrissage.**	**X**	**X**	
E.10.1 Le drone doit contenir uniquement de la visserie normalisée.			**X**
E.10.2 Le drone doit être composé de moins de 20 assemblages.			**X**
E.11.1 Chaque pièce ou ensemble du drone doit peser moins de 20 kg.		**X**	

Pour chaque besoin, plusieurs exigences viennent d'être trouvées. Néanmoins, ces exigences sont d'un haut niveau d'abstraction, il peut être nécessaire de les raffiner en

faisant attention de ne pas descendre dans des exigences spécifiant les solutions techniques. Seules les exigences de contrainte peuvent imposer des choix technologiques. Dans l'exemple, l'exigence E.1.1 est décomposée en trois, comme illustré sur la Figure 3.10.

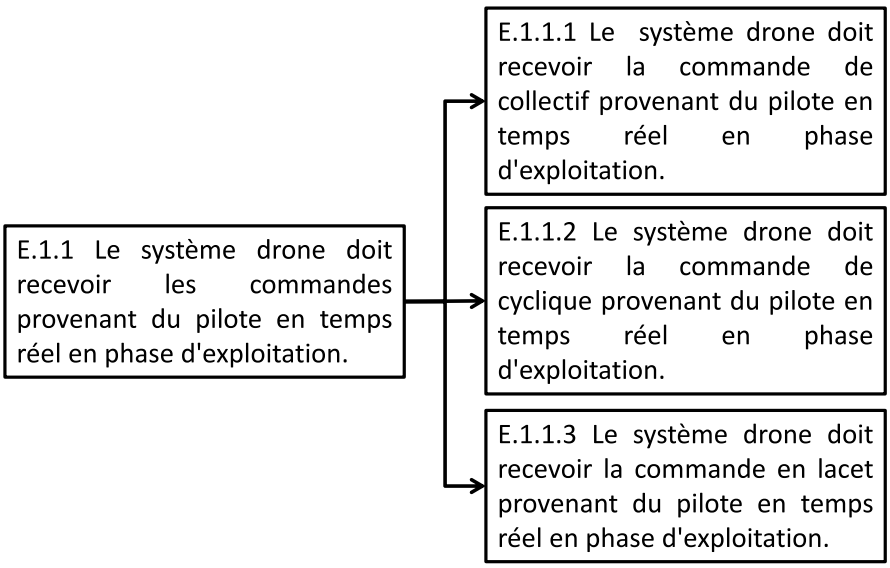
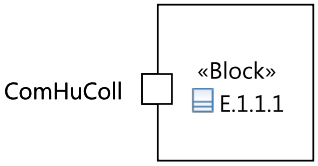
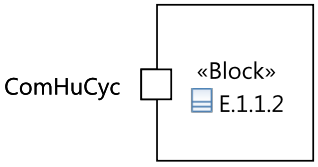


FIGURE 3.10 – Décomposition de l'exigence E.1.1.

3.5.4 Exigences fonctionnelles

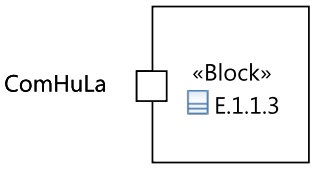
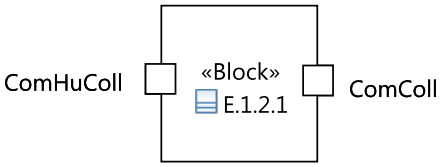
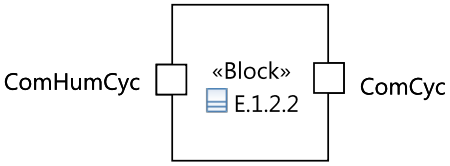
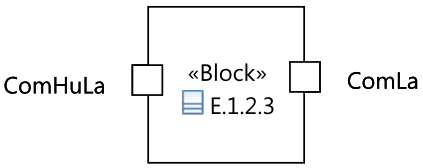
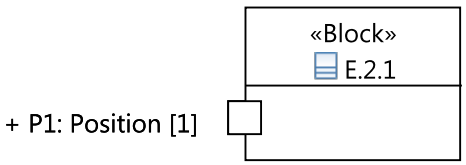
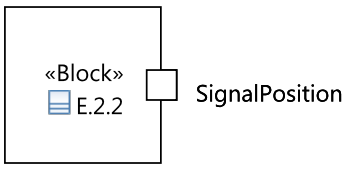
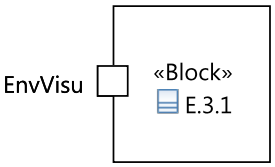
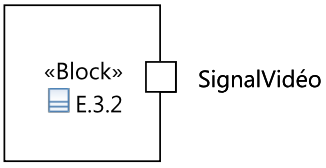
Les exigences fonctionnelles textuelles doivent maintenant être transformées en boîtes noires. Cela permet de décrire le fonctionnement statique d'une fonction, comme cela a été introduit par PAHL et collab. [2007]. A ce moment, les premiers types de port sont définis. Usuellement, ces types sont rangés en trois catégories : matière, énergie et signal. Néanmoins les types sont laissés à l'initiative de l'architecte modélisant le système à concevoir. Chaque boîte noire est définie par son nom, qui est l'identifiant de l'exigence textuelle associée, par un ensemble de ports contenant eux-mêmes un nom et par un intervalle de multiplicités accordées. Les autres attributs du port définis sur la Figure 1.3 sont renseignés plus tard dans le processus de synthèse.

TABLEAU 3.3 – Transformation en boîtes noires

Exigences fonctionnelles	Boîtes noires
E.1.1.1 Le système drone doit recevoir la commande de collectif provenant du pilote en temps réel en phase d'exploitation.	
E.1.1.2 Le système drone doit recevoir la commande de cyclique provenant du pilote en temps réel en phase d'exploitation.	

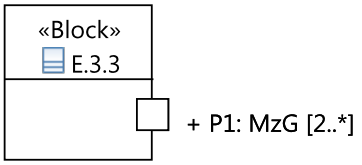
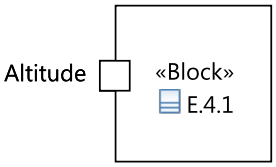
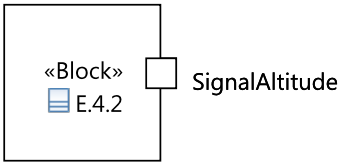
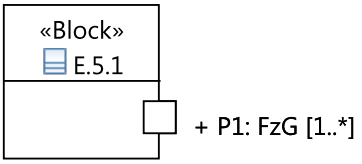
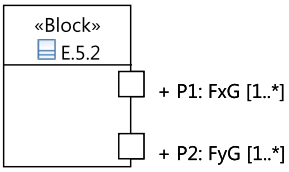
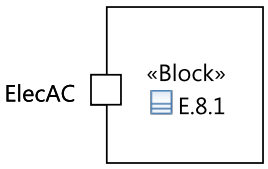
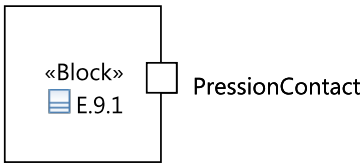
continue sur la page suivante

continue depuis la page précédente

Exigences fonctionnelles	**Boîtes noires**
E.1.1.3 Le système drone doit recevoir la commande en lacet provenant du pilote en temps réel en phase d'exploitation.	
E.1.2.1 Le système drone doit interpréter la commande de collectif provenant du pilote en temps réel en phase d'exploitation.	
E.1.2.2 Le système drone doit interpréter la commande de cyclique provenant du pilote en temps réel en phase d'exploitation.	
E.1.2.3 Le système drone doit interpréter la commande en lacet provenant du pilote en temps réel en phase d'exploitation.	
E.2.1 Le système drone doit acquérir sa position avec une précision de 20 cm en phase d'exploitation.	
E.2.2 Le système drone doit envoyer sa position en temps réel en phase d'exploitation.	
E.3.1 Le système drone doit acquérir une vidéo de son environnement dans une direction en phase d'observation.	
E.3.2 Le système drone doit transférer en temps réel l'information de visualisation en phase d'exploitation.	

continue sur la page suivante

continue depuis la page précédente

Exigences fonctionnelles	Boîtes noires
E.3.3 Le système drone doit générer un couple suivant l'axe de lacet pour diriger la direction d'acquisition lors de la phase d'observation.	
E.4.1 Le système drone doit acquérir l'altitude de vol en temps réel dès le décollage.	
E.4.2 Le système drone doit transférer la valeur réelle de l'altitude au pilote en temps réel après le décollage.	
E.5.1 Le système drone doit générer une poussée supérieure à sa masse maximale chargée dès le décollage.	
E.5.2 Le système drone doit générer une force de propulsion permettant le déplacement à la masse maximale à 100 km/h en exploitation.	
E.8.1 Le système drone doit contenir une prise électrique normalisée pour se connecter au réseau électrique pendant la phase de rechargement.	
E.9.1 Le système drone doit appliquer une pression sur le sol de moins de 3 MPa à l'atterrissage.	

Ici, seize différents types de flux sont définis lors du passage des exigences fonctionnelles à la représentation sous forme de boîtes noires, comme illustré dans le Tableau 3.3. Trois flux concernent la commande du pilote, il s'agit des entrées principales du système à concevoir et ils sont nommés «ComHumColl» pour la commande collective, «ComHuCy» pour la commande cyclique et «ComHuLa» pour la commande en lacet venant de l'humain. Ensuite, les mêmes commandes sont définies, mais cette fois il s'agit de signaux

mécaniques ou électroniques. Le choix n'est pas encore fait à ce niveau. Ces trois signaux sont nommés «ComColl» pour le signal du collectif, «ComCyc» pour le signal du cyclique et «ComLa» pour le signal du lacet. La convention prise pour définir le contrôle du système drone est donc composée de trois parties. Un contrôle dit collectif nous vient du monde hélicoptère et signifie une commande uniquement suivant l'axe z colinéaire au vecteur $\vec{g}$ représentant la gravité. La commande cyclique, elle, revient à déplacer le système drone dans le plan x,y perpendiculaire au vecteur $\vec{g}$, et la commande en lacet permet la rotation du système drone autour de l'axe z passant par son centre de gravité. Ces termes viennent du fait que, pour un hélicoptère conventionnel à rotor principal articulé, chaque pale reçoit un angle différent à l'aide d'un plateau inclinable lors d'une commande cyclique et un même angle lors d'une commande collective. Si chaque pale reçoit le même angle d'inclinaison, elles vont toutes produire la même portance et l'hélicoptère montera sans bouger dans le plan x,y. Par contre, si chaque pale reçoit un angle d'inclinaison différent suivant son emplacement autour du rotor, une différence de portance va être induite et l'hélicoptère va bouger dans le plan x,y. Pour visualiser ces trois axes sur un aéronef, veuillez vous reporter à la Figure 3.9.
Un signal de type position est ensuite introduit sous le nom de «SignalPosition», il s'agit de la véritable position du système drone, qui doit être enregistrée et transmise au pilote. Ensuite, un signal «EnvVisu» pour le véritable environnement est créé, qui devra être transformé en un signal «SignalVidéo», qui devra être transmis au pilote. De la même façon, les flux «Altitude» et «SignalAltitude» sont présentés.
Pour que le système puisse se mouvoir dans les airs, il faut qu'il produise des forces sur son environnement. Ces forces et moment sont modélisées fonctionnellement par quatre flux «MzG», «FxG», «FyG» et «FzG».
Ensuite, un flux «ElecAC» est introduit pour représenter une connexion avec un système électrique permettant la recharge. Notons que ce système est une des parties prenantes précédentes. En effet, le système à concevoir qui est en train d'être spécifié sous la forme d'une boîte noire, qui devra contenir tous les ports en interface avec les parties prenantes et pas de ports internes. Cela sera illustré dans peu de temps lors de la définition des ports d'interface du système.
Enfin, un flux «PressionContact» est utilisé pour représenter l'exigence de non-enfoncement du système drone dans le sol.

3.5.5 Exigences de performance

Les exigences de performance portent sur des valeurs de variables d'état ou de variables d'échange du système. Il est nécessaire de définir pour les flux précédemment décrits les variables d'échange mises en jeu et aussi pour l'ensemble du système, les variables d'état. Néanmoins certains ports ne resteront définis que par une vue logique et pas par une vue physique. Dans le cas décrit dans cet exemple, les ports «ComHuColl», «ComHuCyc», «ComHuLa», «ComColl», «ComCyc», «ComLa», «SignalPosition», «EnvVisu», «SignalVidéo», «Altitude» et «SignalAltitude» ne seront pas définis pour la vue physique. Notons qu'il s'agit principalement de flux d'information qui doivent apparaître dans la vue logique, mais qui n'ont pas autant de sens dans une vue physique lors d'une étude architecturale. Bien entendu, lors d'une étude détaillée, les flux d'information doivent être associées à une technologie. Par exemple un signal électrique, un signal lumineux, un son, etc...
Le détail des flux sera donné lors de la création de la base de données des AO. De même, les variables d'état du système à concevoir seront explicitées lors de la définition de tous les ports d'interface du système. Reprenons ici uniquement les exigences de performance

qui ont émergé précédemment, en déterminant si elles s'appliquent sur des variables d'échange ou des variables d'état du système. Certaines performances sont liées à des variables d'état et portent sur des variables d'échange. Dans ce cas, elles peuvent être allouées aux deux types d'exigences.

TABLEAU 3.4 – Allocation des exigences de performance

	Variable d'échange	**Variable d'état**
E.2.1 Le drone doit acquérir sa position avec une précision de 20 cm en phase d'exploitation.	**X**	**X**
E.5.1 Le drone doit générer une poussée supérieure à sa masse maximale chargée dès le décollage.	**X**	**X**
E.5.2 Le drone doit générer une force de propulsion permettant le déplacement à la masse maximale à 100 km/h en exploitation.	**X**	**X**
E.5.3 Le drone doit avoir un coefficient de traînée inférieur à 0.5.		**X**
E.9.1 Le drone doit appliquer une pression sur le sol de moins de $3\ \mathrm{MP}a$ **à l'atterrissage.**	**X**	**X**
E.11.1 Chaque pièce ou ensemble du drone doit peser moins de 20 kg.		**X**

3.5.6 Exigences de contrainte

Comme décrit précédemment, la synthèse automatisée proposée dans ce manuscrit tend à minimiser le nombre d'AO utilisés dans les solutions générées. Néanmoins, dans certains cas, et cela est vrai ici, l'utilisateur peut vouloir tester des solutions plus complexes vis-à-vis du nombre d'AO. Dans cet exemple, nous souhaitons voir apparaître des architectures de type hélicoptère conventionnel avec un rotor principal articulé et un rotor anticouple, mais aussi des architectures de type quadcopter avec quatre rotors non articulés. Pour obtenir ce dernier type d'architecture, il est donc obligatoire d'imposer le nombre d'occurrences de l'AO rotor non articulé de zéro à quatre. Tous les autres nombres d'occurrences sont laissés libres et seront minimisés, à part dans certains cas qui seront explicités plus tard. Dans le Tableau 3.2 page 60, d'autres exigences de contrainte avaient émergé, concernant le type de visserie à utiliser et le nombre maximal d'assemblages autorisé. La première exigence est trop précise au niveau architectural, c'est une exigence qui s'appliquera plutôt à la conception détaillée, elle est donc laissée de côté pour cet exemple. La seconde par contre, est une contrainte qui va permettre de limiter la complexité des solutions générées par le processus de synthèse automatique. En posant une contrainte sur le nombre d'occurrences d'un ou de plusieurs AO, l'espace des solutions est réduit ou augmenté. Il est réduit si certaines occurrences étaient nécessaires pour instancier

toutes les méta-architectures. Au contraire, l'espace de conception est étendu si l'utilisateur contraint l'occurrence d'un AO au delà du strict nécessaire pour instancier toutes les méta-architectures. L'avantage de réduire l'espace de conception est de limiter le temps de calcul qui peut être long quand trop d'AO sont dans la base de données. L'inconvénient est que l'utilisateur peut passer à côté de l'architecture optimale.

3.5.7 Définition des ports du système

Précédemment ont été définies des exigences fonctionnelles sous forme de boîtes noires. Les ports définis à ce moment-là ne sont pas forcément des ports extérieurs pour le système à concevoir. Or, il est nécessaire à cette étape de ne considérer que les interfaces extérieures pour laisser le choix des technologies à la partie automatisée du processus de synthèse. Pour ce faire, chaque exigence fonctionnelle sous forme de boîte noire est examinée, et les ports qui sont en interface avec des parties prenantes sont gardés pour définir les ports d'interface du système. La représentation sous forme de boîtes noires du système à concevoir est visible sur la Figure 3.11.

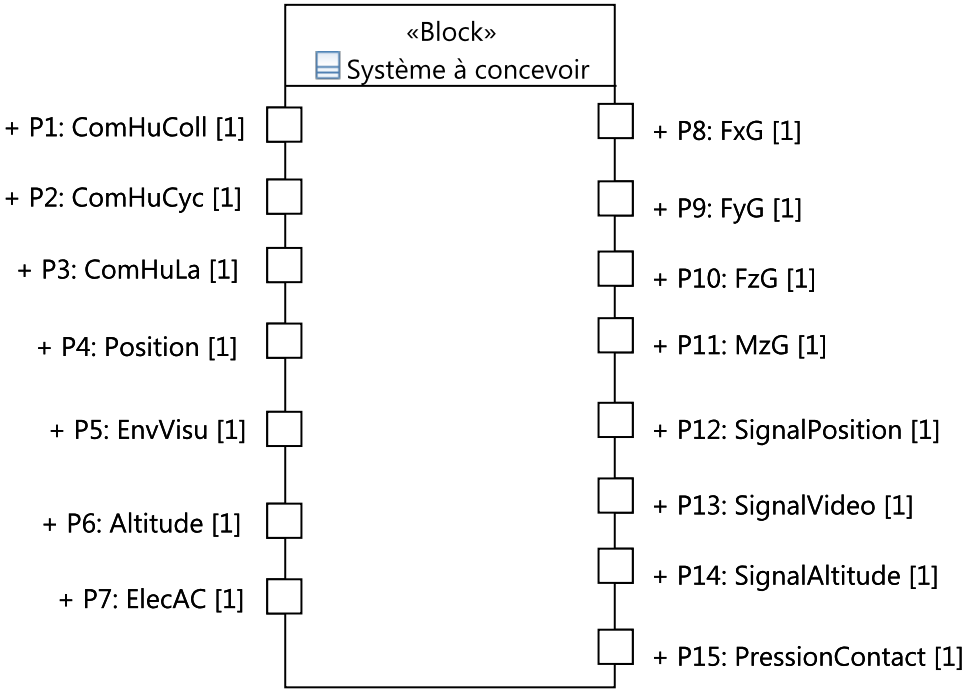


FIGURE 3.11 – Interfaces du système.

Définition des ports du système au global

Certaines parties prenantes n'ont pas été considérées comme spécifiant le système, c'est le cas par exemple de l'air. Or, lors du processus itératif de création de la base de données des AO et des ports d'interface du système, il est possible de voir qu'un port d'interface est important. Dans le cas présent, l'air permet le fonctionnement d'un moteur thermique qui sera dans la base des AO. Un port «Air» est donc ajouté aux interfaces du système au global. Notons aussi que le besoin de protéger le chargement du drone avait été identifié comme une exigence de contrainte dans le Tableau 3.2 page 60. Néanmoins, un port «Intempéries» est rajouté dans la définition des interfaces, il permettra de contraindre

l'ajout d'un AO Cellule. La représentation sous la forme d'une boîte noire de toutes les interfaces considérées du système à concevoir est donnée sur la Figure 3.12.

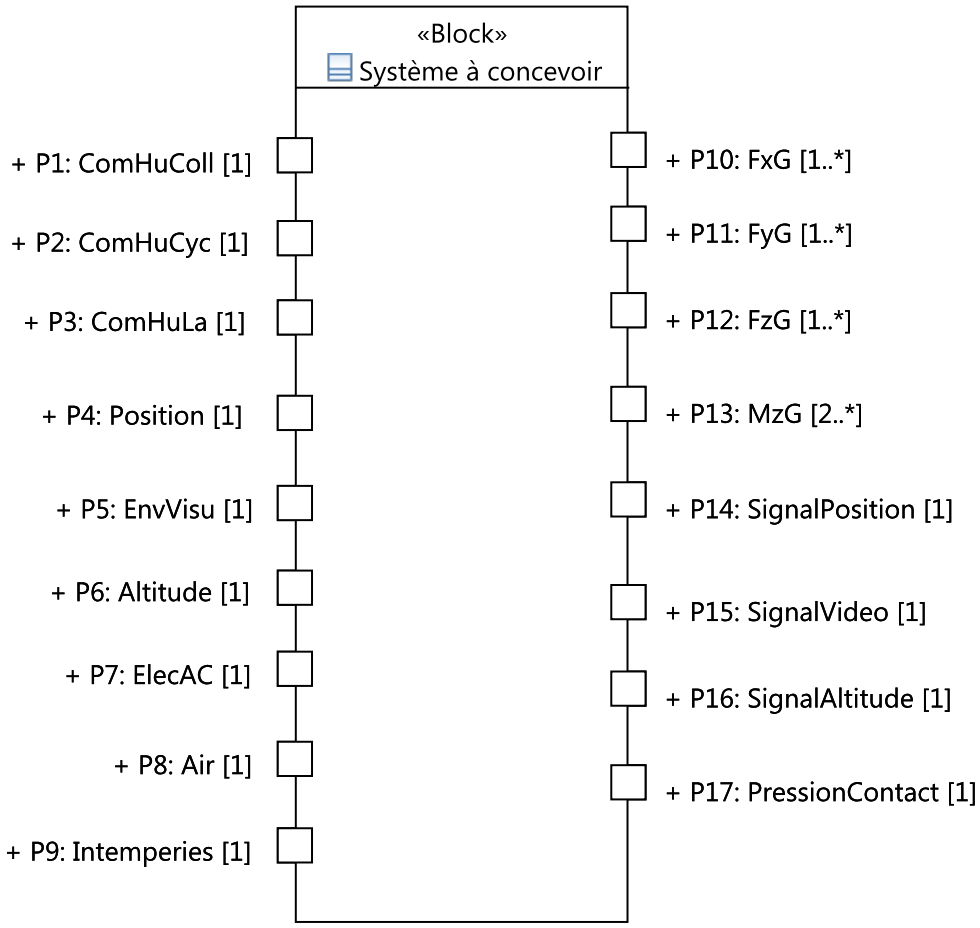


FIGURE 3.12 – Interfaces du système au global.

Définition des ports du système dans chaque situation de vie

Le système devant évoluer dans plusieurs environnements lors des différentes situations de vie qu'il va rencontrer, il est nécessaire de définir ses interfaces avec les parties prenantes dans chaque situation de vie détectée. Toutes les situations de vie dans lesquelles le système ne répond pas à des exigences fonctionnelles ne sont pas traitées par la génération automatique. Par exemple, la situation de vie «Phase de conception» visible sur la Figure 3.7 page 55 ne sera pas étudiée. Les phases de décollage, d'observation et d'atterrissage sont des phases similaires d'un point de vue interface avec l'environnement. En effet, seul un vol stationnaire est demandé au drone lors de ces phases. Les ports d'interface activés dans cette situation de vie sont visibles sur la Figure 3.14. La phase de transport correspond à un vol d'avancement dont les ports d'interface activés sont représentés sur la Figure 3.15 et la phase de rechargement à une phase au sol, rotors non tournants, visible sur la Figure 3.13.

Notons que les exigences de performance ne s'appliquent pas forcément sur toutes les situations de vie de la même façon. Par exemple la performance sur la vitesse maximale à atteindre n'est valable que pour la situation de vie n°3 correspondant à un vol d'avancement.

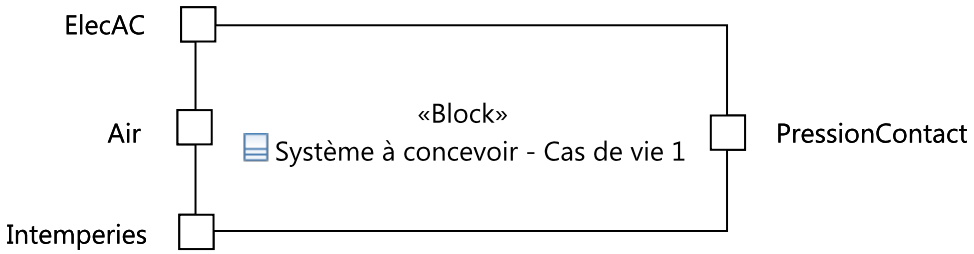


FIGURE 3.13 – Ports du système dans la situation de vie n°1.

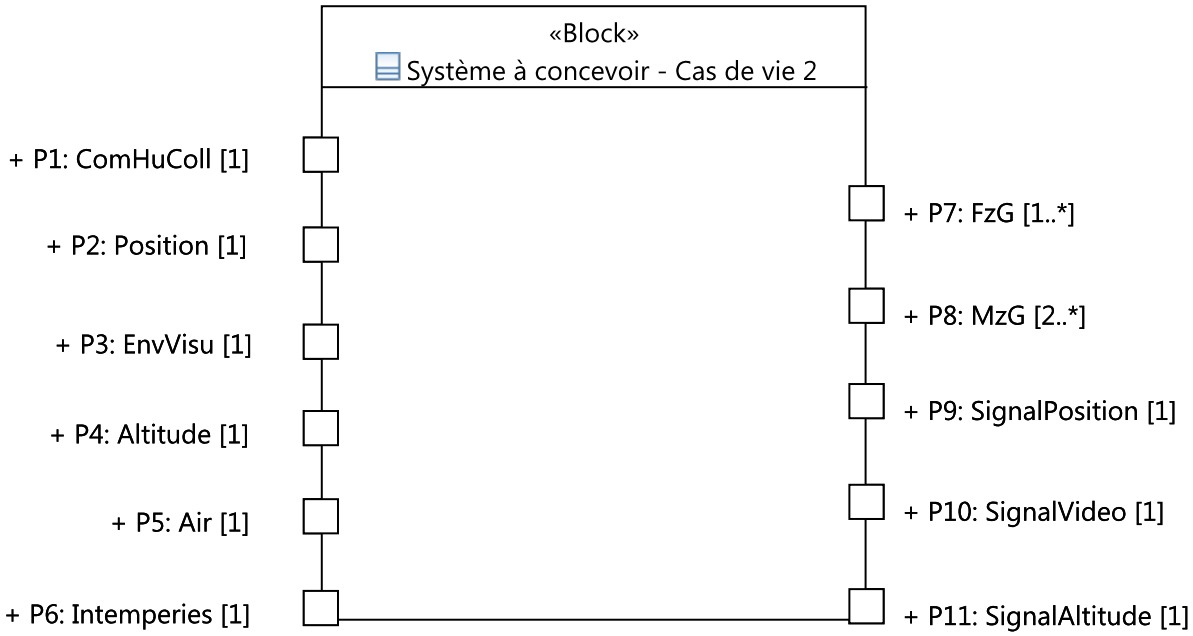


FIGURE 3.14 – Ports du système dans la situation de vie n°2.

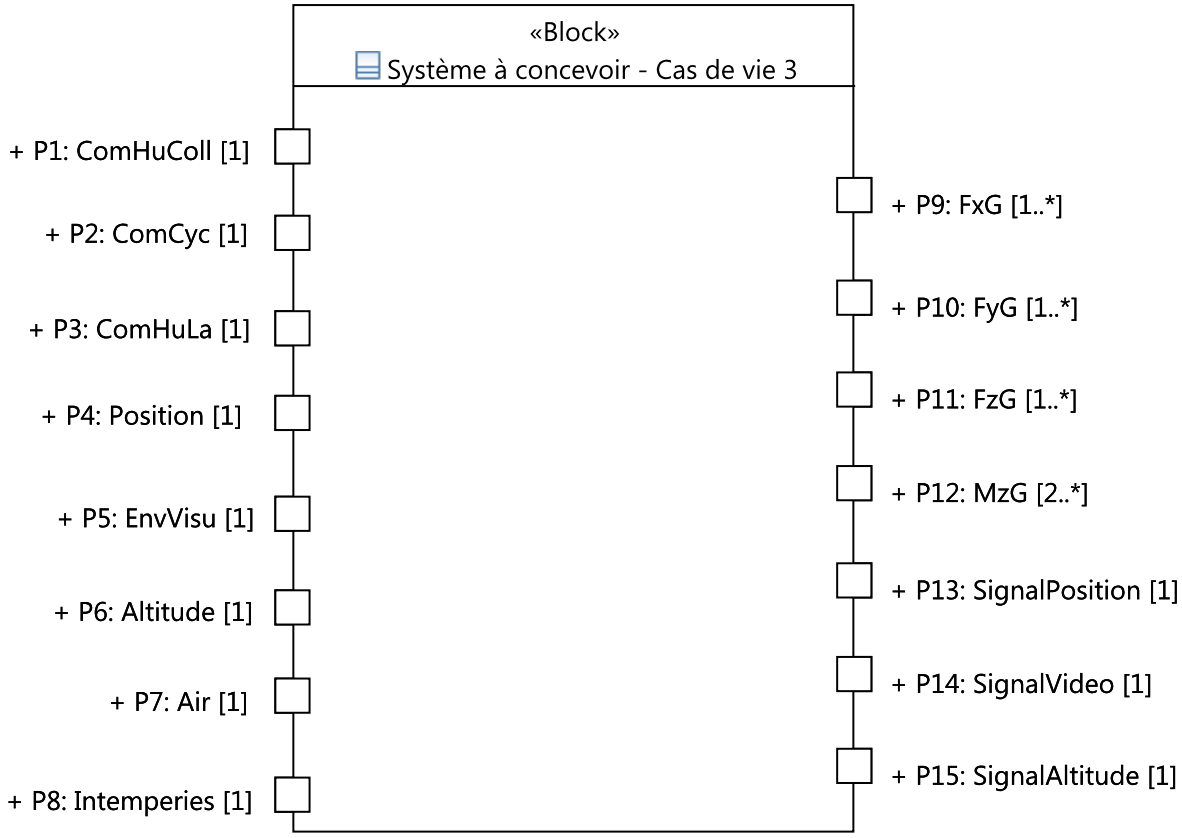


FIGURE 3.15 – Ports du système dans la situation de vie n°3.

Définition de la vue physique des ports du système

La définition physique du système à concevoir permet la prise en compte des exigences de performance. Comme vu précédemment, le système est défini par ses ports, mais aussi par des variables d'échange, contenues dans les ports, et des variables d'état. Les exigences de performance sont exprimées sous la forme de contraintes sur les variables d'échange pour des performances fonctionnelles et sur des variables d'état pour des performances non-fonctionnelles. Les variables d'échange du système drone sont présentées dans le Tableau 3.5. Les variables d'état du système drone sont présentées dans le Tableau 3.6. A noter que le port «FyG» ne contient aucune variable dans cette modélisation. En effet, les efforts dans la direction «y» ne sont pas considérés.

TABLEAU 3.5 – Définition physique des ports du système à concevoir.

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque situation de vie		Est associative ?
Port ElecAC						
PElecAC	kW	0	500	S. de vie 1 S. de vie 2 S. de vie 3	= 0 = 0 = 10	Oui
Port FxG						

continue sur la page suivante

continue depuis la page précédente

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque situation de vie		Est associative ?
FxG	N	0	100000	S. de vie 1 S. de vie 2 S. de vie 3	∅ = 0 = 0	Oui
PxG	kW	0	500	S. de vie 1 S. de vie 2 S. de vie 3	∅ = 0 = 0	Oui
Port FzG						
FzG	N	0	100000	S. de vie 1 S. de vie 2 S. de vie 3	∅ ∅ = 0	Oui
PzG	kW	0	500	S. de vie 1 S. de vie 2 S. de vie 3	∅ ∅ = 0	Oui
Port MzG						
MzG	Nm	0	50000	S. de vie 1 S. de vie 2 S. de vie 3	∅ ∅ ∅	Oui
Port PressionContact						
P	MPa	0	3	S. de vie 1 S. de vie 2 S. de vie 3	= 0 = 0 = 3	Oui

Les variables définies dans le tableau 3.5, sont explicitées maintenant :

- PElecAC : la puissance électrique reçue lors de la phase de chargement,
- FxG : les efforts appliqués sur la structure et permettant à l'aéronef d'avancer,
- PxG : la puissance consommée pour déplacer l'aéronef suivant l'axe x,
- FzG : l'effort à produire suivant l'axe z de l'aéronef,
- PzG : la puissance consommée pour produire un effort suivant l'axe z de l'aéronef,
- MzG : le couple transmis à la structure,
- P : la pression de contact entre l'aéronef et le sol.

TABLEAU 3.6 – Définition physique des variables d'état du système drone.

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque situation de vie		Est associative ?	Impose ou collecte
Système drone							
MAeroC	kg	0	5000	S. de vie 1 S. de vie 2 S. de vie 3	∅ ∅ ∅	Oui	Collecte

continue sur la page suivante

continue depuis la page précédente

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque situation de vie	Est associative ?	Impose ou collecte
MAeroI	kg	0	5000	S. de vie 1 : ∅ S. de vie 2 : ∅ S. de vie 3 : ∅	Oui	Impose
v	km/h	0	300	S. de vie 1 : = 100 S. de vie 2 : = 0 S. de vie 3 : = 0	—	Impose
CxAtot	m^2	0	5	S. de vie 1 : ∅ S. de vie 2 : ∅ S. de vie 3 : ∅	Oui	Impose
CxAElem	m^2	0	5	S. de vie 1 : ∅ S. de vie 2 : ∅ S. de vie 3 : ∅	Oui	Collecte
t	s	0	86400	S. de vie 1 : = 3600 S. de vie 2 : = 3600 S. de vie 3 : = 86400	Oui	Impose

Les variables définies dans le tableau 3.6, sont explicitées maintenant :

— MAeroC : la masse des AO qui seront collectées et sommées,

— MAeroI : la masse totale de l'aéronef qui sera imposée aux sous-systèmes et qui permettra de les dimensionner,

— v : la vitesse imposée pour l'ensemble de l'aéronef,

— CxAtot : le coefficient de traînée multiplié par la surface impliquée dans la traînée pour l'ensemble de l'aéronef, cette variable est imposée aux sous-systèmes pour les dimensionner,

— CxAElem : le coefficient de traînée multiplié par la surface impliquée dans la traînée des AO, cette variable collecte toutes les traînées des sous-systèmes.

Les équations comportementales du système à concevoir sont données ci-après par situation de vie. Ces équations permettent de spécifier le comportement global du système drone qui va devoir être instancié par un ensemble d'AO.

— **Situation de vie 1 -Vol d'avancement-** :

$$\mathrm{MA}ero\mathrm{C} <= \mathrm{MA}ero\mathrm{I}$$

La masse totale de l'aéronef spécifiée doit être supérieure ou égale à la somme des masses de ses sous-ensembles.

$$\mathrm{F}z\mathrm{G} = (\mathrm{MA}ero\mathrm{I} + \mathrm{M}structure) \times 9.81$$

L'effort de portance doit permettre de soulever la masse maximale, plus celle de la structure.

$$\mathrm{F}x\mathrm{G} = 0.5 \times 1.21 \times (\mathrm{C}x\mathrm{A}tot + 3) \times v^2$$

L'effort de propulsion doit vaincre la traînée du système drone avec sa structure à la vitesse maximale.

— **Situation de vie 2 -Vol stationnaire-** :

$$\mathrm{MA}ero\mathrm{C} <= \mathrm{MA}ero\mathrm{I}$$

La masse totale de l'aéronef spécifiée doit être supérieure ou égale à la somme des masses de ses sous-ensembles.

$$\mathrm{F}z\mathrm{G} = (\mathrm{MA}ero\mathrm{I} + \mathrm{M}structure) \times 9.81$$

L'effort de portance doit permettre de soulever la masse maximale spécifiée, plus celle de la structure.

— **Situation de vie 3 -Rechargement-** :

$$\emptyset$$

La recharge de la batterie n'est pas modélisée ici pour les architectures physiques. Un autre exemple complet est donné dans les annexes sur un groupe électrogène ou la charge de la batterie est modélisée.

3.6 Conclusion du chapitre

Toute la complexité du problème de synthèse architecturale a été ramenée au niveau de la définition du problème de conception et plus particulièrement de la spécification du système. L'utilisateur ne définit pas l'architecture interne du système, qu'elle soit fonctionnelle ou logique. Le système est vu comme une fonction de transfert entre ses entrées et ses sorties mettant en jeu ses variables d'état.
Les parties prenantes, les situations de vie et les exigences ont émergé progressivement. Ensuite, les exigences sont une traduction des besoins venant du monde du client au monde du système à concevoir. La définition des exigences sous une forme bien formatée, permet, bien entendu, de lancer une synthèse architecturale automatisée grâce au logiciel qui sera présenté dans les chapitres suivants, mais aide aussi l'architecte à faire une spécification de qualité.
Le problème de synthèse ainsi décrit se focalise sur les échanges souhaités avec l'environnement et sur le comportement global du système. Tous les échanges non souhaités ne sont pas modélisés dans la définition du système, mais apparaîtront lors de la synthèse, et plus précisément lors de l'allocation de certains AO. En effet, la modélisation du système à concevoir sous la forme d'une boîte noire se focalise sur les ports fonctionnels et non sur des ports non-désirés comme de la chaleur émise ou encore des émissions de CO_2 par exemple. Néanmoins les AO sont définis par tous leurs ports, qu'ils soient fonctionnels ou non-désirés. De ce fait, les architectures synthétisées contiendront des ports non-désirés même s'ils n'ont pas été modélisés par l'utilisateur lors de la définition du système à concevoir.

Au vu des travaux présentés dans ce chapitre, nous sommes en mesure de répondre à la question de recherche **Question 1**. L'utilisation de concepts présentés par la méthodologie CESAMES de KROB [2014] sont tout à fait en adéquation pour permettre d'identifier les

parties prenantes et les situations de vie du système à concevoir en étudiant les échanges du système vu sous la forme d'une boîte noire au travers d'un cycle de vie et ses différents cas opérationnels. Grâce à cette étude, l'émergence des besoins peut se faire de manière organisée.
La catégorisation des exigences proposé par NASA [2007] permet de transformer des exigences textuelles en exigences fonctionnelle, de performance et de contrainte. Avec l'aide des travaux de PAHL et collab. [2007] et de PAILHÈS et collab. [2009], une modélisation des exigences fonctionnelles sous forme de boîtes noires est possible. Elle permet de fournir des données d'entrée compréhensibles par l'ordinateur et qui peuvent être exprimées facilement par un architecte. Ce dernier point apporte une réponse concrète à la question de recherche **Question 2**.

Chapitre 4

La génération des architectures logiques

« *The way to get started is to quit talking and begin doing.* »

Walt Disney

Sommaire

4.1 Introduction du chapitre

La méthodologie présentée dans ce manuscrit s'appuie sur des processus mis en place par différents groupes de travail se rattachant, pour la plupart, aux concepts décrits par WIXSON [1999] et par la NASA [2007]. Un des apports des travaux présentés réside dans la sélection automatisée des AO pour la réalisation du système à concevoir. Ils sont aussi présentés par HARTMANN et collab. [2018] à l'exception près que les situations de vie n'y sont pas traitées.

Le processus de synthèse d'architectures logiques est la première phase automatisée de la méthode présentée au cours de ce manuscrit. L'objectif est ici de transformer les données utilisateur en solutions architecturales représentant le système à concevoir et respectant une logique sur les échanges de flux. Ce processus est lui-même découpé en sous-processus synthétisant déjà des familles d'architectures appelées les méta-architectures, puis calculant le nombre d'occurrences nécessaires pour chaque AO et supprimant les isomorphismes. L'ensemble de ce processus avec les différentes données transitant entre ses sous-processus est illustré par la Figure 4.1. La base de données «Exigences fonctionnelles» correspond à la définition fonctionnelle du système à concevoir. La base de données «Base de données des AO» correspond à l'ensemble des sous-systèmes disponibles pour la synthèse architecturale. La base de données «Situations de vie» correspond à la définition fonctionnelle du système à concevoir dupliquée pour chaque situation de vie et avec uniquement les ports actifs dans la situation de vie considérée. Dans la suite de ce chapitre, les contraintes seront exprimées textuellement dans le texte et seront parfois illustrées par un code écrit en Minizinc [1].

1. http://www.minizinc.org/ dernier accès le 30/10/2017

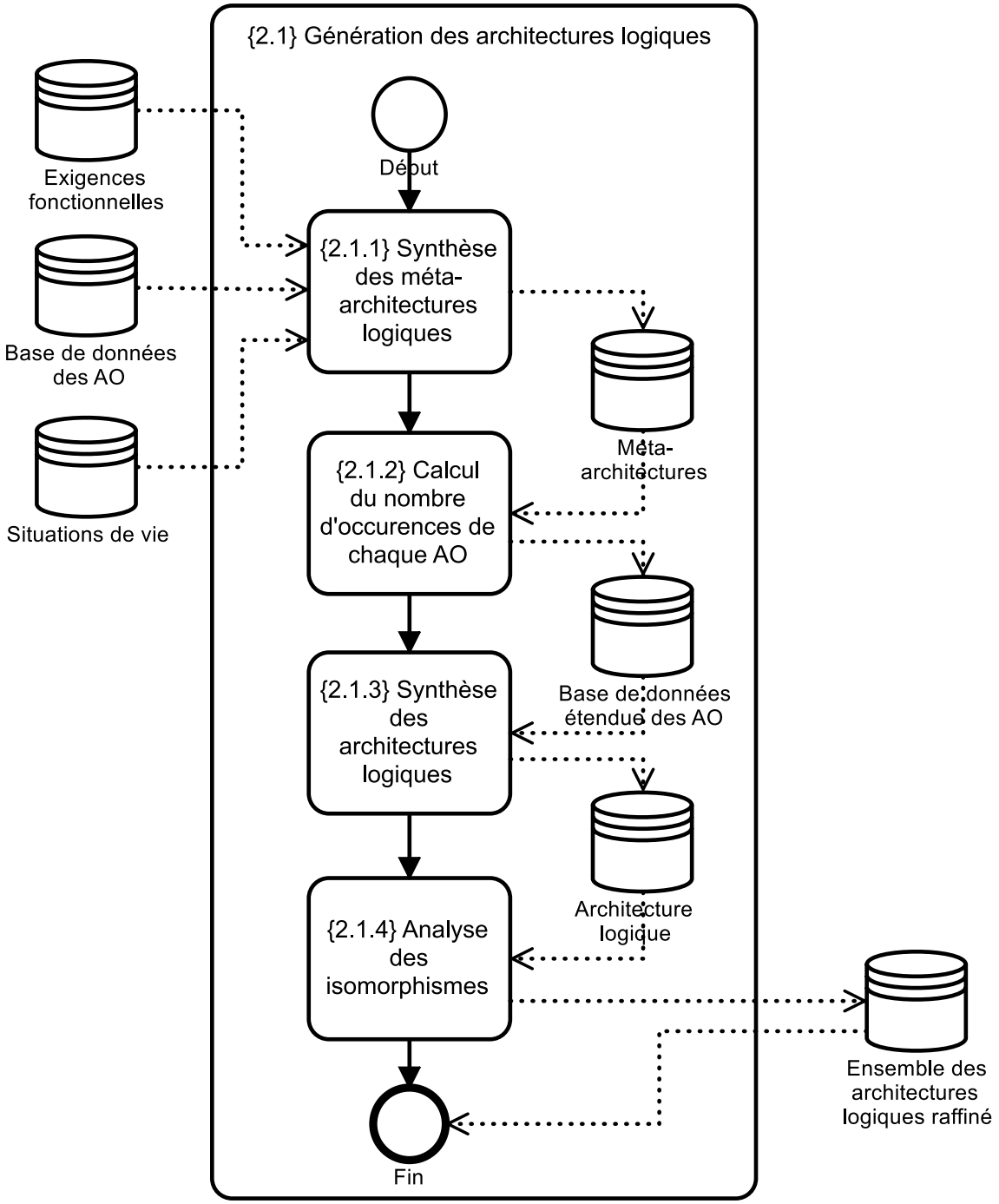


FIGURE 4.1 – Le processus de synthèse logique.

4.2 Définition des ports de la base de données

Comme présenté dans le chapitre 1.2, page 3, un AO possède pour attribut une liste de ports, qui sont aussi utilisés pour les définitions des exigences fonctionnelles sous la forme de boîtes noires.
Ici, nous nous attardons sur la définition de la base de données de ces ports. Il s'agit d'une étape clé du processus de synthèse car la suite de la résolution sera basée très fortement sur cette définition. En effet, la génération des architectures logiques est un problème d'interconnexion entre des ports. La construction de cette base se fait librement par l'utilisateur, néanmoins il est conseillé de suivre au maximum la taxonomie de HIRTZ et collab. [2002]. Il peut être intéressant, par exemple, de rajouter une distinction sur les ports suivant l'ordre de magnitude du flux qu'ils échangent. Ce point peut permettre de guider la résolution plus rapidement vers des solutions possibles. Prenons l'exemple d'un groupe électrogène devant fournir une énergie électrique alternative, cette énergie n'est pas forcément du même ordre de grandeur que l'énergie nécessaire pour activer le démarreur électrique. Ces deux flux peuvent donc être différenciés, ce qui permettra de ne pas interconnecter les mêmes ports pour des flux qui ont le même type, mais qui n'ont pas la même utilité.

Le concept d'AO aux comportements inverses est introduit maintenant pour permettre de prendre en compte une règle architecturale de base. En effet, il doit être interdit de créer dans une architecture, une boucle entre deux sous-systèmes strictement inverses. Deux AO ayant un comportement inverse ont, au moins en partie, les ports d'entrée de l'un identiques aux ports de sortie de l'autre et inversement.

Comme pour les situations de vie introduites précédemment lors de la définition du système à concevoir, à chaque AO est associé un mode de fonctionnement. Ce concept de mode de fonctionnement pour un AO est en tout point similaire à la situation de vie pour le système à concevoir. En analysant un système réel, on voit bien que certains sous-systèmes ne doivent pas fonctionner en même temps pour répondre aux situations de vie définies dans la spécification du système. Le mot clé est qu'ils ne peuvent pas «fonctionner en même temps». C'est à l'aide d'une analyse de ce type que la nécessité de prendre en compte les situations de vie du système et les modes de fonctionnement des AO apparait.

4.3 Vue logique de l'AO

Dans le précédent chapitre, nous avons étudié une partie des données d'entrée de la génération automatique : la spécification du système à concevoir. Maintenant, ce chapitre s'intéresse à la création de la base de données des sous-systèmes disponibles pour la construction des solutions. Autrement dit, nous allons définir les AO qui vont servir de blocs de construction pour la génération des architectures logiques, puis des architectures physiques et des architectures géométriques. Cette étape se fait par une analyse des systèmes similaires à celui que l'on souhaite concevoir et par ajouts de technologies nouvelles avec des caractéristiques logiques ou physiques innovantes. La qualité de la synthèse architecturale est, sans conteste, fonction de la qualité de la spécification et de la base de données des technologies appelées AO.

La vue logique est utilisée lors de la synthèse des architectures logiques. Elle ne se focalise que sur les types de ports. L'AO est un objet, qui au travers de la vue logique, a pour attributs un nom et une liste de ports ayant eux-mêmes, au travers de cette vue, une liste

d'attributs bien déterminés. Il s'agit de :

— son type correspondant au flux échangé,
— sa direction,
— le nom de l'AO auquel il appartient,
— un booléen définissant si le port doit obligatoirement être connecté ou non,
— la multiplicité minimale autorisée pour le port,
— et la multiplicité maximale autorisée pour le port.

La définition donnée ci-dessus est similaire aux modèles qui peuvent être construits à l'aide du langage SysML. Un ajout est notable, il s'agit d'un booléen. L'attribut booléen est une simple traduction de la valeur minimale de la multiplicité. Si elle vaut zéro, alors le port est non obligatoire, dans l'autre cas, il l'est. Comme dans le standard SysML, la multiplicité maximale peut être fixée à l'infini. Or, la résolution du problème de conception requiert que tous les paramètres soient finis. Pour ce faire, dès que l'utilisateur choisira une multiplicité maximale infinie, le logiciel la transformera en la valeur cent. Notons que cette valeur peut être changée à loisir dans le code source. Nous verrons plus tard que cette valeur a un impact sur le temps de résolution du problème de synthèse, même si une valeur élevée de la traduction de l'infini ne générera pas plus de solutions. Dans le reste de ce manuscrit, les multiplicités pourront être représentées sous cette forme [0..1], avec le premier nombre correspondant à la multiplicité minimale et le second à la multiplicité maximale. Si un seul nombre est entre crochets, c'est que les multiplicités minimale et maximale sont égales. Un astérisque, *, est utilisé pour représenter l'infini de cette façon : [0..*].
Il est très important de noter que chaque AO est défini pour chaque mode de fonctionnement qu'il pourrait rencontrer. Par exemple, un moteur à explosions a deux modes de fonctionnement bien différents, le démarrage et l'utilisation nominale. Pour chacun de ces cas, une surcouche de définition est ajoutée. Elle permet de savoir quels sont les ports actifs dans chaque situation.
D'un point de vue logique, les interfaces du système à concevoir sous la forme d'une boîte noire sont construites exactement de la même manière que les AO. Cela est très logique car, comme cela était déjà rappelé sur la Figure 1.4, page 7, un système peut être tout simplement vu comme un sous-système d'un système plus grand.

4.4 Les méta-architectures

Une méta-architecture est une architecture logique simplifiée car, à ce niveau de détail, les multiplicités ne sont pas prises en compte. Cela veut dire que tous les ports de même type et de directions opposées peuvent être connectés sans prendre en compte le fait que l'un des deux ports est déjà connecté plusieurs fois. Les modes de représentation utilisés pour les méta-architectures ne sont pas différents de ceux utilisés pour une vraie architecture prenant en compte les multiplicités. Comme pour les architectures logiques, ces architectures peuvent être représentées par des matrices DSM ou à l'aide de graphes. Dès cette étape, plusieurs contraintes d'association sont prises en compte, ainsi que les différentes situations de vie que le système à concevoir va rencontrer.
La résolution du problème de synthèse est faite à l'aide d'un solveur CSP. Ces types de modélisation et de solveur ont été sélectionnés pour parcourir l'ensemble des solutions architecturales de manière exhaustive et performante. Le modèle CSP est découpé en deux, une partie contenant les valeurs des paramètres du problème sous contraintes et

une partie contenant le modèle à proprement parlé.
Les paramètres sont contenus dans un fichier appelé le «fichier de données» qui est propre à un problème de synthèse donné. Le fichier de données est généré automatiquement à l'aide d'une routine, qui traduit la base de données des AO renseignée précédemment par l'utilisateur. Le modèle en lui-même est écrit dans un autre fichier. Il est statique, ce qui veut dire qu'il restera le même quel que soit le problème de synthèse à résoudre. Le processus de synthèse des méta-architectures est décrit sur la Figure 4.2.

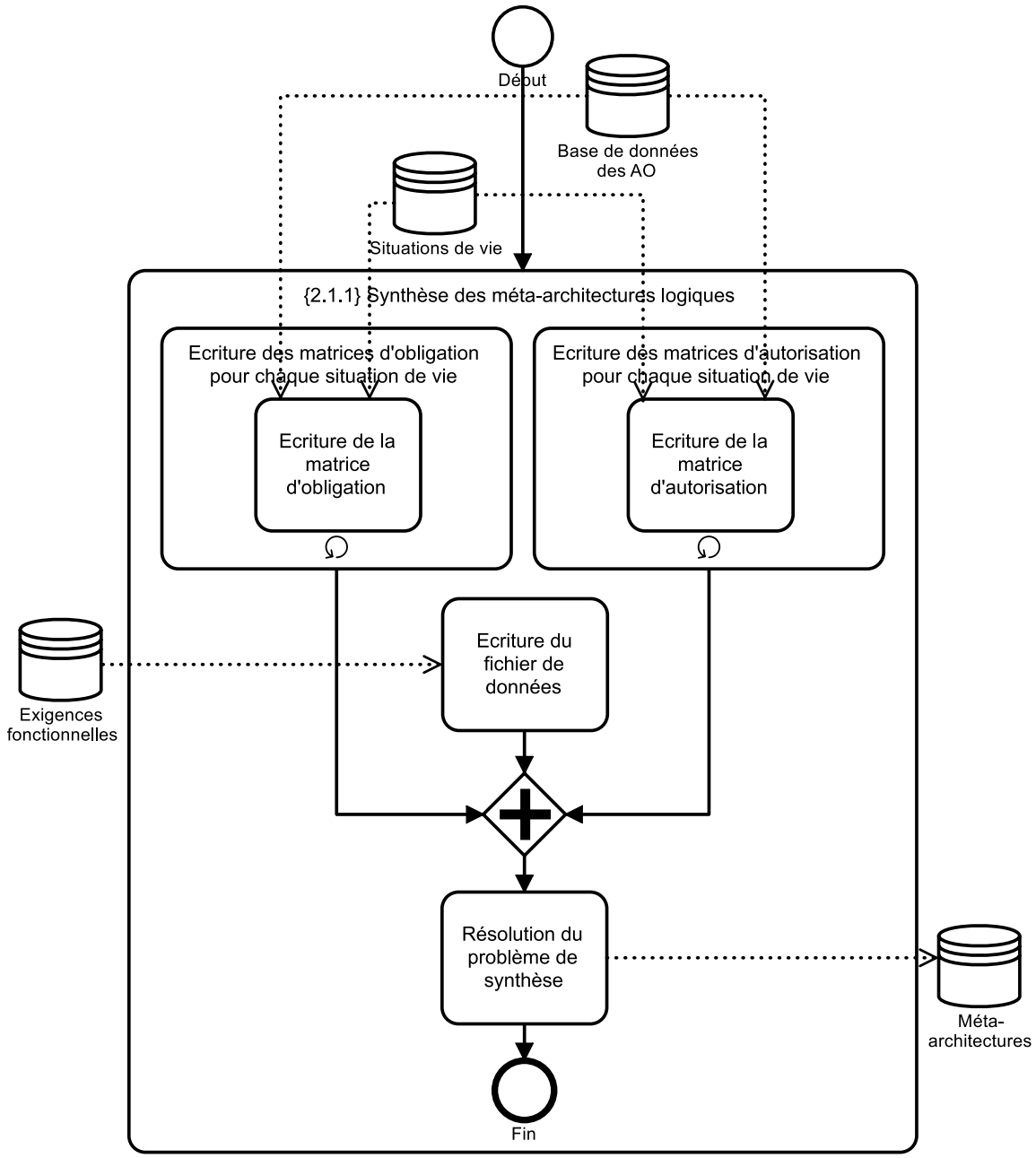


FIGURE 4.2 – Le processus de synthèse des méta-architectures.

4.4.1 Les matrices d'autorisation et d'obligation

Avant de passer à l'étape de résolution des problèmes sous contraintes, un pré-traitement est fait à l'aide d'un algorithme sur la base de données des AO. L'objectif est ici de déter-

miner quels sont les ports pouvant se connecter ensemble et de créer une matrice dite «d'autorisation», puis de détecter quels sont les ports d'un AO qui doivent être connectés pour que le composant puisse fonctionner, et enfin de créer une matrice dite «d'obligation». Ces deux matrices font parties des données renseignées dans le fichier de données introduit précédemment. En effet, ces deux matrices sont liées à la base de données des AO, elles sont bien propres à un problème de synthèse donné.

La matrice d'autorisation

La matrice d'autorisation se remplit en étudiant les connexions entre chaque port. Plusieurs règles sont contenues dans cette matrice. Premièrement, une détection des AO aux comportements inverses est menée. Deuxièmement, il est laissé au concepteur le choix d'interdire les connexions entre des AO aux comportements inverses ou non. Deux AO ayant des comportements totalement inverses, et pas seulement en partie, ne peuvent pas être connectés, même si l'utilisateur autorise la connexion des AO aux comportements partiellement inverses. Un AO ne comportant qu'un seul port n'est pas pris en compte dans cette étude des comportements inverses. En effet, si ces AO étaient aussi concernés, ils seraient forcément vus comme des composants ayant un comportement partiellement inverse d'un autre AO de la base de données.
Cette contrainte avait été introduite dans la matrice d'autorisation pour éviter la connexion entre un composant de type générateur électrique et un composant de type démarreur. En effet, ces deux composants ont des comportements inverses, étant donné que l'un transforme une énergie mécanique en énergie électrique et que l'autre fait exactement le contraire. Ces deux AO ne peuvent en effet pas être connectés ensemble car ils ne peuvent fonctionner en même temps. Néanmoins, un contre-exemple peut être trouvé avec les deux AO suivants : un conduit d'air comprimé et un compresseur. Les deux éléments ont les mêmes types de port, mais doivent pouvoir se connecter entre eux. Partant de ce contre-exemple, le choix est laissé à l'utilisateur de permettre la connexion, ou non, des composants aux comportements inverses.

Nous définissons qu'un AO ne peut pas être connecté à lui-même, donc deux ports ayant pour attribut l'appartenance au même AO ne peuvent pas être connectés. Deux ports ayant un attribut direction du flux de même valeur ne peuvent pas être connectés. Deux ports de types différents ne peuvent pas être connectés. Un exemple simple et visuel est visible sur la figure 4.3. On retrouve, en ligne et en colonne, une base de données des AO avec leurs ports. Cette matrice se lit exactement comme une matrice DSM, la différence est que pour la matrice d'autorisation, toutes les connexions possibles sont affichées. Une case contenant un «1» équivaut à une connexion possible entre deux ports. Les différentes nuances de couleur servent à identifier la contrainte qui impose la valeur de la case à «0». La seule contrainte manquante dans cet exemple simplifié de matrice d'autorisation est celle concernant les AO aux comportements inverses.

La matrice d'obligation

La matrice d'obligation se remplit en étudiant les connexions nécessaires pour qu'un AO puisse fonctionner convenablement. Factuellement, chaque ligne indique quels sont les ports qui doivent être connectés, dans le cas où le port lié à la ligne observée est connecté. Le format choisi est celui d'une matrice pour la simplicité de traitement, mais elle est composée principalement de «0». En effet, une case à «1» ne peut se trouver qu'entre deux ports appartenant au même AO. Les règles de remplissage sont très simples, il s'agit uniquement d'obliger la connexion des ports appartenant au même AO que le port étudié,

		PE				S			A			B		FS	ACDC		DCAC		UI		
		t1i	t2i	t3i	t4o	t5i	t6i	t2o	t7i	t4i	t6o	t8i	t8o	t1o	t6i	t8o	t8i	t6o	t9i	t5o	t7o
PE	t1i	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	t2i	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	t3i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	t4o	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
S	t5i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	t6i	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
	t2o	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	t7i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	t4i	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	t6o	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
B	t8i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	t8o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
FS	t1o	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ACDC	t6i	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
	t8o	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
DCAC	t8i	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
	t6o	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
UI	t9i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	t5o	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	t7o	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Même AO | Même direction | Différents types de ports

FIGURE 4.3 – Une matrice d'autorisation.

et ayant comme attribut «port obligatoire» fixé à la valeur booléenne «OUI». Pour rappel, un port est dit obligatoire si sa cardinalité minimale est supérieure à zéro. Cela signifie qu'il doit être connecté dans une solution architecturale si son AO père est utilisé.

Un exemple sur un AO «PE» arbitraire est illustré avec la figure 4.4. Par exemple, si le port «t1i» est connecté, on peut voir sur sa ligne que tous les autres ports de ce même AO doivent être connectés. Cela signifie que tous les ports de cet AO sont obligatoires.

		PE			
		t1i	t2i	t3i	t4o
PE	t1i	0	1	1	1
	t2i	1	0	1	1
	t3i	1	1	0	1
	t4o	1	1	1	0

FIGURE 4.4 – Une matrice d'obligation pour un AO.

4.4.2 Les règles d'interconnexion

Nous avons vu précédemment que quelques règles d'interconnexion propres à la base de données des AO sont définies dans les matrices d'autorisation et d'obligation. A l'intérieur même des codes CSP, d'autres règles sont ajoutées sous la forme de contraintes mathématiques. Pour rappel, à ce niveau du processus de synthèse architecturale, nous générons des méta-architectures. Nous travaillons donc sur des graphes traités comme des matrices DSM, ce qui nous permet d'utiliser du calcul matriciel simple.
Sont définies en sortie du code CSP quatre matrices :

— une matrice DSM d'interconnexion entre les AO,

— une matrice DSM d'interconnexion entre les AO et les ports d'entrées du système à concevoir, définis comme exposé dans le chapitre 3.2, page 47,

— une matrice DSM d'interconnexion entre les AO et les ports de sortie du système à concevoir, définis comme précédemment,

— et une matrice indiquant, pour chaque situation de vie considérée pour le système à concevoir, quels modes de fonctionnement pour les AO sont considérés.

Les trois matrices DSM sont respectivement appelées DSM d'interconnexion, DSM d'entrée et DSM de sortie. Chacune des cases de ces matrices sont des variables de décision du modèle CSP. Les variables peuvent prendre deux valeurs : un ou zéro, avec la valeur un représentant une connexion et zéro représentant une non-connexion.
Les contraintes appliquées assurent que tous les ports d'entrées et de sorties du système sous la forme d'une boîte noire sont bien connectés à des AO de la solution architecturale. Tous les ports de la base de données des AO ayant un attribut «direction» égal à« ENTREE» peuvent être connectés à un port d'entrée du système à concevoir. Tous les ports de la base de données des AO ayant un attribut "direction" égal à «SORTIE» peuvent être connectés à un port de sortie du système à concevoir. La matrice d'autorisation est appliquée aux solutions architecturales, c'est-à-dire aux matrices DSM d'interconnexion entre les AO. Cela signifie que si une connexion est interdite par la matrice d'autorisation, aucune solution architecturale ne pourra contenir cette connexion, comme cela est illustré par la contrainte 4.1.

```
%---------------------------
% Parametres
%---------------------------
% Le nombre de ports presents dans la base des AO
int: nb_interfaces_in_database;
% La matrice d'autorisation
array[1..nb_interfaces_in_database,1..nb_interfaces_in_database] of 0..1: authorized_matrix;
%---------------------------
% Variables
%---------------------------
% Une solution architecturale sous la forme d'une matrice DSM
array [1..nb_interfaces_in_database, 1..nb_interfaces_in_database] of var 0..1: DSM;
%---------------------------
% Contraintes
%---------------------------
% Les cases de la matrice DSM sont toujours inferieures ou egales aux cases de la matrice d'autorisation. Comme cela, si une connexion est interdite, la case de la DSM sera egale a zero.
constraint
forall(i in 1..nb_interfaces_database)
(
  forall(j in 1..nb_interfaces_database)
  (
    matrice_autorisation[i,j]>=DSM[i,j]
  )
);
```

Code 4.1 – Application de la matrice d'autorisation

Si un port est connecté dans une des trois DSM, alors la matrice d'obligation s'applique, et tous les ports du même AO ayant leur attribut «obligatoire» égal à «OUI» sont connectés. Un composant ayant au moins un port avec un attribut «direction» égal à «ENTREE» connecté doit au moins avoir un port avec un attribut «direction» égal à «SORTIE» connecté. Sinon, l'AO est considéré comme inutile et n'est donc pas intégré à la solution architecturale. Cette dernière contrainte est illustrée par la contrainte 4.2.

```
%--------------------------
% Parametres
%--------------------------
% Un vecteur pour savoir si le port est de sortie.
array[1..nb_interfaces_in_database] of 0..1: out_available;
% Un vecteur pour connaitre le type d'AO auquel le port appartient.
array[1..nb_interfaces_in_database] of int: components;
%--------------------------
% Variables
%--------------------------
% Une matrice DSM entre les AO de la solution architecturale et l'environnement.
array [1..nb_outputs_function, 1..nb_interfaces_in_database] of var 0..1: output_function_connections;
%--------------------------
% Contraintes
%--------------------------
% Si un port d'entree d'un AO est connecte, alors au moins un port de sortie de ce meme AO doit etre connecte, sinon il est inutile et retire de la solution.
constraint
% On boucle sur les lignes de la matrice DSM.
forall (i in 1..nb_interfaces_in_database)
(
  % On boucle sur les colonnes de la matrice DSM.
  forall (j in 1..nb_interfaces_in_database)
  ( % On somme les connexions des ports de sortie de l'AO contenant le port ''i''. Le bool2int est utilise pour faire une fonction ''si''.
    DSM[i,j]*(sum(k in j+1..nb_interfaces_in_database)(bool2int(components[k] == components[j])*out_available[k]*(sum(l in 1..nb_interfaces_in_database)(DSM[k,l]) + sum(m in 1..nb_outputs_function)(output_function_connections[m,k])))) >= DSM[i,j]
  )
);
```

Code 4.2 – Application de la matrice d'obligation

Les contraintes explicitées au-dessus ne prennent pas en compte les différences comportementales induites par l'introduction des différentes situations de vie. En effet, les ports d'entrée et sortie ne sont pas forcément attendus tout le temps, suivant la situation de vie étudiée. Les contraintes énoncées maintenant s'appuient sur les matrices DSM précédemment introduites. Par de simples multiplications, elles identifient les modes dans lesquels chaque AO doit être utilisé pour satisfaire les exigences décrites par l'utilisateur lors de la définition des situations de vie, comme décrit dans le chapitre 3.4, page 52.
Premièrement, une contrainte enregistre, pour chaque situation de vie considérée, le mode de l'AO qui est étudié et ce pour chaque AO. Deuxièmement, on fait attention à ce que chaque port normalement connecté, sans prendre en compte les situations de vie, soit au moins connecté une fois dans une situation de vie, dans n'importe laquelle des trois DSM, qu'elle soit d'interconnexion, d'entrée ou de sortie. Une contrainte fait en sorte que chaque port recevant des connexions dans une situation de vie donnée du système à concevoir, soit actif dans cette situation. Cette dernière contrainte est donnée en exemple sur le code source 4.3. Enfin, tous les ports appartenant à un même AO doivent être considérés dans le même mode pour une situation de vie donnée. Par exemple, en considérant l'AO moteur à explosions, si un port est utilisé dans le mode «DÉMARRAGE», alors tous les ports de ce composant doivent être considérés dans le même mode «DÉMARRAGE».

```
%--------------------------
% Parametres
%--------------------------
% Une matrice qui permet de savoir pour chaque AO quels sont ses ports actifs dans chacun de ses modes de fonctionnement.
array[1..index_max_Muc[nb_AO],1..nb_interfaces_in_database] of int: Muc;
% Une matrice qui permet de savoir, pour chaque interface avec l'exterieur du systeme a concevoir, quels sont les ports actifs dans chaque situation de vie du systeme.
array[1..nb_uc,1..nb_outputs_function] of int: Muc_output;
%--------------------------
% Variables
%--------------------------
% Un vecteur qui garde en memoire les modes selectionnes pour chaque AO.
array[1..nb_uc,1..nb_AO] of var int: Muc_i;
%--------------------------
% Contraintes
%--------------------------
constraint
% Pour chaque situation de vie :
forall (n in 1..nb_uc)(
```

```
19   % Pour chaque port de sortie du systeme a concevoir ↩
        sous forme de boite noire :
20   forall(i in 1..nb_outputs_function)(
21     % Si le port est connecte dans la situation de vie ''↩
         n'', alors il doit etre dans un mode actif.
22     sum(j in 1..nb_interfaces_in_database)(↩
         output_function_connections[i,j]*Muc[Muc_i[n,↩
         components[j]],j])>=Muc_output[n,i]*bool2int(sum(j ↩
         in 1..nb_interfaces_in_database)(↩
         output_function_connections[i,j])>=1)
23   )
24 );
```

Code 4.3 – Un port connecté dans une situation de vie doit être actif dans cette situation de vie.

4.4.3 La résolution du problème de synthèse

L'ensemble de ces contraintes est appliqué sur l'espace des solutions. Chaque méta-architecture est compatible avec toutes les contraintes, et inversement, aucune autre solution n'est compatible avec les contraintes. La synthèse est exhaustive, mais finie. En effet, lors de cette étape, les multiplicités ne sont pas considérées, ce qui implique que les occurrences des AO sont toutes fixées à un. De ce fait, l'ensemble des solutions est fini, étant donné que l'étude porte sur une base de données des AO finie.

Comme dit précédemment, la résolution de ce problème de synthèse est faite à l'aide d'un langage et d'un solveur CSP. Les contraintes sont écrites avec le langage Minizinc, dont la définition vient de MARRIOTT et collab. [2014]. Notons qu'il ne s'agit pas d'un problème d'optimisation, mais uniquement d'un problème de résolution de contraintes. Le sous-processus finira donc avec un ensemble de solutions et non pas une solution unique optimale. Les variables de décision qui ont été créées par le solveur sont ensuite traitées à l'aide d'un code qui va permettre un affichage sous forme de graphes à l'aide du logiciel Graphviz[2].

4.5 Les occurrences des objets architecturaux

A la sortie de la sous-étape précédente, plusieurs méta-architectures, ou encore familles d'architectures logiques, ont été générées. Pour rappel, dans l'étape précédente, les multiplicités ne sont pas considérées. Cette étape permet, par une analyse des méta-architectures, de prendre en compte les multiplicités et les exigences de contrainte définies dans le chapitre 3.3, page 49, pour définir les occurrences nécessaires ou attendues par l'utilisateur sur chaque AO. Cette analyse est faite à l'aide d'un modèle CSP pour éviter d'avoir à définir un ordre de résolution dans l'algorithme. En effet, l'introduction d'occurrences supérieures au nombre «un» va causer la redéfinition du problème de départ, et donc cela se prête bien à une résolution à l'aide de modèles CSP. Par exemple, sur l'illustration 4.5, le calcul de l'occurrence de l'«AO 1» va faire apparaître de nouvelles connexions en amont de cet AO.

2. http://www.graphviz.org/ dernier accès le 31/10/2017

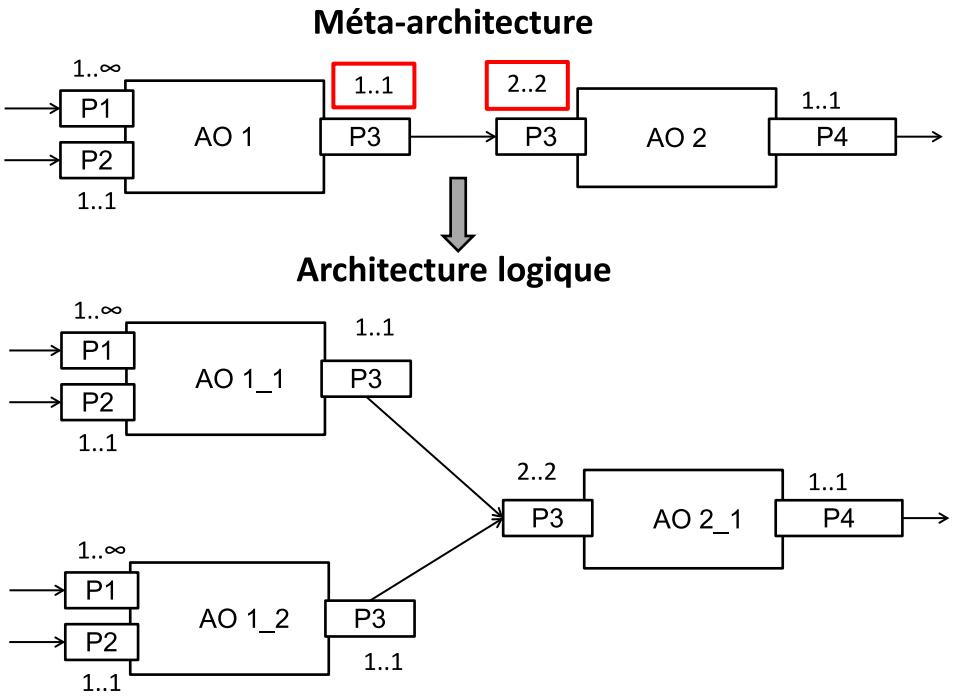


FIGURE 4.5 – Calcul des occurrences de chaque AO.

4.5.1 La prise en compte des exigences de contrainte

Lors de la définition des exigences, l'utilisateur a défini des exigences dites «de contrainte» qui, pour certaines, vont forcer le nombre d'occurrences de certains AO. Par exemple, l'utilisateur peut imposer au logiciel de n'obtenir que des solutions architecturales contenant deux moteurs à explosions sous la forme d'une exigence de contrainte, qui est ensuite traduite par une contrainte mathématique dans le modèle CSP permettant la détermination des occurrences.
Concrètement, l'utilisateur, après avoir défini son système à concevoir, comme expliqué au chapitre 3, page 43, va lancer le logiciel de synthèse automatique. Premièrement, il lui sera demandé s'il autorise la connexion des AO aux comportements inverses, comme introduit dans la section 4.4.1, page 81, et juste ensuite, il pourra renseigner toutes les occurrences à imposer à l'aide d'une borne minimale et d'une borne maximale. Si aucune occurrence n'est imposée pour un AO, le logiciel choisira la plus faible occurrence pour que la méta-architecture devienne viable, comme cela est le cas sur la Figure 4.5.
Une combinatoire est ensuite réalisée entre les différentes occurrences imposées sur les AO pour générer autant de modèles CSP que nécessaire. Par exemple, si l'«AO 1» reçoit une occurrence entre «zéro» et «un» par l'utilisateur, et l'«AO 2» entre «un» et «deux», quatre modèles seront créés grâce à la combinatoire.

4.5.2 La prise en compte des multiplicités

Pour chaque méta-architecture, un modèle CSP est créé. Les paramètres d'entrée de ce modèle sont les multiplicités admissibles de chaque port, qu'il soit port d'un AO ou port de l'environnement extérieur au système à concevoir, ainsi que les DSM d'interconnexion, d'entrée et de sortie de la méta-architecture considérée. Ce modèle fournira, après résolution, le nombre d'occurrences de chaque AO.
Un fichier de données Minizinc est créé par méta-architecture. En effet, le calcul des occurrences est propre à chaque famille de solutions et donc à un triplet de DSM d'inter-

connexion, d'entrée et de sortie. Pour rappel, une architecture logique est composée de ces trois DSM, qui représentent les connexions entre les AO et l'environnement du système.
L'objectif est donc, ici, d'avoir un fichier de données par triplet de DSM, car il y aura un ensemble d'occurrences par triplet de DSM et donc par méta-architecture.
Par exemple, prenons deux familles d'architectures différentes et intéressons-nous à l'occurrence d'un de ses AO. La famille des hélicoptères conventionnels aura une occurrence de l'AO rotor principal à «un», alors que la famille des hélicoptères quadrotor aura l'occurrence de ce même AO à «zéro». Dans les deux cas, les DSM ne sont pas les mêmes, c'est pour cette raison que les occurrences sont différentes.

Plusieurs contraintes permettent de connaître le nombre de connexions que reçoit chaque port. Ce nombre est aussi appelé cardinalité. La cardinalité est calculée avec des sommes sur les lignes ou les colonnes des DSM, suivant si le port est typé d'entrée ou de sortie. Cela permet, en effet, de calculer le nombre de connexions dans lesquelles le port est impliqué.
La cardinalité de chaque port est donc calculée, puis, en comparant ce résultat de cardinalité aux multiplicités minimale et maximale admissibles par le port, l'occurrence du port en est déduite. Or, un AO a forcément la même occurrence que tous ses ports. Une contrainte homogénéise donc les occurrences des différents ports pour avoir l'occurrence de chaque AO égale au maximum des occurrences de ses ports pris individuellement. Ce processus est illustré sur la Figure 4.5. Si un AO n'est pas connecté dans la méta-architecture, alors son occurrence est fixée à «zéro», pour qu'il ne soit pas pris en compte dans la base de données étendue, qui sera utilisée pour générer les architectures logiques. Si un AO obtient une occurrence de «deux», par exemple, cela veut dire que deux AO du même type seront connectés suivant les connexions décrites dans la méta-architecture. Cela aura donc un impact sur les cardinalités des ports des autres AO auxquels l'AO ayant une occurrence de «deux» est connecté dans la méta-architecture. Le même raisonnement est donc appliqué sur ces AO. Le travail de détermination des occurrences des AO est récursif, c'est pour cela qu'il a été modélisé sous la forme d'un CSP.
Notons que ce problème est un problème d'optimisation, où il est demandé de minimiser le nombre d'AO dans la solution architecturale, dans le but d'obtenir, pour chaque méta-architecture, la configuration la plus simple.

4.6 Les architectures logiques

Une fois que les méta-architectures sont définies, ainsi que les occurrences de chaque AO de la base de données, la génération des architectures logiques peut commencer.
Une base de données étendue des AO est créée à partir des résultats obtenus sur les occurrences sur toutes les méta-architectures. Pour ce faire, l'occurrence maximale de chaque AO pour toutes les méta-architectures est enregistrée, puis on définit une base de données étendue des AO, dans laquelle chaque AO est présent le même nombre de fois que son occurrence maximale. Notons que si plusieurs AO ont une occurrence à zéro, cette base étendue peut être, en fait, plus limitée que pour la synthèse des méta-architectures.
La phase de synthèse des architectures logiques commence donc avec les mêmes entrées que pour la synthèse des méta-architectures. Les différences sont le changement de base de données et la prise en compte des cardinalités réelles des ports. Précédemment, la base de données réduite était utilisée, maintenant nous utilisons la base de données issue de l'analyse précédente des occurrences.

4.6.1 Les nouvelles matrices d'autorisation et d'obligation

Les matrices d'autorisation

Les matrices d'autorisation sont similaires à celles générées précédemment pour la synthèse des méta-architectures. La différence est que toutes les connexions qui seraient possibles au regard de la base de données étendue des AO, mais qui ne sont pas présentes dans la méta-architecture mère, ne sont pas autorisées. Cela permet de segmenter la génération des architectures logiques en séparant bien les architectures émanant de familles différentes. En effet, lors de la synthèse des architectures logiques, une résolution est lancée par méta-architecture. Cela permet de ne pas traiter toutes les solutions déjà rejetées par la première étape de synthèse.

Une nouveauté est introduite à cette étape, il s'agit de l'utilisation de matrices d'autorisation d'entrée et de sortie. Ces matrices sont en fait les DSM d'entrée et de sortie associées à chaque méta-architecture. Elles permettent de n'autoriser que des connexions vers l'environnement qui étaient présentes dans la méta-architecture considérée.

Les matrices d'obligation

Les matrices d'obligation utilisées pour la génération des architectures logiques sont créées de la même façon que celles utilisées pour la génération des méta-architectures. La seule différence est que la matrice d'obligation est construite à partir de la base de données étendue considérée et correspondant à la méta-architecture étudiée, exactement de la même façon que pour les matrices d'autorisation.

4.6.2 La synthèse des architectures logiques par méta-architecture

Un modèle CSP très proche de celui utilisé pour la génération des méta-architectures est créé. Y sont seulement rajoutées des contraintes sur les cardinalités des ports devant être comprises entre ses multiplicités minimale et maximale et des contraintes permettant d'éviter d'obtenir des connexions ne correspondant pas à la famille d'architectures considérée. En effet, dans chaque méta-architecture, ce n'est pas forcément le même AO qui fournit le flux nécessaire à la sortie attendue du système à concevoir, idem pour les flux attendus en entrée et les flux internes. Pour ce faire, des contraintes restreignent l'espace des possibles en interdisant les connexions entre AO non connectés dans la méta-architecture étudiée, comme présenté dans les deux sections précédentes.
Les contraintes assurant que les multiplicités des ports sont bien respectées demandent l'introduction de la cardinalité réelle de chaque connexion. Par rapport aux modèles CSP précédemment décrits pour les méta-architectures, les modèles ici présents voient les paramètres et variables présentés dans le code source 4.4 ajoutés. Il s'agit de variables définissant pour chaque connexion possible entre deux ports, la cardinalité réelle.
Prenons un exemple concret : la connexion entre deux batteries et trois moteurs électriques. Ces deux types d'AO peuvent être connectés un certain nombre de fois. Disons qu'une batterie peut être connectée jusqu'à deux fois à un récepteur et qu'un moteur électrique peut recevoir de l'énergie de trois sources différentes. Dans ce cas, une même batterie peut être connectée deux fois à un même moteur pour une question de redondance, au cas où un des deux câbles électriques céderait. Mais dans ce cas, la batterie serait déjà totalement prise par un seul moteur et ne pourrait pas être connectée à un deuxième moteur, comme illustré sur la Figure 4.6. Cela illustre bien l'obligation de connaître, pour chaque connexion entre deux ports, la cardinalité réelle de la connexion afin de ne pas surconnecter un AO

dans une architecture logique.

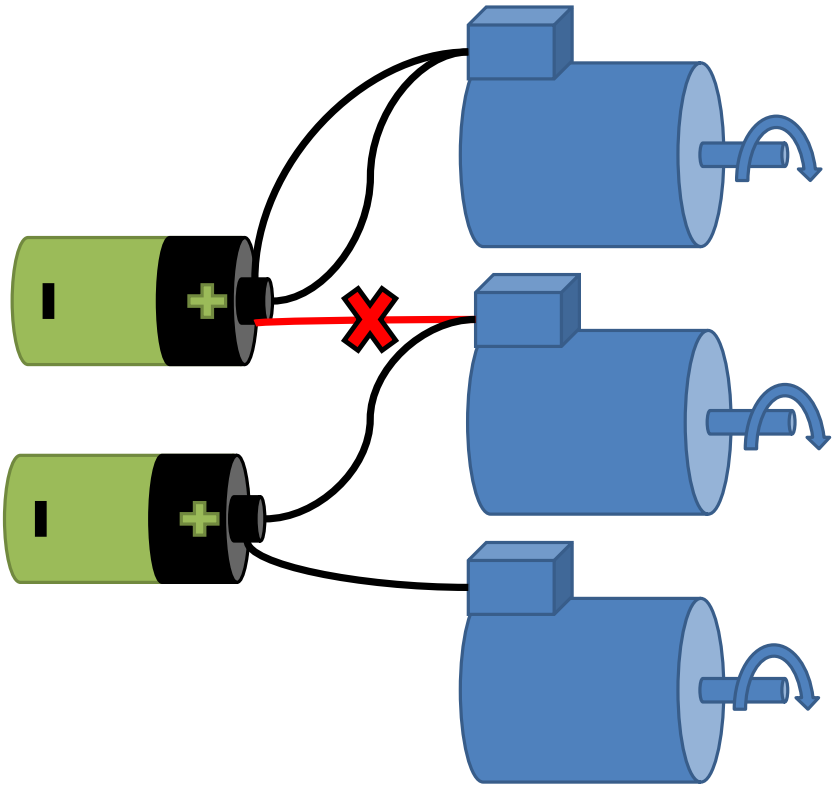


FIGURE 4.6 – De l'importance de connaître les cardinalités de connexion.

```
%---------------------------
% Parametres
%---------------------------
% Les multiplicites max admissibles pour chaque port de la base de donnees
array[1..nb_interfaces_in_database] of int: upper_cardinality_port;
% Les multiplicites min admissibles pour chaque port de la base de donnees
 array[1..nb_interfaces_in_database] of int: lower_cardinality_port;
% Les multiplicites max admissibles pour chaque port d'entree du systeme vu sous forme de boite noire
array[1..nb_inputs_function] of int: upper_cardinality_input;
% Les multiplicites min admissibles pour chaque port d'entree du systeme vu sous forme de boite noire
array[1..nb_inputs_function] of int: lower_cardinality_input;
% Les multiplicites max admissibles pour chaque port de sortie du systeme vu sous forme de boite noire
array[1..nb_outputs_function] of int: upper_cardinality_output;
```

```
14 % Les multiplicites min admissibles pour chaque port de sortie du systeme vu sous forme de boite noire
15 array[1..nb_outputs_function] of int: lower_cardinality_output;
16
17 % La DSM d'entree de la meta-architecture mere
18 array[1..nb_inputs_function,1..nb_interfaces_in_database] of 0..1: input_authorized_matrix;
19 % La DSM de sortie de la meta-architecture mere
20 array[1..nb_outputs_function,1..nb_interfaces_in_database] of 0..1: output_authorized_matrix;
21 %--------------------------
22 % Variables
23 %--------------------------
24 % Le vrai nombre de connexions entre un port d'entree du systeme et un port de la base de donnees
25 array[1..nb_inputs_function,1..nb_interfaces_in_database] of var int: real_card_input;
26 % Le vrai nombre de connexions entre un port de sortie du systeme et un port de la base de donnees
27 array[1..nb_outputs_function,1..nb_interfaces_in_database] of var int: real_card_output;
28 % Le vrai nombre de connexions entre deux ports de la DSM d'interconnexion
29 array[1..nb_interfaces_in_database,1..nb_interfaces_in_database] of var int: real_card_port;
30 % Le vrai nombre de connexions entre un port d'entree du systeme et un port de la base de donnees pour chaque situation de vie
31 array[1..nb_uc,1..nb_inputs_function,1..nb_interfaces_in_database] of var int: real_card_input_uc;
32 % Le vrai nombre de connexions entre un port de sortie du systeme et un port de la base de donnees pour chaque situation de vie
33 array[1..nb_uc,1..nb_outputs_function,1..nb_interfaces_in_database] of var int: real_card_output_uc;
34 % Le vrai nombre de connexions entre deux ports de la DSM d'interconnexion pour chaque situation de vie
35 array[1..nb_uc,1..nb_interfaces_in_database,1..nb_interfaces_in_database] of var int: real_card_port_uc;
```

Code 4.4 – Nouveaux paramètres et variables

Comme cela est visible dans la déclaration des variables dans le code source 4.4, les cardinalités réelles de connexion sont définies pour l'architecture au global et pour l'architecture dans chaque situation de vie, en sachant que les cardinalités réelles de connexion de chaque situation de vie, doivent être inférieures ou égales à celles calculées pour l'architec-

ture au global.

Revenons au problème de la multiplicité maximale infinie. Comme exposé précédemment, la résolution étant finie, il est nécessaire de transformer la multiplicité infinie en un nombre fini. De base, la valeur cent a été proposée. Néanmoins, cette valeur doit être réfléchie car elle peut fortement augmenter le temps de calcul pour la synthèse des architectures logiques si deux AO ayant des multiplicités infinies sont connectés, comme introduit dans la section 4.9.3, page 95. En effet, comme exposé dans le code source 4.4, les cardinalités réelles sont, cette fois-ci, calculées. Or, si deux AO ayant des multiplicités infinies sont connectés, et si cette même multiplicité est définie comme étant transformée en cent, alors il y aura une solution calculée, mais non affichée, pour les cent connexions, donc cent solutions calculées pour une seule solution affichée. Il est à noter que toutes les variables du problème CSP qui ne sont pas affichées à la fin, et donc non récupérées, ne génèrent pas de nouvelles solutions au problème de synthèse. Néanmoins, elles font partie intégrante du problème mathématique et peuvent varier dans tout leur domaine en prenant un temps de calcul non négligeable.
A la sortie de ce processus de génération, le logiciel fournit un ensemble d'architectures logiques représentées à l'aide du langage Graphviz sous forme de plusieurs graphes MIMO. Le premier graphe représente toutes les interfaces entre les AO de la solution et les autres graphes représentent la même solution, mais dans chaque situation de vie. Chaque AO est donc affiché dans le mode dans lequel il est utilisé dans cette situation de vie et seules les interfaces actives sont affichées.

4.7 La détection des isomorphismes

4.7.1 Détection d'isomorphismes sur des graphes MIMO

Pour générer les architectures logiques, des bases de données étendues d'AO ont été créées. Ces bases peuvent donc contenir deux AO de même type, mais de noms différents. Ces deux AO ont le même comportement et peuvent donc être interchangés au sein d'une architecture logique, comme illustré sur la Figure 4.7. Notons que cela n'est pas forcément vrai dans le cadre d'une architecture physique, où deux AO de même type peuvent être instanciés différemment. Si deux AO peuvent être interchangés sans modifier l'architecture physique, il faut savoir que la génération à l'aide du modèle CSP ne verra pas cette similarité et générera bien deux solutions différentes. Ces deux solutions sont dites «isomorphiques» et sont détectées après la génération à l'aide d'un algorithme dédié s'appuyant sur la théorie présentée par MERRIS [1994]. Les algorithmes de détection d'isomorphismes dans la littérature ne sont pas adaptés aux architectures logiques qui sont traitées dans ce manuscrit car il s'agit, dans notre cas, de graphes MIMO. De plus, il est nécessaire de garder les architectures logiques identiques d'un point de vue des connexions au global, mais avec des arrangements différents suivant la situation de vie considérée.

L'algorithme considère les solutions deux à deux pour trouver les isomorphismes. On recherche plutôt les non-isomorphismes car cela est plus simple à implémenter et surtout, la résolution est beaucoup plus rapide. Pour rappel, les architectures logiques sont générées par familles, suivant une méta-architecture mère. La détection des isomorphismes se fait, dans un premier temps, à l'intérieur d'une même famille et entre deux solutions qui ont été générées l'une après l'autre, car elles ont plus de chance d'être identiques du point de vue de l'isomorphisme. En effet, le solveur CSP, dans le parcours de l'espace des solutions, va de proche en proche. C'est-à-dire que deux solutions générées l'une après l'autre auront plus de chance d'être proches d'un point de vue des connexions et donc

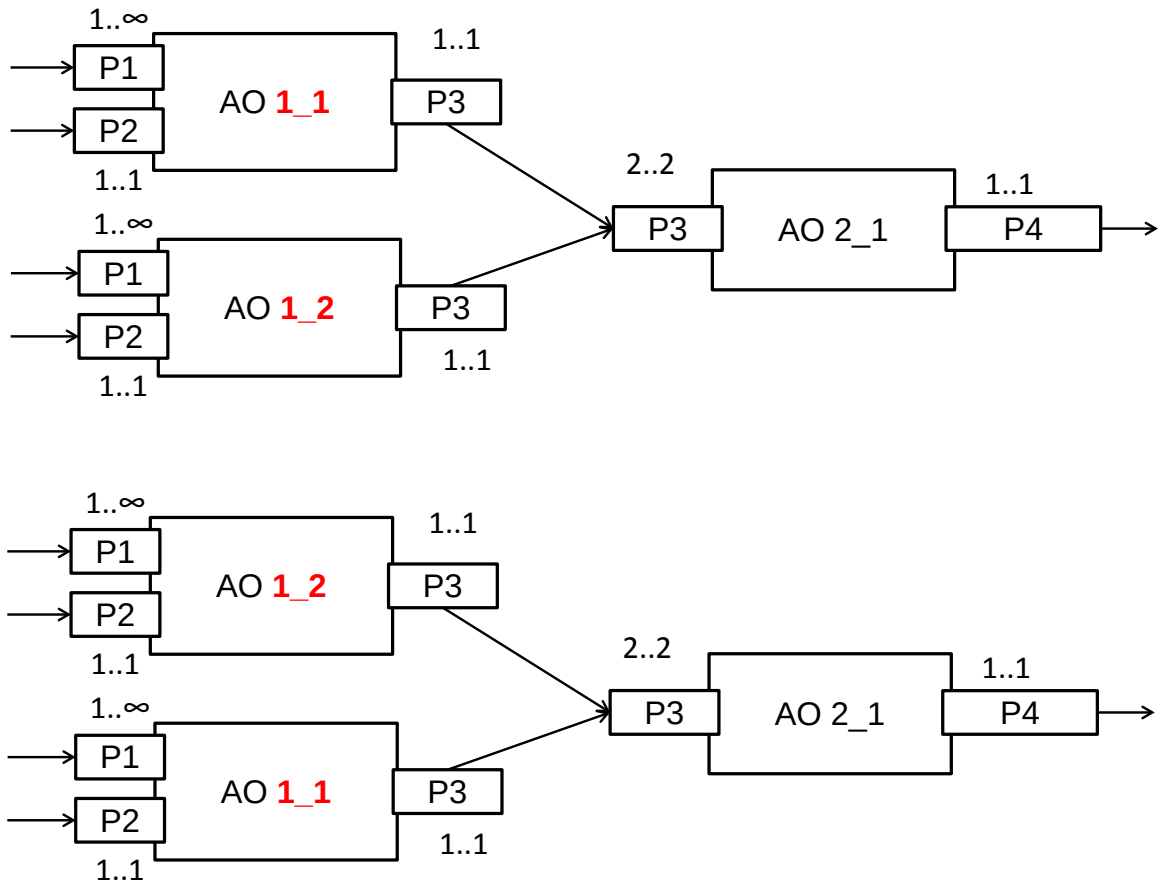


FIGURE 4.7 – Exemple d'un isomorphisme.

isomorphiques. Entre deux familles, à certaines exceptions près, deux solutions ne sont pas isomorphiques, car elles sont trop éloignées l'une de l'autre.
Si ces deux solutions, générées l'une après l'autre, ne sont pas isomorphiques, alors la comparaison continue avec toutes les autres solutions.
Premièrement, les nombres de connexions dans les deux solutions sont comparés. Si les nombres sont différents, alors les deux solutions sont différentes. Deuxièmement, le même processus est appliqué sur les connexions entre les AO et les ports d'entrée du système à concevoir. Si sur les deux architectures logiques ces nombres de connexions sont différents, alors les solutions sont différentes. Le même processus est appliqué pour les ports de sortie du système à concevoir. Pour rappel, les ports d'entrée et de sortie du système à concevoir, sont les interfaces avec l'environnement définies par l'utilisateur.
Après s'être attaché à comparer les nombres totaux de connexions entre deux solutions, l'algorithme vérifie que, pour les connexions entre le système à concevoir et les ports d'entrée et de sortie attendus, les AO connectés dans les deux architectures logiques sont bien du même type. Si un port d'entrée ou un port de sortie n'est pas connecté au même type d'AO, alors les deux solutions sont différentes.
Si malgré tous ces tests, les deux architectures logiques ne sont toujours pas démontrées comme identiques, une formule matricielle permet de savoir si deux graphes sont isomorphiques à l'aide de leur matrice d'adjacence et d'une matrice de permutation. Une matrice d'adjacence contient des uns et des zéros et représente les connexions entre les nœuds du graphe. Or, nos DSM sont représentées par des matrices et donc aussi des graphes. Il se trouve qu'une matrice DSM, même MIMO, est une matrice d'adjacence d'un graphe. Nous pouvons alors utiliser la formule (4.1), avec DSM_1 et DSM_2 les deux matrices DSM

des deux architectures logiques et P une matrice de permutation :

$$DSM_1 = P \times DSM_2 \times P^T \tag{4.1}$$

On calcule toutes les matrices de permutation possibles à l'intérieur de la base de données des AO étendue. Une matrice de permutation est une matrice carrée qui contient des zéros et des uns avec exactement un chiffre un par colonne et par ligne. Les matrices de permutation sont construites de la façon suivante : si deux AO sont de même type dans la base de données étendue, alors ils peuvent être intervertis. La matrice de permutation aura donc des uns entre tous les ports de ces deux AO, l'un étant représenté en ligne et l'autre en colonne dans la matrice. Notons que le nombre de matrices de permutation est défini par l'équation (4.2) avec o_i le nombre d'occurrences de l'AO numéro i.

$$\prod_{i=1}^{n}(o_i!) \tag{4.2}$$

L'algorithme parcourt toutes les permutations possibles entre les AO de la base de données étendue et, si une matrice P valide l'équation (4.1), alors les deux architectures logiques sont isomorphiques.

4.7.2 La prise en compte des situations de vie dans la détection des isomorphismes

Comme cela est présenté dans ce manuscrit, la génération des architectures prend en compte les différentes situations de vie que le système à concevoir va rencontrer et les différents modes de fonctionnement de chaque AO. Il est donc possible d'avoir deux solutions avec les mêmes connexions entre AO, mais avec des modes de chaque AO activés différemment suivant la situation de vie. L'algorithme précédemment décrit ci-dessus est donc modifié pour porter non plus sur la vue globale des architectures logiques, mais bien sur les différentes vues relatives aux situations de vie. C'est-à-dire qu'au lieu de tester les DSM globales, l'analyse va porter sur les DSM réduites associées chacune à une situation de vie donnée. Deux solutions sont maintenant dites isomorphiques uniquement si elles ont des graphes isomorphiques dans toutes leurs situations de vie. Cela veut dire que l'algorithme décrit précédemment est lancé tel quel sur les matrices DSM réduites.

4.8 La représentation des architectures logiques

Comme exposé par CAGAN et collab. [2005], la représentation des solutions est une étape clé dans les processus de synthèse. En effet, cela permet aux architectes de vérifier les solutions proposées automatiquement et de pouvoir modifier leurs données d'entrée pour obtenir des sorties leur convenant mieux. Dans le cas présent, on utilise Graphviz, présenté par GANSNER et NORTH [2000], qui est un logiciel de représentation de graphes. Graphviz prend, en entrée, un code dédié dans le langage DOT au format texte et fournit, en sortie, au choix, plusieurs formats tels que PNG, PDF, SVG, JSON ... Ce logiciel a été sélectionné car il permet de tracer des graphes avec une grande liberté. De plus, l'interfaçage avec un logiciel en développement est très simple car il suffit de créer, à la volée, plusieurs fichiers DOT qui seront ensuite exécutés par Graphviz.
La représentation choisie est proche de celle des diagrammes de blocs internes, du langage SysML. Il a néanmoins été décidé de ne pas directement utiliser du SysML pour garder de la liberté dans la représentation et s'affranchir de certains codes graphiques pouvant

être déroutants pour des mécaniciens. Cet avis est partagé avec HAMPSON [2015]. De plus, Graphviz dispose de son propre algorithme de placement des nœuds, en minimisant les intersections entre les liens liant les différents nœuds. Un AO est représenté comme un tableau avec la première colonne représentant ses ports d'entrée, chaque ligne de cette colonne étant un port. La deuxième colonne porte le nom de l'AO en question sur une seule ligne et la troisième colonne est similaire à la première, sauf qu'elle porte sur les ports de sortie. Ensuite, des connexions sont faites entre les cases représentatives des ports.
Une solution est représentée par un graphe global permettant de visualiser toutes les interfaces, ainsi que par plusieurs graphes représentant chacun une situation de vie considérée pour le système à concevoir. Dans ce cas, les ports qui ne sont pas activés sont grisés et aucun flux n'en sort.

4.9 Exemple fil rouge

4.9.1 Construction de la base de données des AO

Pour la construction de la base de données qui nous importe, un travail préliminaire de synthèse manuelle a été conduit. Il s'agit d'une analyse systématique des architectures aéronefs d'après la fonction principale de sustentation. La synthèse s'est néanmoins focalisée sur les voilures tournantes et non pas sur les voilures fixes telles que les avions. Ce travail est résumé sur la Figure A.1 en Annexe, page VIII. Cette analyse permet de détecter les différentes technologies utilisées dans le champ de conception qui intéresse l'architecte. Cela permet aussi de valider les résultats obtenus en vérifiant que toutes les architectures trouvées à la main ont bien émergé de la synthèse automatique. Si elles n'ont pas toutes émergé, il peut être nécessaire soit de modifier la définition des exigences qui contraignait trop le champ des solutions, soit de rajouter des AO qui peuvent être manquants. Il n'est pas rare de lancer une synthèse avec le logiciel présenté dans ce manuscrit et de n'obtenir aucune solution car un AO simple mais obligatoire n'a pas été défini.

4.9.2 Définition des ports de la base de données

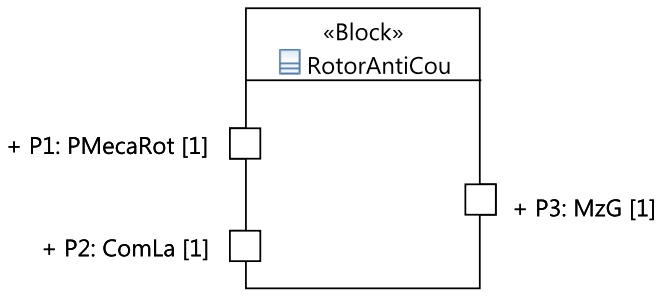
Dans l'exemple proposé ici, les ports définis dans la base de données des AO sont les mêmes que ceux qui ont été introduits précédemment pour la définition des interfaces du système à concevoir avec, en plus, certains ports qui seront internes au système. Il est néanmoins possible de créer une base plus étendue que celle utile pour la résolution, les AO inutiles ne seront simplement pas utilisés dans les solutions. Le problème d'utiliser une base plus étendue est de pénaliser le temps de résolution du problème. Cela sera étudié plus en détail dans la suite du manuscrit. La liste exhaustive des ports utilisés pour l'exemple est présentée sur les Annexes A.4 page XII , A.3 page XI et A.2 page X. Seules les feuilles de ces graphes sont à considérer, les autres nœuds sont utilisés pour regrouper les ports par type et pour améliorer la lecture.

4.9.3 Vue logique de l'AO

Pour l'exemple du drone, dix-huit AO ont été sélectionnés pour constituer la base de données. Ce chiffre est volontairement faible pour limiter les temps de calcul des solutions qui peuvent être longs sans l'utilisation de supercalculateurs. Les AO définis permettent de générer de la variété dans les solutions, tout en restant dans des temps raisonnables. Pour les mêmes raisons, la multiplicité infinie n'est pas transformée en une multiplicité égale à

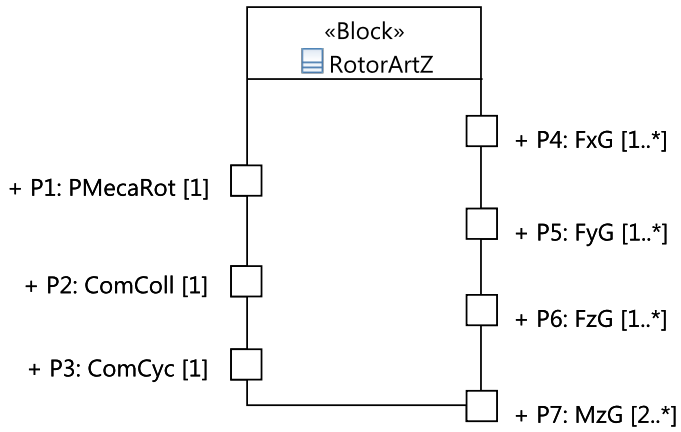
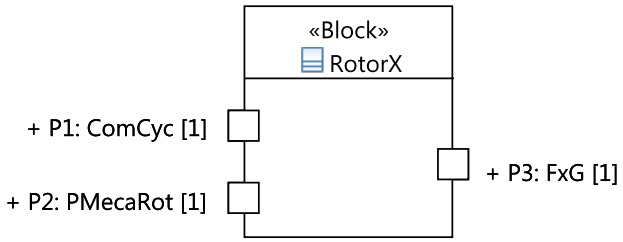
cent, mais à dix. Cela veut dire que si un port peut être connecté une infinité de fois, alors, lors de la synthèse, il ne sera considéré que comme pouvant être connecté dix fois. Cela est suffisant vu le nombre d'AO de la base de données. Une valeur de la multiplicité infinie trop élevée causerait des temps de calcul très longs si deux ports ayant cette multiplicité étaient connectés. Ce point sera creusé plus en détail dans le chapitre 4.6.2, page 89.
Les dix-huits AO sont présentés en Annexe A.1, page XIII. Les sous-ensembles conservés sont les plus importants pour répondre aux besoins des parties prenantes. Une caméra, un altimètre et un GPS sont utilisés dans le but de répondre à des besoins du pilote, mais la vraie valeur ajoutée de la synthèse automatique réside plutôt dans la variété des systèmes de production d'énergie mécanique de rotation et dans les types systèmes propulsifs utilisés, qui sont présents dans la base de données. Un extrait de la base de données est visible sur le Tableau 4.1.
Dans la base de données des AO utilisée, les couples et efforts ont une place prépondérante. En effet, pour un aéronef à voilures tournantes, l'équilibrage entre les différents ports, voulus ou non, concernant les couples est du premier ordre. Rappelons qu'un rotor principal d'hélicoptère reçoit un couple de la Boîte de Transmission Principale (BTP), elle-même fixée sur la structure. Or, d'après le principe d'action/réaction, le rotor fournit un couple repris par le carter de la BTP et transmis à la structure. C'est pour cette raison qu'un rotor anticouple est placé sur la queue de l'appareil. L'architecture que nous venons de prendre en exemple est celle qui s'est imposée à travers le siècle dernier du fait de sa simplicité et de son efficacité. Néanmoins, toutes les nouvelles architectures d'aéronefs à voilures tournantes sont soumises à la même contrainte.
L'important est donc de connaître la magnitude, la direction et le point d'application des différents efforts mis en jeu. Pour ce faire, nous reprenons la dénomination des axes principaux utilisée dans l'aéronautique, rappelée sur la Figure 3.9.

TABLEAU 4.1 – Extrait de la base de données des AO

Description	AO
Le rotor anticouple est un rotor placé sur la queue de la machine, suivant l'axe de roulis de l'aéronef. Il reçoit une énergie mécanique de rotation et son rôle est de fournir un anticouple. Bien entendu, il fournit aussi une force suivant l'axe de tangage, mais pour une modélisation à notre niveau, où les angles d'orientation exacts des rotors ne sont pas étudiés, cette information est superflue.	

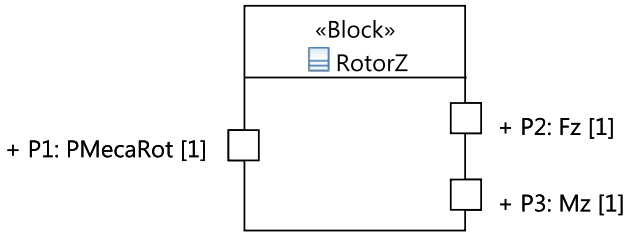
continue sur la page suivante

continue depuis la page précédente

Description	**AO**
Le rotor articulé reçoit une commande mécanique cyclique et une commande mécanique collective provenant du système de commande, ainsi qu'une puissance mécanique de rotation, et donne à la structure les efforts nécessaires pour se déplacer dans les airs. Ce rotor est orienté suivant l'axe de lacet.	
Le rotor de poussée est une hélice orientée suivant l'axe de roulis. Il permet la propulsion de l'aéronef. Il reçoit une commande cyclique et pas de commande collective. C'est-à-dire que l'hélice ne produira pas de portance. Il reçoit bien entendu une puissance mécanique de rotation.	

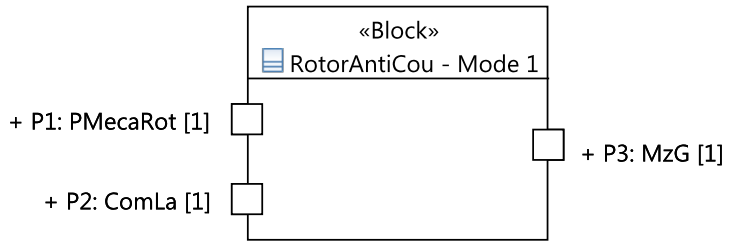
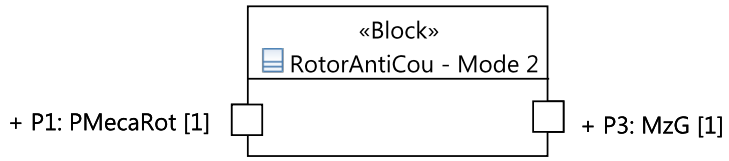
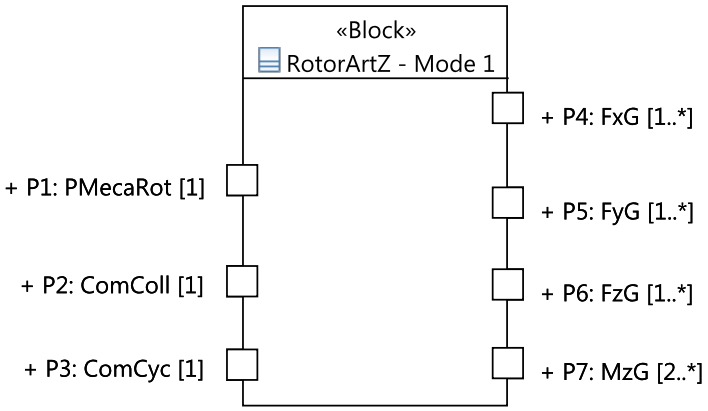
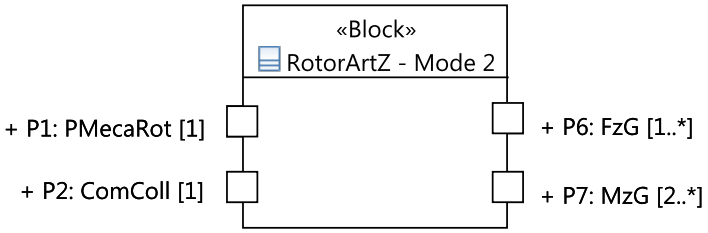
continue sur la page suivante

continue depuis la page précédente

Description	**AO**
Ce rotor est orienté suivant l'axe de lacet et n'est pas articulé. Cela veut dire que seul, il ne peut pas produire autre chose que de la portance. L'AO calculateur est important justement pour modérer les portances des différents rotors non articulés permettant la création des efforts de propulsion. La vitesse de rotation constitutive du flux est modulée pour générer plus ou moins de portance et forcément de couple.	

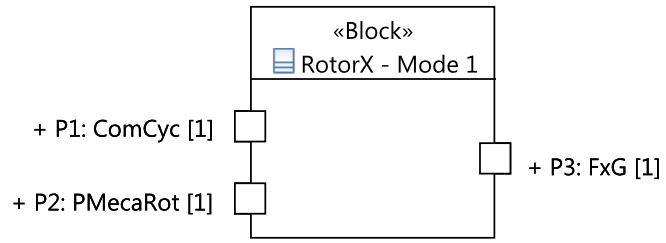
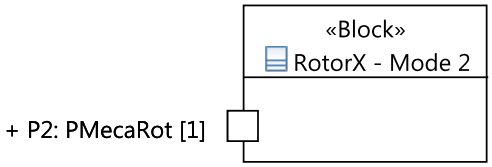
Maintenant que tous les AO ont été définis vis-à-vis de tous les ports qui les composent, il est nécessaire de définir leurs différents modes de fonctionnement. En effet, les interfaces du système à concevoir ont été définies pour les différentes situations de vie sur les Figures 3.13, 3.14 et 3.15. Elles seront instanciées par des AO dans différents modes de fonctionnement. Notons que tous les AO ont un mode à l'arrêt, c'est-à-dire qu'aucun port ne sera activé et donc qu'aucun flux ne sera échangé. Cela permet donc d'avoir un AO utilisé dans certaines situations de vie du système à concevoir et en même temps à l'arrêt dans d'autres cas. Les AO qui n'ont que deux modes, avec, dans le premier, tous les ports actifs et, dans le second, tous les ports inactifs, ne sont pas illustrés dans le Tableau A.2 en Annexe, page XX. Le Tableau 4.2 présente un extrait du tableau donné en Annexe, il concerne les mêmes AO que ceux du Tableau 4.1.

TABLEAU 4.2 – Extrait des modes des AO

Description	AO
Le premier mode du rotor anticouple correspond à une utilisation où du pas est appliqué aux pales de ce rotor. Du pas est appliqué lorsque le couple transmis du rotor principal à la structure augmente ou lorsque l'aéronef doit être commandé en lacet.	
Le second mode du rotor anticouple correspond à la production d'un anticouple suffisant pour équilibrer l'aéronef, mais pas pour de la commande en lacet.	
Le premier mode du rotor articulé correspond à un vol d'avancement.	
Le premier mode du rotor articulé correspond à un vol stationnaire.	

continue sur la page suivante

continue depuis la page précédente

Description	**AO**
Le premier mode du rotor orienté dans l'axe de roulis correspond à un vol d'avancement.	
Le premier mode du rotor orienté dans l'axe de roulis correspond à un vol stationnaire.	

Dans le but de réduire les temps de calcul et de produire des architectures plus faciles à étudier sous la forme d'un graphe, il a été décidé de réduire encore la base de données des AO présentée au dessus. Tous les AO ne mettant pas en jeu des transformations de puissance sont laissés de côté. Sont donc retirés le GPS, la caméra, l'altimètre et la cellule. Tous ces sous-systèmes peuvent être regroupés en un sous-système commun à chaque architecture qui serait un système d'aide au pilotage modélisé sous une boîte noire fictive. Par application récursive de la méthodologie de synthèse, cette boîte noire pourrait ensuite être étudiée pour trouver son instanciation optimale lors d'une synthèse architecturale.
Nous verrons plus tard que, pour le vol d'avancement qui est une situation de vie du système, les coefficients de traînée doivent être pris en compte et la cellule est un gros contributeur au coefficient global. Comme la cellule est toujours présente, il a été décidé d'ajouter à la valeur calculée lors de l'analyse physique une valeur fixe pour la cellule.
Ces suppressions d'AO permettent aussi et surtout d'alléger le nombre de ports présents dans la base de données. Ainsi, les ports «Position», «EnvVisu», «Altitude», «Intempéries», «SignalPosition», «SignalVideo» et «SignalAltitude» sont supprimés.
Pour les mêmes raisons, la multiplicité infinie n'est pas transformée en une multiplicité égale à cent, mais à dix. En effet, cela implique que lors de la génération des architectures logiques, la cardinalité réelle d'une connexion entre deux ports ayant une multiplicité maximale fixée à l'infini sera uniquement testée entre 1 et 10, et non pas entre 1 et 100. Cela réduit très fortement le nombre de solutions non affichées, mais néanmoins calculées par le code CSP. Pour illustrer cet exemple, prenons le cas d'un moteur thermique connecté à un réservoir à carburant. Comme cela est visible sur la Figure 4.8, les deux ports de type «Essence» peuvent être connectés une infinité de fois. Mais étant donné que ces deux ports ne sont connectés qu'entre eux, cela n'a que peu d'intérêt de savoir quelle sera la cardinalité réelle de cette connexion. Cette dernière néanmoins calculée par le CSP faisant la synthèse des architectures logiques. Nous savons, de part la définition de nos AO, qu'il ne devrait pas à avoir de cardinalité pour un port supérieure à dix. C'est pour cette raison que nous prenons la liberté de modifier cette borne infinie en une borne à dix.

A la fin du sous-processus présenté jusqu'à présent, le problème de synthèse est totalement défini. Il ne reste plus qu'à le résoudre à l'aide du logiciel créé au cours des travaux de recherche conduisant à ce manuscrit. En effet, le système à concevoir est, à ce stade,

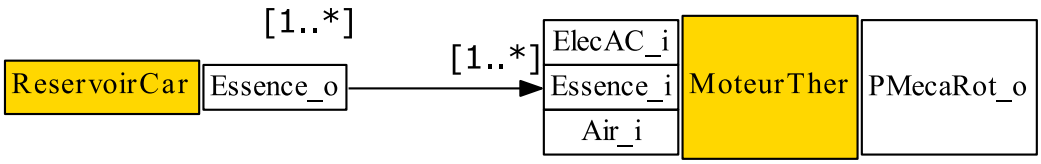


FIGURE 4.8 – Exemple d'une multiplicité infinie.

totalement spécifié et la base de données des technologies utilisables vient d'être définie. Notons, de plus, que la base de données des AO n'a pas besoin d'être redéfinie pour chaque problème de synthèse. En effet, un AO est défini une fois pour toute, il représente une technologie particulière qui peut être utilisée ou non dans un système complexe. Néanmoins, pour des problèmes de synthèse très différents, une nouvelle base de données doit, dans certains cas, être créée.

4.9.4 Les méta-architectures

Pour l'exemple fil rouge, après réduction de la base de données des AO au strict minimum, ce qui équivaut à une base de quatorze AO, le nombre de ports est de 47. Directement, cela donne la taille des matrices carrées d'autorisation et d'obligation. Il s'agit donc de matrices 47 x 47. Elles sont construites de manière automatique, en traduisant la base de données des AO. Il n'est pas intéressant de les présenter en détail. Pour rappel, ces matrices, composées majoritairement de zéros, permettent de fixer les variables du CSP directement à zéro sans avoir à déduire ces valeurs des contraintes mathématiques exprimées dans le fichier modèle Minizinc. Si la base de données des AO n'avait pas été réduite au minimum, et qu'un AO avait été modélisé, mais pas utilisé dans les architectures, alors ce dernier aurait reçu une occurrence égale à «zéro». Il aurait donc été retiré de la base de données étendue.

Les interfaces définies dans la section 3.5.7, page 67 et 3.12, page 68, ont été simplifiées pour se concentrer uniquement sur les flux d'échange de puissance, qui sont les plus importants dans les architectures que nous souhaitons synthétiser. Néanmoins, les interfaces du système à concevoir, visibles sur la Figure 4.9, font bien partie de ces interfaces définies lors de la capture du besoin et de la modélisation des connaissances, qui correspondent à l'étape précédente de la méthodologie présentée.

Le nombre d'occurrences de l'AO «MoteurTher» est forcé entre zéro et un. Cela veut dire que nous obtiendrons l'ensemble des solutions minimalistes pour le problème de synthèse donné, mais avec au plus un moteur thermique. D'un autre côté, les moteurs électriques ne sont pas limités en nombre, cela permettra de générer des solutions avec un moteur électrique par rotor. Au premier abord, il est déjà clair que des solutions hybrides seront proposées, car la base de données dispose d'un moteur électrique ainsi que d'un moteur thermique.

Sans prendre en compte les multiplicités des ports à ce stade, 82556 méta-architectures sont générées. Cela équivaut au résultat d'une synthèse sur un système à concevoir possédant trois situations de vie distinctes et avec une base de données de treize AO, ayant eux-mêmes plusieurs modes de fonctionnement. Pour être précis, chaque AO a entre deux et quatre modes de fonctionnement. Aucun AO ne peut avoir moins de deux modes de fonctionnement car il y a toujours le mode nominal et le mode à l'arrêt. Les modes de fonctionnement de tous les AO sont visibles sur l'Annexe A.2.2, page XIX. Les occurrences du moteur thermique et du rotor articulé orienté en «Z» ont été forcées de «zéro» à «un». A ce stade de la synthèse, toutes les solutions répondant aux contraintes ont été trouvées.

La Figure 4.9 est une des méta-architectures mettant en œuvre tous les AO de la base de données. Plusieurs d'entre elles ne seront pas viables une fois que les multiplicités seront prises en compte, comme par exemple la solution représentée sur la Figure 4.9. Néanmoins, il était obligatoire de les générer pour pouvoir ensuite calculer les occurrences de chaque AO. Une autre façon de procéder aurait été de préfixer les occurrences de tous les composants à une valeur de cinq par exemple, mais le temps de calcul au global aurait été très largement augmenté car la base de données des AO aurait été cinq fois plus grande. De plus, cette base aurait été fortement augmentée par des AO identiques, fournissant ensuite un grand nombre d'isomorphismes très longs à détecter.

4.9.5 Les occurrences des objets architecturaux

Autant de fichiers de données Minizinc que de solutions au problème de synthèse des méta-architectures sont créés de manière automatique. Il y en a donc 82556 pour quatre fichiers modèles Minizinc car deux AO ont eu leurs occurrences forcées. En effet, une combinatoire est faite entre les valeurs des intervalles d'occurrences imposées par l'utilisateur. Ici, l'intervalle [0..1] est rentré pour le nombre d'occurrences du rotor articulé et l'intervalle [0..1] est rentré pour le nombre d'occurrences du moteur thermique. Nous obtiendrons quatre fichiers modèles avec des contraintes sur les nombres d'occurrences différentes. Chaque fichier de données permet de déterminer le nombre d'occurrences nécessaire pour chaque AO et pour chaque méta-architecture, c'est-à-dire pour chaque famille d'architectures.

Une fois que tous les fichiers CSP ont été traités par Minizinc, un code vient récupérer tous les résultats pour en extraire uniquement l'occurrence maximale nécessaire pour chaque AO. Dans notre exemple voici les occurrences calculées pour la nouvelle base de données des AO visibles sur le Tableau 4.3.

TABLEAU 4.3 – Les occurrences de la base de données étendue.

AO	Occurrence
ControlVol	1
RotorArtZ	1
RotorAntiCou	1
Calculateur	1
RotorZ	4
RotorX	1
MoteurTher	1
ReservoirCar	1
MoteurElec	6
Batterie	1
ConvertDCAC	1
ConvertACDC	1
BTP	1
Trains	1

Certaines méta-architectures ne sont pas du tout en accord avec les multiplicités des AO, même en augmentant le nombre d'occurrences de ses AO. Ces architectures ne sont pas

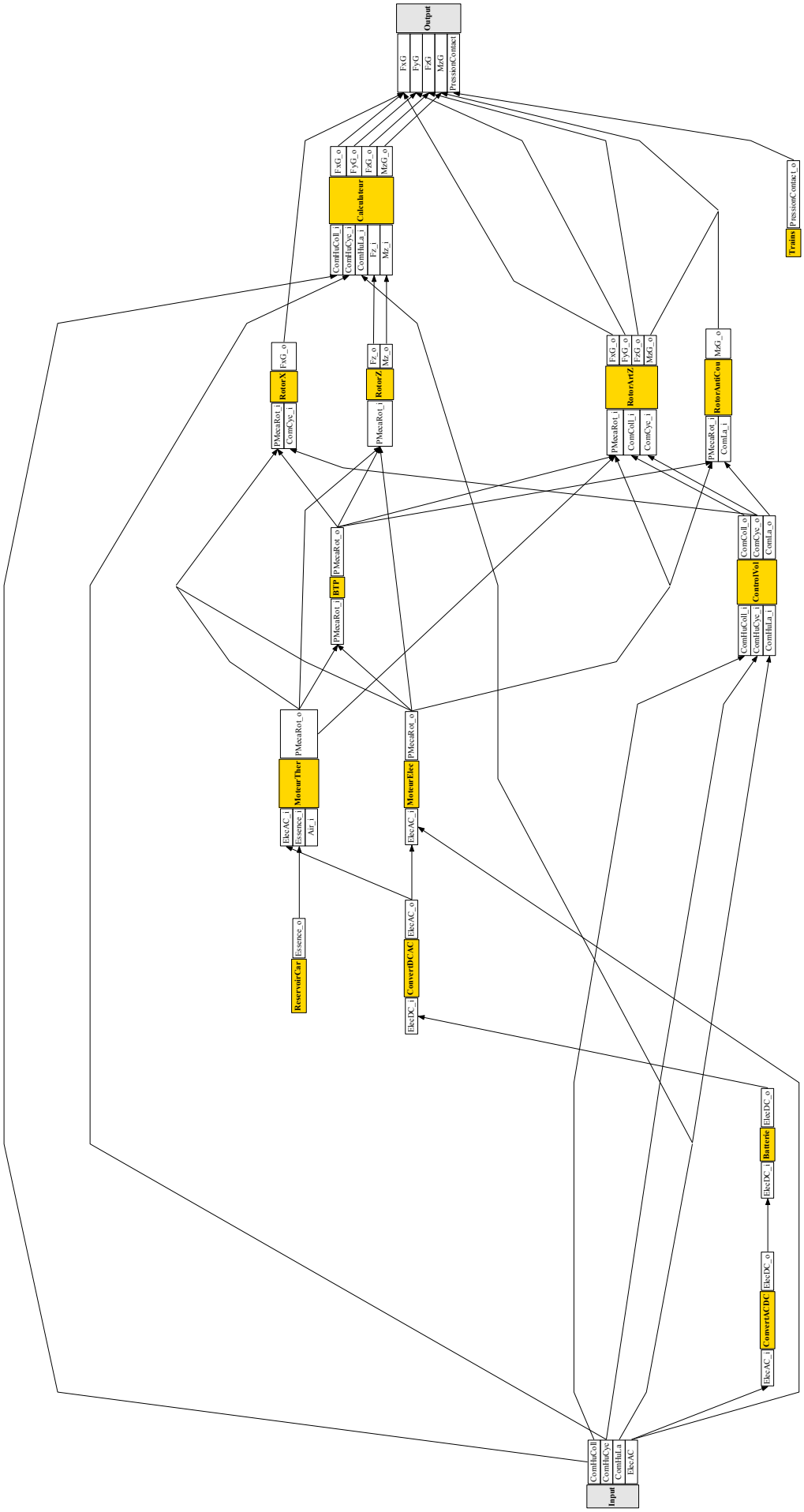


FIGURE 4.9 – Une méta-architecture avec tous les AO de la base.

gardées et ne formeront donc pas une famille d'architectures logiques. Pour illustrer le propos, la Figure 4.10 est une méta-architecture qui ne respecte pas les multiplicités des ports de ses AO. En effet, rappelons-nous des trois situations de vie du système à étudier :

— Vol stationnaire

— Vol d'avancement

— Rechargement

Pour les deux premières situations de vie, aucun problème n'a été rencontré, mais pour le dernier, nous avions spécifié que l'environnement ne possédait qu'une seule prise de parc, en mettant la multiplicité du port d'entrée «ElecAC» du système à concevoir à [1..1]. En analysant cette méta-architecture sur la Figure 4.10, il est clair que quatre moteurs thermiques seront nécessaires, car il n'y a pas de boîte de transmission permettant de distribuer la puissance d'un moteur vers tous les rotors non articulés. Or, cette architecture ne possédant pas de batterie, le démarrage des moteurs thermiques ne peut se faire que sur prise de parc et il en faudrait quatre, ce qui n'est pas le cas. Cette famille d'architecture est alors laissée de côté.
Sur les 82556 méta-architectures, seules 208 sont viables et peuvent permettre la génération d'architectures logiques.

La synthèse des méta-architectures a permis de savoir combien d'AO de chaque type étaient nécessaires pour que toutes les solutions soient fonctionnelles. Par exemple, quatre rotors fixes orientés suivant l'axe de lacet sont nécessaires pour faire fonctionner une solution avec un calculateur. Pour rappel, ici un calculateur est un composant permettant de commander de quatre à huit rotors à pales fixes pour obtenir tous les efforts nécessaires ramenés sur le drone lui permettant de se déplacer librement.

4.9.6 Les architectures logiques

Les matrices d'autorisation et d'obligation sont maintenant plus imposantes depuis que la base a été étendue. Ces dernières sont maintenant de taille 63 x 63 pour 21 AO. Avec la base de données étendue et en respectant les familles de solutions obtenues grâce à la synthèse des méta-architectures, le logiciel trouve plus de 70000 solutions toutes différentes mathématiquement. Ce chiffre est approximatif, car, pour que l'exemple puisse être incorporé dans ce manuscrit à temps, le temps de calcul a dû être réduit et donc la génération n'est pas tout à fait exhaustive. La synthèse des architectures logiques a été réduite à sept minutes par méta-architecture ou famille d'architectures. Néanmoins, ce grand nombre de solutions énoncés n'est pas le nombre traité au final pour l'analyse physique. En effet, étant donné la combinatoire entre plusieurs éléments, comme les rotors non articulés et les moteurs électriques, qui peuvent être utilisés plusieurs fois, un grand nombre de solutions sont en réalité identiques, elles sont dites isomorphiques. Une détection des isomorphismes est nécessaire. 17280 matrices de permutations sont donc calculées. Ce nombre est retrouvé à l'aide de l'équation 4.1, page 94, tel que $4! \times 6! = 17280$. Puis, suite à l'application de l'algorithme de détection des isomorphismes durant une période de 30 h, seulement 47 architectures logiques demeurent. Notons que la recherche, pour cet exemple, n'a pas été exhaustive à cause du temps de calcul qui a été limité. L'ordinateur utilisé est un ordinateur portable de bureau avec 16Gb de ram et un processeur Intel(R) Core(TM) i7-4910MQ cadencé à 2.90GHz et tournant sous Windows 7 Professionnel 64-bit. Il serait très intéressant de traiter, dans le futur, des problèmes plus gros avec l'aide de supercalculateurs.
A l'étude de ces 47 solutions, on se rend compte que seules des solutions quadrotor avec

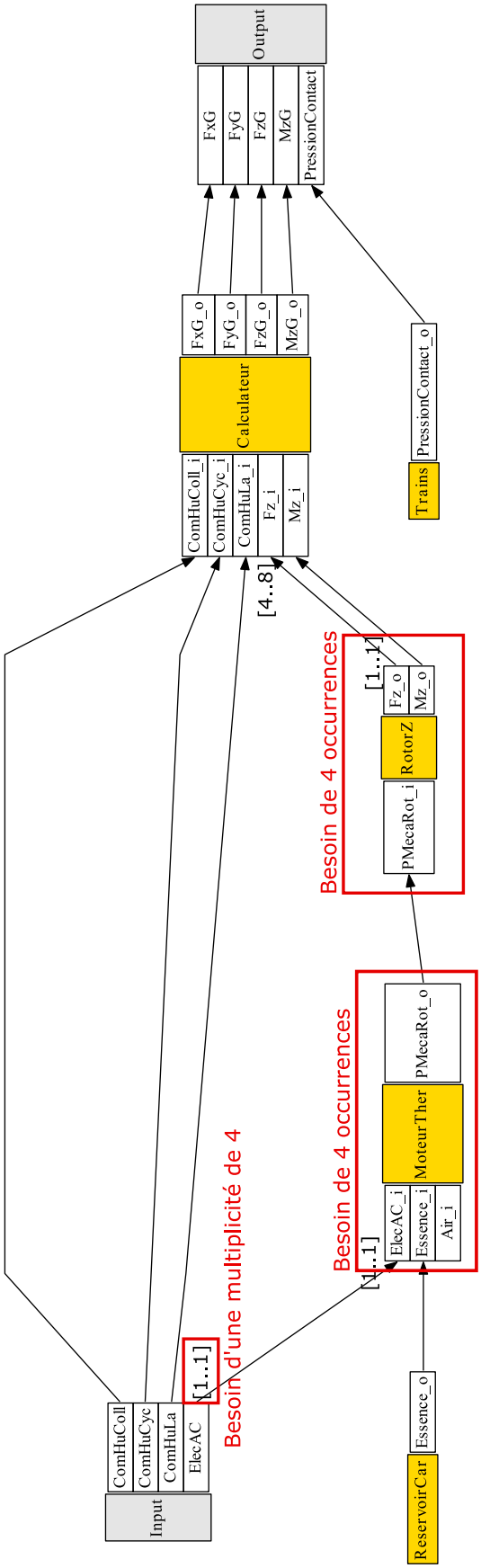


FIGURE 4.10 – Une méta-architecture qui ne respecte pas les multiplicités des ports.

BTP ont émergé. Pour obtenir plus de variété, il aurait fallu laisser l'algorithme de détection des isomorphismes tourner plus longtemps. En effet c'est cet algorithme qui parcourt une à une toutes les solutions synthétisées par le processus de synthèse logique automatique. Si la solution courante est nouvelle, c'est-à-dire si elle n'a pas un isomorphisme dans les solutions déjà trouvées, alors elle est rajoutée aux solutions architecturales. Si l'algorithme de détection des solutions non-isomorphiques est arrêté avant d'avoir parcouru l'ensemble des solutions, alors le résultat n'est pas exhaustif, mais aucune solution n'est redondante.

Deux autres synthèses des architectures logiques avaient été menées à leur terme précédemment. La définition du problème de conception et de la base de données des AO était légèrement différente.
Dans le premier cas, la définition du problème de conception ne s'est focalisée que sur les flux de commande et de puissance. Il n'y avait pas de rotor de poussée dans la base, ni de moteur thermique et aucun AO annexe. Après 5 jours de calcul, 99 architectures logiques ont été générées.
Dans le second cas, la définition du problème de conception et la base de données des AO étaient complètes. Néanmoins, seules les architectures conventionnelles ont été étudiées, c'est-à-dire avec un rotor principal et un rotor anticouple à la queue de la machine. Pour ce faire, les occurrences des rotors fixes et des rotors de poussée ont été mises à «0». Après une semaine de calcul, 3225 architectures logiques ont été générées. Ce nombre est important car deux moteurs thermiques et deux moteurs électriques sont utilisés dans la base de données étendue. Il y a donc une combinatoire entre ces moteurs et leur placement : soit avant une BTP, soit avant le rotor principal, ou avant le rotor anticouple et aussi entre les différentes situations de vie. En effet, suivant la situation de vie, tous les moteurs présents dans l'architecture ne sont pas forcément activés. Un des cas observés est l'utilisation de tous les moteurs en vol stationnaire et d'uniquement les moteurs thermiques en vol d'avancement.
Trois exemples provenant des trois synthèses présentées précédemment sont donnés sur les Figures 4.11, 4.12 et 4.13.
La Figure 4.11, représente un quadrotor avec un seul moteur thermique et donc une boîte de transmission pour fournir de la puissance mécanique aux quatre rotors.
La Figure 4.12, représente un quadrotor avec quatre moteurs électriques. Il s'agit de l'architecture conventionnelle pour de petits drones commerciaux.
La Figure 4.13, représente un hélicoptère conventionnel monomoteur.

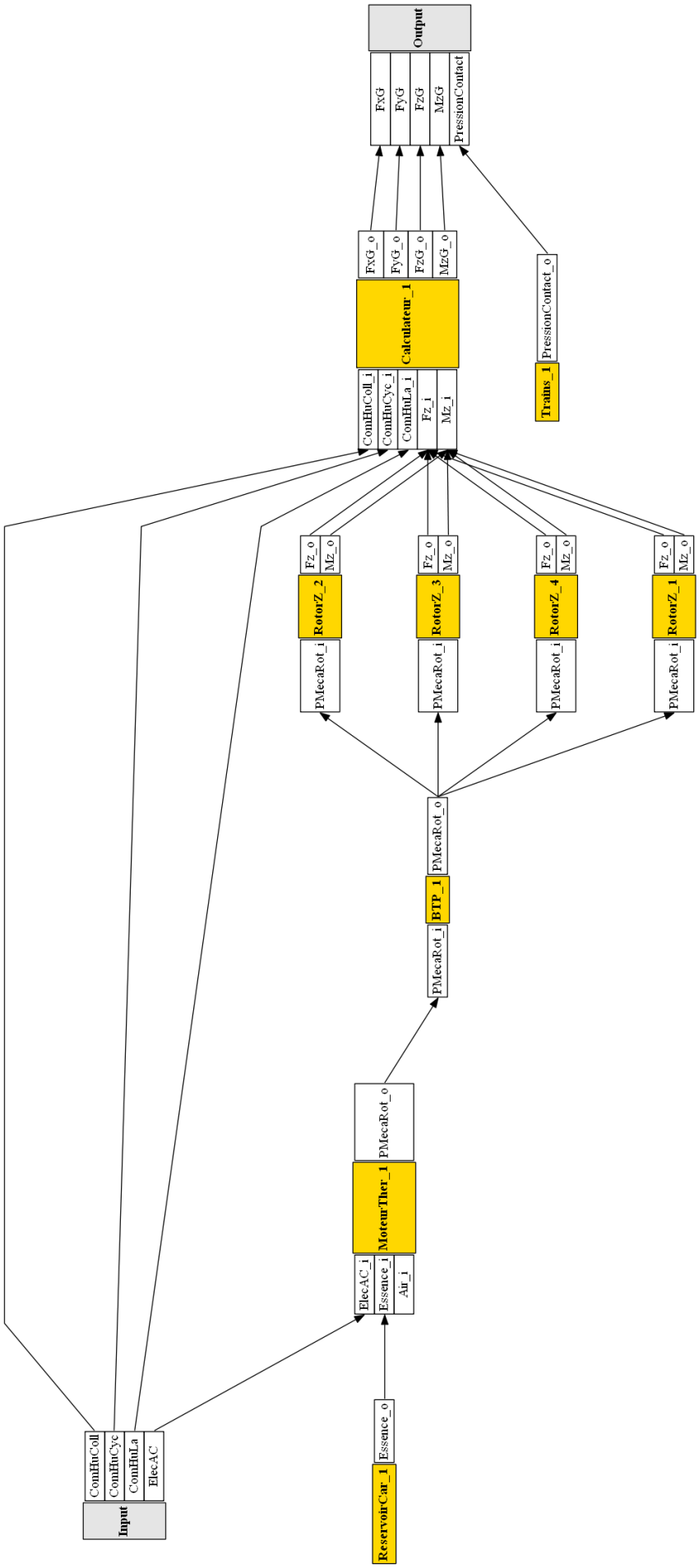


FIGURE 4.11 – Une architecture logique de la première synthèse. 107

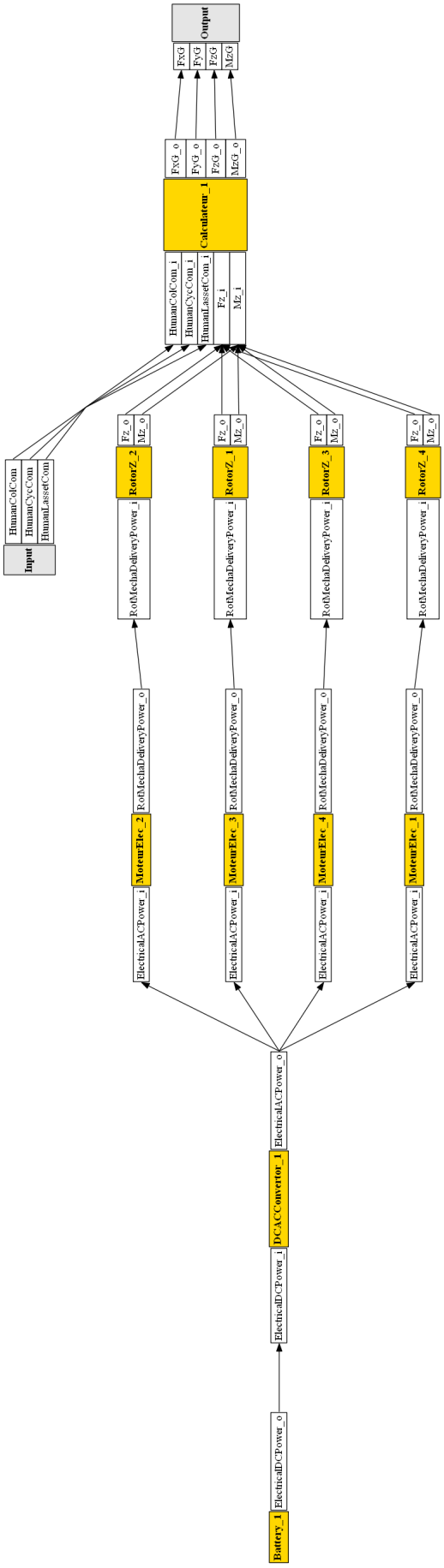


FIGURE 4.12 – Une architecture logique de la seconde synthèse.

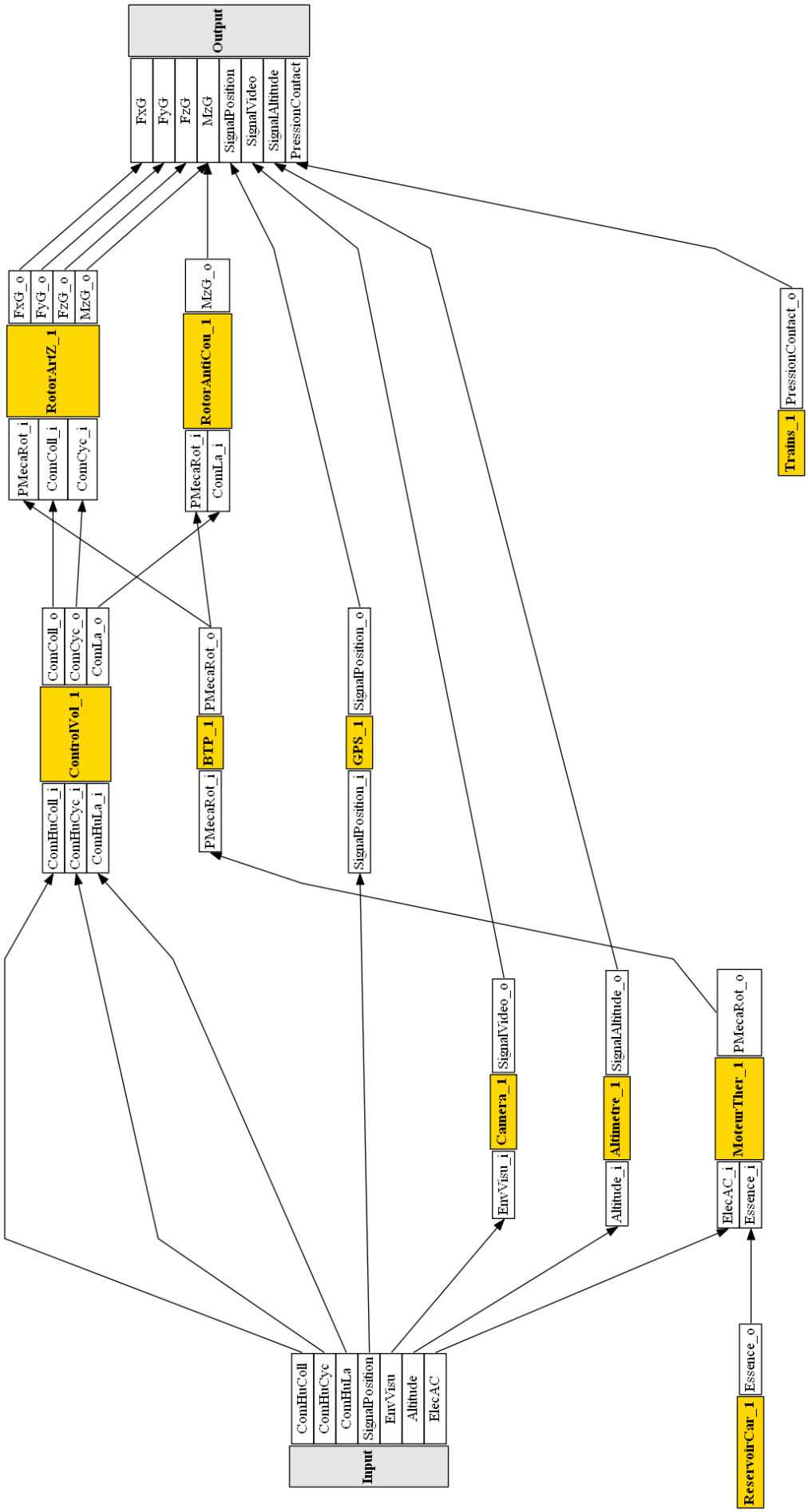


FIGURE 4.13 – Une architecture logique de la troisième synthèse.

4.10 Conclusion du chapitre

Ce chapitre représente le cœur de la synthèse architecturale assistée par ordinateur. C'est, en effet, ce sous-processus qui permet de générer la variété des solutions architecturales car il s'agit de la phase d'expansion du processus de synthèse. A l'aide du démonstrateur créer pour supporter la méthode, les entrées, données par l'architecte, sont traitées et mises en forme mathématiquement, pour ensuite être résolues par un solveur CSP. L'utilisation de ce type de solveur est un outil adéquat à l'interconnexion d'AO. Ce chapitre apporte donc une réponse à la question de recherche **Question 3**.
Un des points importants de la synthèse est la présentation des résultats à l'architecte qui exprime le problème de conception. Pour présenter une architecture logique, la forme du graphe MIMO est tout à fait adaptée. Un système peut donc être représenté par un ensemble de graphes qui correspondent à chaque situation de vie. Ce chapitre, avec cette proposition de représentation, répond à la question de recherche **Question 6**.

La synthèse n'est néanmoins pas terminée, car plusieurs centaines, voire plusieurs milliers d'alternatives, peuvent exister. Il n'est pas possible de toutes les traiter à la main. Une dernière phase d'analyse est nécessaire pour rejeter toutes les architectures n'étant pas viables physiquement. Plusieurs solutions sont maintenant possibles, allant du prédimensionnement sans équations différentielles, à la simulation des modèles plus complexes.

Chapitre 5

Analyse des architectures physiques

Sommaire

5.1 Introduction du chapitre

La troisième et dernière étape traitée dans le cadre de ces travaux de recherche sur la méthodologie de synthèse architecturale porte sur l'analyse physique des architectures logiques. Lors de cette phase, chaque solution est étudiée pour générer de manière automatique un système d'équations non linéaires et non différentielles, appelé architecture physique. Le but est de prédimensionner chaque sous-système pour faire des analyses comparatives sur les solutions générées, avec la même ambition que les travaux présentés par CHENOUARD [2007]. Ces analyses porteront sur les variables d'état globales du système à concevoir. D'autres travaux, comme celui de RENIER et CHENOUARD [2011], s'efforcent de lier des architectures logiques à des architectures physiques. De même, CHAPON et BOUCHEZ [2009] présentent un profil UML utilisant le langage OCL, pour décrire les équations comportementales des systèmes spécialement conçus pour créer des modèles Modelica pour l'aéronautique. L'utilisation de Modelica dans le cas des travaux présentés dans ce manuscrit n'est pas directement adaptée. A priori, les valeurs des variables d'état de nos sous-systèmes, les AO, ne sont pas connues avant une première résolution. Or, Modelica ne gère que des systèmes carrés, c'est-à-dire avec autant d'inconnues que d'équations. KOMOTO et collab. [2014] présentent un outil permettant la visualisation des paramètres d'une architecture physique sous la forme d'un graphe. Les différentes variables sont reliées par des liens représentant les équations d'état et les équations comportementales des sous-systèmes. Les variables d'état sont obtenues depuis des tables fixes, mais la sélection automatique des valeurs de ces variables d'état n'est pas décrite dans le papier. Dans ce manuscrit, les variables d'état ne sont pas fixées à l'avance, leur valeur est le fruit d'une optimisation sur un unique critère matérialisé par une variable d'état globale à tout le système, telle que la masse par exemple.
Tout d'abord, chaque constituant du modèle physique est défini, puis la méthode de création du modèle est expliquée. Ensuite le modèle, écrit sous la forme d'un CSP, est résolu à l'aide d'un solveur continu et finalement, les architectures physiques sont triées. Le tri proposé dans ce manuscrit est un tri simple sur la seule variable d'état du système à optimiser. En effet, les travaux ne portaient pas uniquement sur cette problématique complexe, traitée par LO [2013], mais sur l'ensemble de la chaîne de la synthèse. Le processus d'analyse est illustré sur la Figure 5.1.

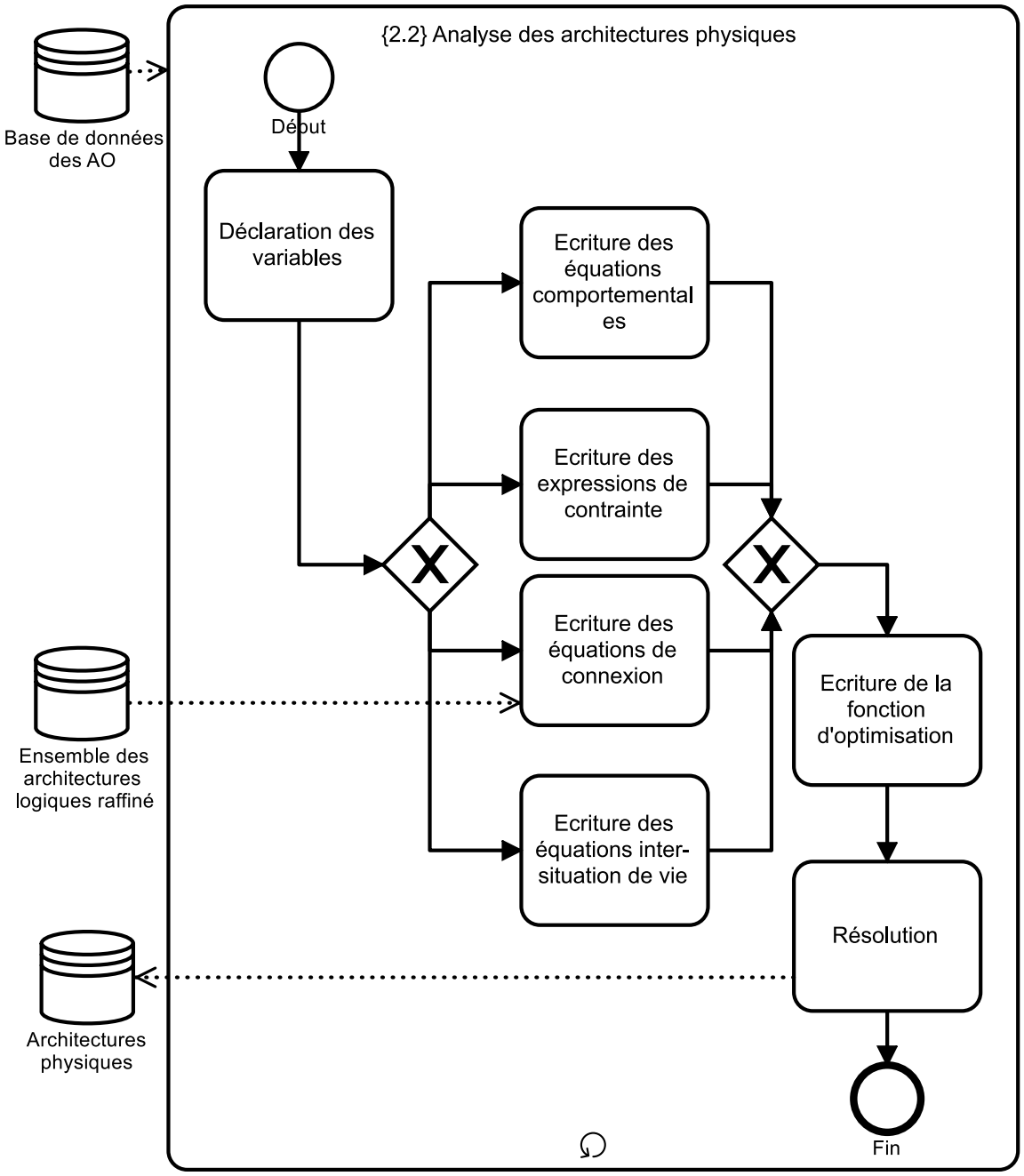


FIGURE 5.1 – Analyse des architectures physiques.

5.2 Vue physique de l'AO

La vue physique est utilisée lors de la phase d'analyse des architectures physiques. Elle prend en compte toutes les contraintes de la vue logique et y rajoute toute une liste d'attributs permettant l'écriture d'équations non linéaires et non différentielles comme cela est visible sur la Figure 1.2 du chapitre 1, page 5. Un AO est composé des mêmes attributs que précédemment plus :

- une liste de variables d'état, chaque variable ayant elle-même plusieurs attributs qui sont :
  - une unité par variable d'état,
  - une borne basse pour la valeur que pourra prendre la variable d'état,
  - une borne haute pour la valeur que pourra prendre la variable d'état,
  - une liste de contraintes associées à chaque variable d'état pour chaque situation de vie,
  - un booléen pour savoir si la variable d'état est associative ou non,
  - et une variable de choix pour savoir si la variable d'état globale est la somme, ou le max des sous-variables d'état pour chaque situation de vie,
- une liste d'équations comportementales pour chaque situation de vie,
- une liste d'équations permettant de lier les variables d'état et d'échange entre les différentes situation de vie,
- et une liste de ports ayant eux-mêmes leur propre vue physique composée d'une liste de variables d'échange ayant pour attributs :
  - une unité par variable d'échange,
  - une borne basse pour la valeur que pourra prendre la variable d'échange,
  - une borne haute pour la valeur que pourra prendre la variable d'échange,
  - une liste de contraintes associées à chaque variable d'échange pour chaque situation de vie,
  - et un booléen pour savoir si la variable d'échange est associative ou non.

5.3 La définition du modèle physique

Un modèle physique est composé d'équations de catégories différentes, liant des variables représentant les sous-systèmes et leurs échanges de flux. La définition du modèle démarre par la déclaration de variables dans $\mathbb{R}$ avec leurs domaines de validité. Ces variables sont de différents types, on compte :

- les variables d'état qui définissent les paramètres internes aux AO,
- et les variables d'échange qui définissent les composantes des flux échangés et leur magnitude.

Ces variables d'état sont encore rangées en trois types : les variables globales, les variables liées aux situations de vie et la variable d'optimisation.

- Les variables globales sont les vraies valeurs prises pour le dimensionnement des AO.
- Les variables liées aux situations de vie sont différentes pour chaque situation de vie, mais sont traitées pour former les variables globales,

— Enfin, la variable d'optimisation est une somme de variables globales de même type et est partie prenante de la fonction d'optimisation.

Les variables d'état du système à concevoir peuvent, soit, imposer leurs valeurs pour chaque situation de vie aux variables du même type des AO qui vont composer le système, soit collecter les valeurs des variables d'état globales des AO du même type. Les précédentes variables sont utilisées dans les équations suivantes :

— les équations comportementales, qui sont des équations modélisant le fonctionnement interne d'un AO ou du système à concevoir. (On peut voir ces équations comme des fonctions de transfert entre des entrées et des sorties.),

— les équations de contrainte, qui s'appliquent à tous les types de variables (Cela peut correspondre à des exigences de performance, comme décrit dans le chapitre 3.3 page 49),

— les équations de connexion, qui lient les variables d'échange d'après les connexions des architectures logiques,

— les équations intersituation de vie,

— et les équations qui imposent les valeurs des variables d'état du système à concevoir, telle que la variable «temps», ou qui collectent les variables d'état communes, comme la variable «masse».

Toutes les équations présentées sont définies pour chaque situation de vie, sauf les équations intersituation de vie, qui servent justement à faire le lien entre certaines variables, définies uniquement dans des situations de vie distinctes.
L'AO BTP (Boîte de Transmission Principale) est traité différemment des autres lors de sa définition physique. En effet, il a la particularité de gérer des flux de même type, mais avec des magnitudes différentes. Un algorithme spécial de traitement pour l'objet BTP a été créé pour générer un nouvel ensemble de variables par connexion reçue ou donnée par l'AO. Prenons l'exemple d'une BTP qui a deux arbres d'entrée et un arbre de sortie. Dans ce cas la vitesse de rotation de l'arbre «entrée 1» et celle de l'arbre «entrée 2» peuvent être différentes. Le logiciel va donc créer deux ensembles de variables identiques dans leurs définitions pour les arbres d'entrée. Ces variables sont identiques dans leur définition, mais peuvent être instanciées différemment. De même les variables d'état vont aussi être doublées pour avoir deux rapports de réduction différents : un entre l'arbre «entrée 1» et la sortie et l'autre entre l'arbre «entrée 2» et la sortie.

5.3.1 Les variables d'état

Les variables d'état sont des paramètres dimensionnant des AO. Les variables d'état globales sont les vrais paramètres dimensionnants, qui regroupent les variables d'état de chaque situation de vie. Les variables d'état définies dans les AO sont instanciées plusieurs fois dans le modèle physique : une fois en tant que variable d'état globale, et au plus une fois par situation de vie. Si la variable d'état n'est pas utilisée dans une certaine situation de vie, alors elle n'est pas instanciée dans la sous-partie du modèle physique associée à la situation de vie. Une autre instanciation est créée si la variable d'état est signalée par l'utilisateur comme la variable à minimiser ou à maximiser dans la fonction d'optimisation. Pour rappel, une instanciation est la création d'un objet émanant d'une classe. L'exemple souvent utilisé pour expliquer ce concept du langage objet est le plan de la maison et la maison elle-même. La classe correspond au plan et l'objet à la maison. Le fait de créer plusieurs objets venant d'une même classe reviendrait ici à créer un lotissement de maisons suivant le même plan.

Chaque variable d'état possède un attribut pour savoir si son instanciation globale est égale à la somme ou au maximum de ses instanciations par situation de vie.
Une variable à optimiser est créée en sommant toutes les variables d'état globales. Cette dernière est renseignée par l'utilisateur, elle porte souvent sur la masse totale du système ou sur la consommation totale de fuel. La Figure 5.2 illustre l'interdépendance des différents types de variables d'état.

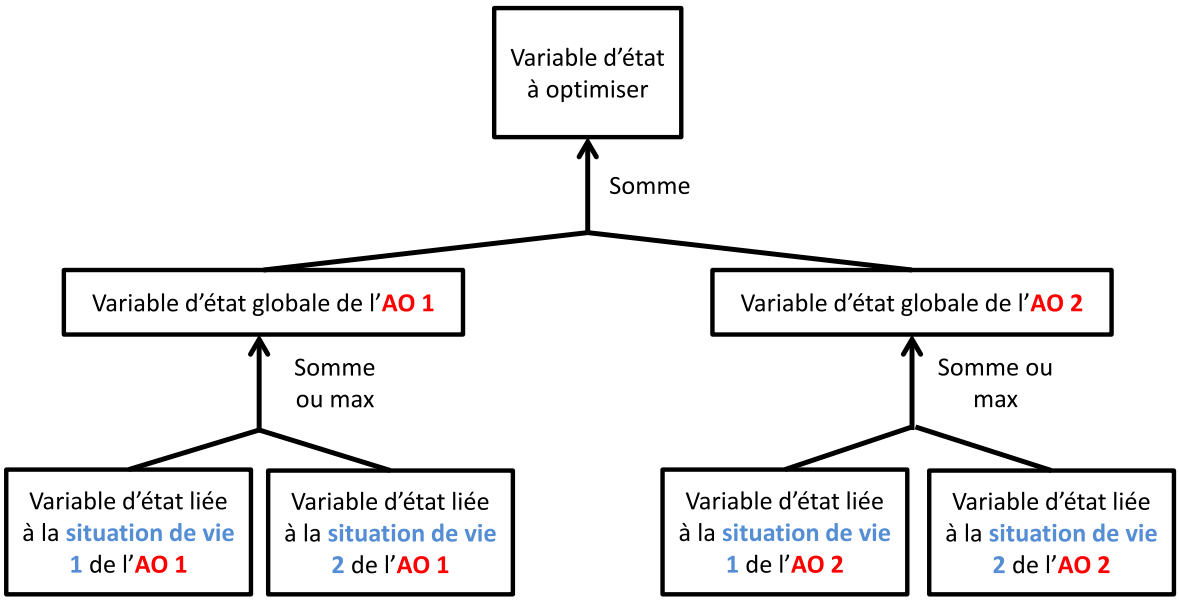


FIGURE 5.2 – Interdépendance des différents types de variables d'état

Concernant la définition des variables d'état du système à concevoir, elle est un peu différente. Ce sont usuellement ces variables sur lesquelles s'appliquent les exigences de performance, comme décrit dans le chapitre 3.3, page 49, et non pas sur les variables d'état des AO. Ces variables d'état du système sont forcément liées aux variables d'état des AO et peuvent, soit collecter les valeurs des variables d'état globales des AO, soit imposer différentes valeurs, suivant la situation de vie considérée. La Figure 5.3 explicite le concept de variable d'état imposant sa valeur ou au contraire collectant les valeurs des variables d'état des sous-systèmes.

Pour savoir quelles sont les variables à sommer ou alors lesquelles doivent recevoir une valeur imposée depuis la définition du système, les variables de même type sont nommées de manière similaire.

Reprenons chaque type de variable d'état pour spécifier son nommage, avec en italique, les parties changeantes suivant la variable, l'AO et la situation de vie considérés.

- Variable d'état d'un AO
  - Variable d'état globale : *nom de la variable_nom de l'AO*
  - Variable d'état liée à une situation de vie : *nom de la variable_nom de l'AO_nom de la situation de vie*
- Variable d'état du système
  - Variable d'état «collecte» : *nom de la variable*_Function
  - Variable d'état «impose» par situation de vie : *nom de la variable*_Function_*nom de la situation de vie*
  - Variable d'état à optimiser : *nom de la variable*

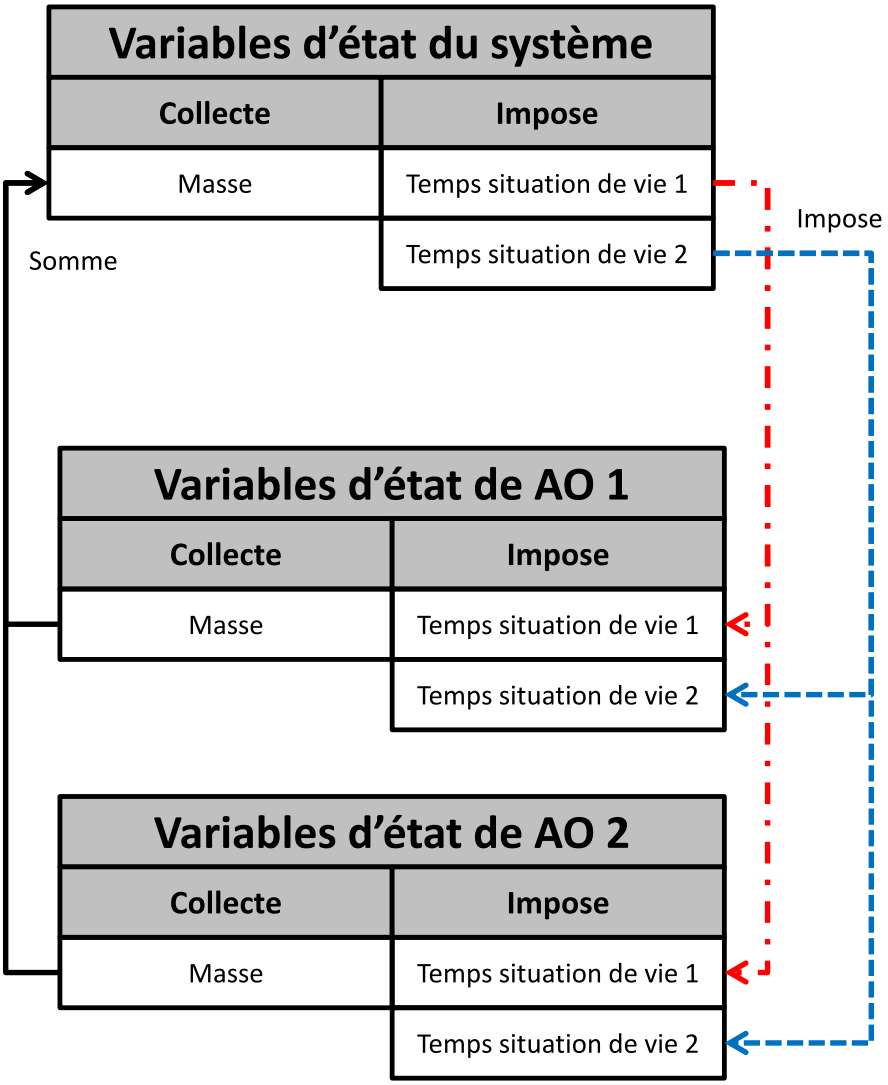


FIGURE 5.3 – Illustration des deux types de variables d'état du système

Concernant les variables d'état de l'AO BTP, dans l'architecture physique, elles sont créées un certain nombre de fois suivant le nombre de connexions dans l'architecture logique étudiée. Ces variables d'état sont créées m fois tel que

$$m = m_e \times m_s \tag{5.1}$$

avec m_e le nombre de connexions au port d'entrée de l'AO BTP et m_s le nombre de connexions au port de sortie. Les variables d'état de l'AO BTP sont : le rapport de réduction r, le rendement n et la masse M. Le logiciel créée donc m fois les variables r et n. La masse M n'est pas prise en considération pour la multiplication des variables d'état, en effet, l'équation permettant le calcul de cette masse n'est pas déclinable pour chaque couple entrée/sortie de la BTP.

5.3.2 Les variables d'échange

Les variables d'échange caractérisent les flux. Elles ne sont pas instanciées de la même façon que les variables d'état car il n'y a pas d'intérêt à avoir des variables d'échange globales. En effet, il s'agit de flux qui peuvent varier pour chaque situation de vie et qui ne sont pas sommables entre situations de vie. Néanmoins, ces variables d'échange sont définies dans chaque port et instanciées pour chaque situation de vie dans lequel le port conteneur est activé, c'est-à-dire connecté. Notons que ce sont les variables d'échange des ports d'un AO qui permettent le calcul des variables d'état à travers les équations comportementales de l'AO.

Chaque port possède un ensemble de variables d'échange comme attributs et tous les ports de même type possèdent les mêmes variables d'échange. La différentiation sera faite dans le nommage. Reprenons chaque type de variable d'état pour spécifier son nommage, avec en italique, les parties changeantes suivant la variable, le port, l'AO et la situation de vie considérés.

— Variable d'échange d'un AO : *nom du port_nom de l'AO_nom de la variable_nom de la situation de vie*

— Variable d'échange du système : *nom du port*_Function_*nom de la variable_nom de la situation de vie*

Concernant les variables d'échange de l'AO BTP dans l'architecture physique, elles sont créées un certain nombre de fois suivant le nombre de connexions dans l'architecture logique étudiée. Ces variables d'état sont créées m_e fois pour les variables d'échange d'entrée et m_s fois pour les variables d'échange de sortie avec m_e le nombre de connexions au port d'entrée de l'AO BTP et m_s le nombre de connexions au port de sortie. Les variables d'échange de l'AO BTP sont : la vitesse de rotation, le couple et la puissance à chaque arbre.

5.3.3 Les équations comportementales

Les équations comportementales sont contenues dans la vue physique des AO. Elles permettent de lier les différentes variables d'échange aux variables d'état. L'utilisateur peut rentrer autant d'équations comportementales qu'il le souhaite, il doit par contre les renseigner pour chaque mode d'utilisation de l'AO. Suivant la situation de vie du système à concevoir, chaque AO sera dans un mode différent et, à ce mode, sera associé un ensemble d'équations comportementales. Au global, l'architecture physique sera constituée d'autant de systèmes d'équations comportementales qu'il y a de situations de vie.
Les équations comportementales sont des équations non différentielles car elles doivent

être résolues par un solveur CSP qui n'est pas capable de résoudre des équations différentielles. Cela impose donc des simplifications, mais cette partie du logiciel n'a qu'une vocation de prédimensionnement, ou encore de présélection des variables d'état. Une fois que ces dernières sont choisies, il est tout à fait possible de simuler un modèle plus complexe reprenant les paramètres calculés jusqu'à présent.
Ces équations peuvent donc mettre en jeu les opérations classiques «=», «+», «−», «∗», «/», «$\sqrt{}$» et toutes les autres puissances, «sin», «arcsin», etc...
Concernant les équations comportementales de l'AO BTP dans l'architecture physique, elles sont créées un certain nombre de fois suivant le nombre de connexions dans l'architecture logique étudiée. Ces équations comportementales sont créées m_e fois, ou m_s fois ou m fois suivant le cas tel que présenté sur l'équation 5.1 page 118 avec m_e le nombre de connexions au port d'entrée de l'AO BTP et m_s le nombre de connexions au port de sortie.
Les équations comportementales liant le port d'entrée au port de sortie, sont créées m fois pour avoir la combinatoire nécessaire entre les différents arbres d'entrée et de sortie nécessaires de la BTP.
Les équations comportementales ne liant que des variables d'un même port, qu'il soit d'entrée ou de sortie, sont créées m_e fois ou m_s fois suivant le cas.
Toutes les équations comportementales doivent être normalisées par l'utilisateur lors de la création de la base de données des AO pour que la résolution du CSP puisse se faire efficacement.

5.3.4 Les équations de contrainte

Les équations de contrainte sont renseignées à l'intérieur même de l'objet «variable», que ce soit une variable d'état ou une variable d'échange. En effet, comme décrit au chapitre 5.2, page 114, des contraintes qui peuvent être liées à des exigences de performance sont directement renseignées par l'utilisateur. Ces contraintes sont simplement reprises et appliquées à la variable dans laquelle elles ont été définies. Il s'agit de contraintes numériques, telles que supérieur ou égal, inférieur ou égal, égal ou non égal. Plusieurs contraintes peuvent être renseignées pour une seule variable. Ces exigences de performance sont, bien entendu, appliquées uniquement sur les variables du système à concevoir. Concernant les variables des AO, elles se verront appliquer les exigences de contrainte, soit au travers d'équations de connexion en étant connectées aux variables du système à concevoir, soit, si elles sont de même type qu'une variable d'état du système imposant ses valeurs, elles recevront les mêmes contraintes que cette variable du système à concevoir. En effet, c'est le système, dans son ensemble, qui a été spécifié durant la phase d'extraction des besoins et de transformation des besoins en exigences. De ce fait, aucun AO n'aura des exigences de performance qui lui seront directement allouées, car au moment de la spécification, l'intérieur du système n'est pas encore connu. Les variables d'échange ou d'état des AO ne reçoivent donc pas directement des contraintes venant des exigences de performance.
Un cas contraire mérite d'être soulevé, qui est le cas des variables d'état des AO liées à une situation de vie et identiques à une seule ou plusieurs variable(s) d'état «impose» du système. Par exemple, la variable d'état «temps de fonctionnement» peut être spécifiée à 1h00 au niveau système. Cette variable d'état est du type «impose», ce qui veut dire que toutes les variables d'état liées à une situation de vie ayant le même nom, recevrons la même exigence de performance. Dans ce cas, un AO moteur thermique aura sa variable d'état «temps de fonctionnement» contrainte par l'exigence de performance appliquée au système complet.
Concernant les équations de contrainte de l'AO BTP dans l'architecture physique, elles

sont créées un certain nombre de fois suivant le nombre de connexions dans l'architecture logique étudiée. Ces équations de contrainte sont créées m_e fois, ou m_s fois ou m fois suivant le cas tel que présenté sur l'équation 5.1 page 118 avec m_e le nombre de connexions au port d'entrée de l'AO BTP et m_s le nombre de connexions au port de sortie.

5.3.5 Les équations de connexion

Les équations de connexion sont construites à l'aide de l'analyse des différentes vues relatives aux situations de vie du système à concevoir. Une architecture logique a autant de vues que de situations de vie considérés. A chaque vue correspond trois matrices DSM : une matrice pour les connexions internes au système, une matrice pour les connexions vers les ports d'entrée et une matrice pour les connexions vers les ports de sortie du système à concevoir. Suivant la situation de vie du système à concevoir, chaque AO sera dans un mode différent et, à ce mode, est associé un ensemble de variables d'échange, différent suivant les ports actifs de l'AO dans le mode considéré. Au global, l'architecture physique sera constituée d'autant de systèmes d'équations de connexion qu'il y a de situations de vie.
Lorsque deux ports sont connectés, toutes les variables d'échange contenues dans ce port doivent faire partie d'une même équation. C'est-à-dire que pour chaque chaîne de connexion, un bilan est fait par type de variable. Rappelons que si deux ports sont connectés, ils ont forcément la même définition, à l'exception de leur direction et de leur appartenance à des AO différents. Dans ce cas, ces ports ont forcément les mêmes types de variables d'échange, qui peuvent donc être rajoutées dans le bilan. Dans ce type d'équation, il n'y aura pas de variable d'état.
Concernant les équations de connexion mettant en jeu l'AO BTP dans l'architecture physique, les nouvelles variables d'échange sont utilisées. Par les nouvelles variables d'échange, il s'entend les variables créées m_e fois, le nombre de connexions au port d'entrée de l'AO BTP et m_s fois, le nombre de connexions au port de sortie. Chaque équation de connexion correspond à une des connexions qui a permis la création de la nouvelle variable d'échange. Les équations de connexions sont automatiquement normalisées par le logiciel de synthèse architecturale. Cette normalisation est nécessaire pour que le solveur soit efficace dans sa recherche de l'optimum recherché.

5.3.6 Les équations intersituation de vie

Les variables d'état ou d'échange sont définies pour chaque situation de vie et, jusqu'à présent, aucune passerelle entre les situations de vie n'a été exposée, mis à part celle créée par la variable d'état globale, qui capitalise l'ensemble de ses sous-variables d'état par situation de vie. Il peut être intéressant de lier des modèles d'un même AO, mais dans des modes différents, par des équations simples issues d'analyses de l'existant. Par exemple, on peut se rendre compte de l'existence d'une relation entre la puissance mécanique nécessaire pour démarrer un moteur à combustion et sa puissance mécanique maximale émise en fonctionnement. Sans ce type d'équation intersituation, une telle équation ne pourrait pas être exprimée au sein d'un AO car chacune de ces puissances est définie dans un système d'équations comportementales dédié à chaque mode de fonctionnement de l'AO. Le logiciel laisse donc la possibilité de lier deux variables d'échange appartenant à deux ports différents, qui ne seraient actives que dans un seul mode, différent pour chaque variable.
Pour illustrer cette définition, il est possible d'observer l'Annexe A.2.2, page XIX et, en

particulier, les ports «P1» et «P2» de l'AO «MoteurTher» qui ne sont actifs respectivement que dans le mode 1 et 2.
Toutes les équations intersituation de vie doivent être normalisées par l'utilisateur lors de la création de la base de données des AO pour que la résolution du CSP puisse se faire efficacement.

5.3.7 Les équations de collecte

Les équations de collecte servent à plusieurs tâches, mais ont globalement pour but le rassemblement de plusieurs variables de même type. Ces équations sont de types différents, listés ci-après :

— Équation de collecte des variables d'état du système à concevoir depuis les variables d'état globales des AO,

— Équation de collecte des variables d'état globales des AO depuis les variables d'état par situation de vie. (Ces équations peuvent être soit une somme, soit un maximum sur toutes les variables d'état par situation de vie.),

— Équation de collecte de la variable d'état du système à optimiser (Cette équation est une égalité avec une variable d'état de collecte du système à concevoir).

Les équations de collecte sont automatiquement normalisées par le logiciel de synthèse architecturale. Cette normalisation est nécessaire pour que le solveur soit efficace dans sa recherche de l'optimum recherché.

5.4 La génération des modèles physiques

La définition des modèles précédents est faite par l'utilisateur avant de lancer la résolution automatique du problème de synthèse. L'utilisateur a donc dû rentrer la définition de chaque AO avec ses variables d'état, ses équations comportementales pour chaque mode de fonctionnement, ses équations intersituation de vie et ses ports. Eux-mêmes contiennent les variables d'échange, aussi définies à ce moment du processus global de synthèse. De même, l'utilisateur a entièrement défini le système à concevoir vis-à-vis des équations comportementales attendues ; les contraintes sur les variables d'état et d'échange émanant des exigences de performance. Il est à noter que les équations comportementales du système ne doivent pas prendre en compte les AO utilisés pour instancier le système à concevoir. Ce sont des équations qui considèrent le système comme une boîte noire.
Toutes les étapes présentées dans cette section sont faites de manière automatique à l'aide d'un code C++. Toutes les étapes de définition des modèles physiques doivent être faites manuellement par l'utilisateur lors de la création de la base de données des AO et de la définition du système à concevoir. Mais comme dit précédemment, le modèle global qui lie toutes les équations des AO dans toutes les situations de vie est généré automatiquement. Ce dernier point est une nécessité car lors de la synthèse d'un système complexe, des centaines d'architecture logique peuvent exister et il n'est pas concevable que l'utilisateur construise manuellement tous les modèles physiques de ces solutions architecturales.
Pour résumer, précédemment, chaque modèle a été déclaré comme étant isolé. Mais maintenant, c'est le modèle pour l'ensemble de l'architecture qui est construit. La construction des modèles pour l'ensemble de l'architecture, appelés architectures physiques est une agrégation des différents modèles des AO. La difficulté réside dans la création de liens entre les modèles isolés des AO et la prise en compte des différentes situations de vie.

5.4.1 La construction des modèles vis-à-vis des équations de comportement

Dans un premier temps, pour chaque situation de vie, chaque AO, s'il est connecté, est utilisé pour récupérer les équations comportementales qu'il contient. Pour rappel, un AO a plusieurs ensembles d'équations comportementales, en fait, il en a un par mode. Il est donc nécessaire de trouver les modes de fonctionnement de l'AO associés aux situations de vie, pour ensuite récupérer la bonne liste des équations comportementales.
Dans un second temps, les équations sont réécrites avec le bon nommage pour les variables d'échange et les variables d'état associées à chaque situation de vie. Les variables d'état globales n'apparaissent pas dans ces équations. Elles proviennent ensuite de la collecte de toutes les sous-variables d'état associées à chaque situation de vie. Les variables d'état liées à une situation de vie, utilisées dans les équations comportementales, sont stockées dans un vecteur C++. En effet, pour rappel, toutes les variables d'état globales ne sont pas décomposées en autant de variables d'état associées aux situations de vie qu'il y a de situations de vie. Dans certains cas, il peut y avoir des situations de vie où la variable d'état n'apparaît pas dans les équations comportementales. Dans ce cas, il ne faudra pas déclarer la variable dans le modèle.
Les équations comportementales du système à concevoir sont traitées de la même façon que les équations comportementales des AO. Une différence vient du fait que les variables d'état sont de deux types différents :

— «collecte»,

— et «impose».

Les variables nommées «collecte» restent globales et celles nommées «impose» sont uniquement liées à une situation de vie. Ce point de détail est important dans le nommage des variables et dans leur déclaration. Par exemple, il ne faudra donc pas déclarer de variables d'état globales pour celles ayant l'attribut «impose».

5.4.2 La construction des modèles vis-à-vis des équations de contrainte

Pour chaque variable d'échange utilisée dans une équation de connexion, des équations de contrainte lui sont appliquées. Pour ce faire, le mode dans lequel l'AO est utilisé dans la situation de vie actuelle est récupéré, et la liste des contraintes associées à la variable en question et au mode de l'AO est enregistrée. Une simple boucle permet d'écrire toutes les équations de contrainte s'appliquant à la variable d'échange dans cette situation de vie.
Pour chaque variable d'état utilisée dans des équations comportementales, un ensemble d'équations de contrainte émanant de la définition de l'AO père sont écrites.

— Variable d'état d'un AO
  — Variable d'état globale : aucune équation de contrainte n'est écrite pour ce type de variable, car il s'agira de faire une somme ou de prendre le maximum des sous-variables d'état par situation de vie.
  — Variable d'état liée à une situation de vie : un ensemble d'équations de contrainte seront écrites d'après la définition de l'AO.

— Variable d'état du système
  — Variable d'état «collecte» : aucune équation de contrainte n'est écrite pour ce type de variable car il s'agit d'une somme de variables d'état globales, elles-mêmes étant des sommes de variables d'état par situation de vie.

— Variable d'état «impose» par situation de vie : une équation contrainte par situation de vie sera écrite. C'est ici que les exigences de performance sur des variables d'état du système à concevoir peuvent être exprimées.

5.4.3 La construction des modèles vis-à-vis des connexions des architectures logiques

Pour chaque architecture logique et chaque situation de vie, un algorithme permet d'écrire des équations de connexion à partir de l'étude d'une matrice DSM étendue. Ce concept permet d'écrire les équations de connexion d'une architecture logique à l'aide d'une seule matrice qui regroupe la DSM interne et les DSM d'entrée et de sortie du système à concevoir.

La construction des modèles de connexion sans prise en compte des situations de vie

La DSM étendue est parcourue ligne par ligne et colonne par colonne. Quand une connexion est trouvée, ses coordonnées sont enregistrées. Ensuite, toutes les autres connexions associées à ces mêmes ports sont recherchées. Leurs coordonnées sont enregistrées, à leur tour, et ainsi de suite. A chaque fois que de nouvelles connexions sont trouvées, on vérifie qu'elles n'ont pas déjà été enregistrées. Pour ce faire, il faut regarder si les coordonnées de la connexion sur la DSM étendue ont déjà été enregistrées. Si elles ont déjà été enregistrées, elles sont supprimées. A la sortie de ce sous-algorithme, pour chaque connexion dans la DSM étendue, sont enregistrées les autres connexions mettant en jeu les mêmes ports que ceux de la connexion étudiée.
Une fois que les connexions utiles pour chaque bilan sur les variables d'échange sont connues, les équations de connexion sont écrites en soustrayant les variables entre elles et en mettant cette soustraction égale à zéro. La figure 5.4 permet de visualiser les actions derrière les algorithmes. La matrice représentée est une DSM étendue se focalisant sur les ensembles de variables d'échange contenus dans chaque port. Tout d'abord, la première connexion est trouvée entre les ensembles v_2 et v_3. Sur la même ligne se trouve une seule connexion entre v_2 et v_5 et pas d'autre connexion sur la colonne. En suivant la chaîne des connexions liées à la connexion étudiée, nous trouvons, sur la même colonne que la connexion v_2 et v_5, la connexion v_4 et v_5. Puis, sur le même principe, nous décelons la connexion v_4 et v_7.
Après cette première étape d'identification de la chaîne de connexions, l'équation de connexion est créée. Nous partons de l'ensemble de variables v_2, auquel nous soustrayons l'ensemble v_3. Étant donné que la connexion suivante se passe aussi avec v_2, alors v_5 est aussi soustrait dans l'équation bilan. Cette fois, par contre, nous ne soustrayons pas v_4 car il vient s'additionner au reste de l'équation. Pour finir, v_7 est aussi soustrait.
Sur un exemple réel, cette méthodologie est appliquée pour chaque connexion présente dans la DSM étendue et les chaînes de connexion sont recherchées à chaque fois. Si une chaîne a déjà été trouvée, l'équation n'est pas réécrite. Lors de la construction des équations de connexion, chaque variable utilisée est renommée avec son vrai nommage, prenant en compte la situation de vie dans laquelle elle est utilisée. Ces noms de variables utilisées sont tous stockés pour savoir quelles variables doivent être déclarées dans le modèle physique et quelles variables seront soumises aux contraintes déclarées par l'utilisateur lors de la définition de chaque AO.

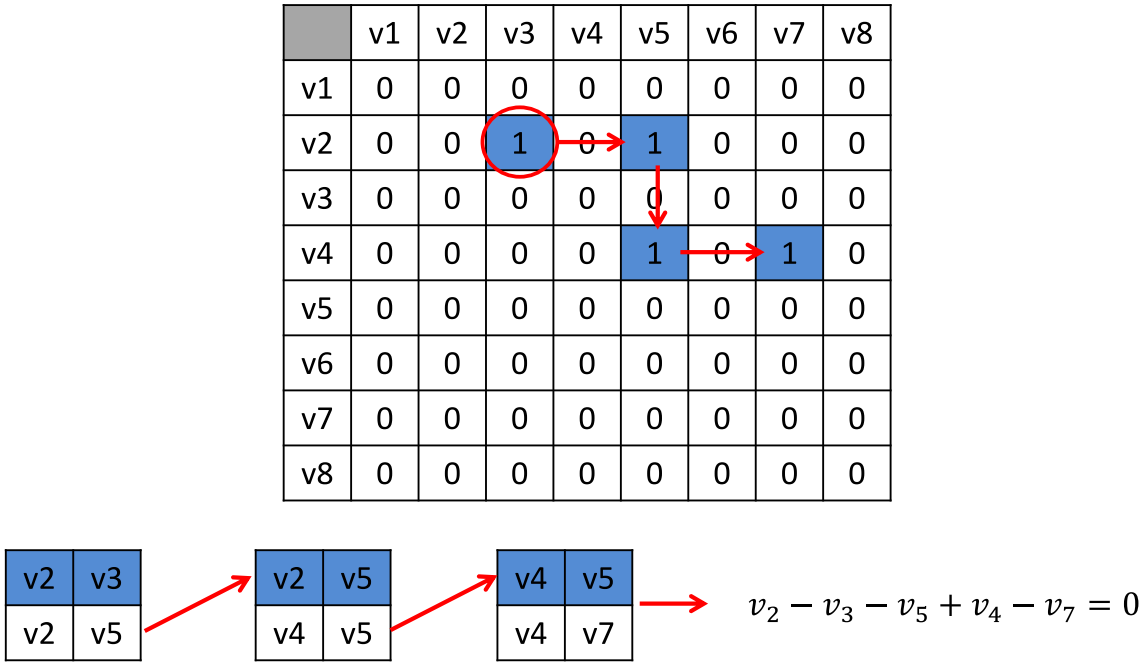


FIGURE 5.4 – Méthode d'écriture des équations de connexion.

La prise en compte des différentes situations de vie

Les différentes situations de vie sont prises en compte dans le nom des variables d'échange qui porte un indice "uci" pour la situation de vie "i". De même, les DSM étendues qui sont traitées ne sont pas des DSM étendues globales, mais bien des DSM étendues sous la vue des situations de vie étudiés. Cela veut dire que pour traiter une architecture physique, il faut utiliser cet algorithme d'écriture des équations de connexion autant de fois qu'il y a de situations de vie étudiés, et que le modèle associé sera composé des sous-modèles correspondant aux connexions présentes dans chaque situation de vie.

5.4.4 La construction des modèles vis-à-vis des équations intersituation de vie

Pour chaque AO connecté dans au moins deux situations de vie, on enregistre ses équations intersituation de vie. Toutes les variables d'échange créées avec un bon nommage, c'est-à-dire celui contenant l'AO d'origine, le port d'origine et la situation de vie d'origine, ont été enregistrées dans un vecteur C++ temporaire. Ce dernier est utilisé maintenant pour réécrire les équations intersituation de vie avec le bon nommage et pour vérifier si elles répondent bien à l'exigence de lier uniquement des variables utilisées dans une seule situation de vie et dans des situations de vie différentes. En effet, lors de la récupération de l'équation intersituation de vie, cette dernière contient uniquement des variables nommées par leur simple nom et pas par leur nom complet. Pour rappel, le nom complet est de la forme suivante : *nom du port_nom de l'AO_nom de la variable_nom de la situation de vie.*
Si les variables d'échange présentes dans les équations ne sont utilisées ni dans une seule situation de vie ni dans des situations de vie différentes, alors un message d'erreur est renvoyé à l'utilisateur pour lui dire que la définition d'un de ses AO n'est pas juste.

5.4.5 La construction des modèles vis-à-vis des équations de collecte

Comme le lecteur a pu le constater, les variables d'état constituent la pierre angulaire de l'architecture physique. Elles sont déterminées à travers les équations comportementales et les équations de synthétisation, appelées ici «équations de collecte». Voici la liste des différentes variables d'état du modèle :

- Variable d'état d'un AO
  - La variable d'état globale est la somme des variables d'état liées à une situation de vie existantes, ou la valeur maximale de ces variables.
- Variable d'état du système
  - La variable d'état «collecte» est la somme des variables d'état globales des AO connectés dans la solution architecturale.
  - La variable d'état à optimiser est l'une des variables d'état «collecte» du système à concevoir.

5.5 La résolution des modèles physiques

La résolution des modèles physiques se fait à l'aide du langage MinIbex [1]. Chaque variable de décision possède un domaine $\mathbb{D} \in \mathbb{R}$ bien défini entre deux bornes, ce qui permet de réduire le temps de résolution. Les domaines $\mathbb{D}$ de variation des variables d'état ou d'échange sont définis par l'utilisateur lors de la définition de la vue physique des AO. La précision de la résolution, c'est-à-dire le nombre de chiffres significatifs, peut être changée à loisir, tout comme le temps maximal de résolution pour chaque architecture physique. Ici, contrairement aux résolutions précédentes réalisées à l'aide de CSP, il s'agit d'un problème d'optimisation et non pas d'obtention de toutes les solutions. Si le processus de calcul est arrêté avant la fin de l'optimisation, cela veut dire que la solution trouvée ne sera pas forcément l'optimum car tout le champ des possibles n'aura pas encore été parcouru.
L'optimisation qui est conduite ici est mono-objectif. Elle s'applique sur un type de variable défini par l'utilisateur comme par exemple la masse ou encore la consommation de carburant. La variable à optimiser est en faite une des variables «Collecte» du système à concevoir. Elle correspond donc à la somme des variables de même type contenues dans les AO du système à concevoir. La variable à optimiser est donc liée à plusieurs AO du système, eux mêmes liés par des équations d'échange. Le système complet est donc optimisé pour répondre à un objectif. Pour faciliter le dépouillement pour l'utilisateur, les solutions architecturales sont ensuite triées simplement par rapport à la variable à optimiser.

5.6 Exemple fil rouge

5.6.1 Vue physique de l'AO

Premièrement, lors de la description de la vue physique des AO, chaque type de port doit être défini. Bien entendu, deux ports du même type doivent contenir les mêmes variables. Contrairement aux travaux de HIRTZ et collab. [2002] ou à la méthodologie bond graph, les variables ne sont pas rangées dans les trois types définis : «Puissance», «Effort» et «Flux». En effet, seules les variables nécessaires sont créées par l'utilisateur et ce dernier

1. http://www.ibex-lib.org/doc/minibex.html dernier accès le 30/11/2017

n'est pas obligé de définir pour un port trois variables de ces types.
Dans le cas de l'exemple, les ports de commande ne sont pas décrits physiquement, c'est-à-dire qu'aucune variable n'y est associée. Même si deux ports sont de même type, leurs bornes min et max peuvent avoir des valeurs différentes suivant l'AO auquel ils appartiennent. Cela peut permettre de réduire les domaines de variation des variables d'échange pour mieux coller à la réalité des technologies physiques, et aussi pour réduire le temps de calcul, afin d'obtenir la solution architecturale optimale lors de l'analyse physique. La définition physique des ports de l'AO «RotorArtZ» est donnée sur le Tableau 5.1. La définition physique de tous les ports de la base de données des AO est donnée en Annexe A.3, XXIV.

TABLEAU 5.1 – Définition physique des ports de l'AO RotorArtZ.

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque mode		Est associative ?
AO RotorArtZ						
Port PMecaRot						
PMeca	kW	0	500	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui
Omega	rpm	100	400	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	—
Couple	Nm	0	50000	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui
Port FxG						
FxG	N	0	100000	Mode 1 Mode 2 Mode 3	∅ = 0 = 0	Oui
PxG	kW	0	500	Mode 1 Mode 2 Mode 3	∅ = 0 = 0	Oui
Port FzG						
FzG	N	0	100000	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
PzG	kW	0	500	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
Port MzG						
MzG	Nm	0	2000	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui

Les variables définies dans le tableau 5.1, sont explicitées maintenant :

— PMeca : la puissance mécanique reçue par l'AO,
— Omega : la vitesse de rotation de l'arbre d'entrée du rotor,
— Couple : le couple de l'arbre d'entrée du rotor,
— FxG : l'effort produit par le rotor suivant l'axe x de l'aéronef,
— PxG : la puissance consommée par le rotor pour produire un effort suivant l'axe x de l'aéronef,
— FzG : l'effort produit par le rotor suivant l'axe z de l'aéronef,
— PzG : la puissance consommée par le rotor pour produire un effort suivant l'axe z de l'aéronef,
— MzG : le couple transmis par le rotor sur la structure.

Tous les AO sont définis par un ensemble de ports, mais dans la vue physique, les AO sont définis avec plus d'attributs. Des équations comportementales et variables d'état sont rajoutées à la définition. Le Tableau 5.2 représente la définition physique des variables d'état de l'AO «RotorArtZ», dont les ports ont été présentés, dans la vue physique, sur le tableau A.3. Enfin, les équations comportementales liant toutes les variables d'échange et d'état de l'AO sont écrites par mode. La vue physique globale de l'AO «RotorArtZ» est l'union de ces deux tableaux et des équations comportementales.

TABLEAU 5.2 – Définition physique des variables d'état de l'AO RotorArtZ.

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque mode	Est associative ?	Somme, max ou NA
AO RotorArtZ						
MAeroC	kg	0	5000	Mode 1 : ∅ Mode 2 : ∅ Mode 3 : ∅	Oui	max
l	m	0.5	10	Mode 1 : ∅ Mode 2 : ∅ Mode 3 : ∅	—	max
n	—	0	1	Mode 1 : = 0.7 Mode 2 : = 0.75 Mode 3 : ∅	—	max
Ptot	kW	0	500	Mode 1 : ∅ Mode 2 : ∅ Mode 3 : ∅	Oui	max
v	km/h	0	300	Mode 1 : ∅ Mode 2 : ∅ Mode 3 : ∅	—	imposée
CxAtot	m^2	0	5	Mode 1 : ∅ Mode 2 : ∅ Mode 3 : ∅	Oui	imposée
CxAElem	m^2	0	5	Mode 1 : ∅ Mode 2 : ∅ Mode 3 : ∅	Oui	max

Les variables définies dans le tableau 5.2, sont explicitées maintenant :

— MAeroC : la masse de l'AO qui sera collectée par la variable d'état du système,

— l : le rayon du rotor, pouvant être assimilé à la longueur des pales,

— n : le rendement du rotor entre la puissance totale du rotor et la puissance mécanique d'entrée,

— Ptot : la puissance globale fournie par le rotor,

— v : la vitesse à atteindre pour l'ensemble de l'aéronef,

— CxAtot : le coefficient de traînée multiplié par la surface impliquée dans la traînée pour l'ensemble de l'aéronef,

— CxAElem : le coefficient de traînée multiplié par la surface impliquée dans la traînée pour le rotor.

Les équations comportementales de l'AO «RotorArtZ» sont données ci-après par mode de fonctionnement :

— **Mode 1 -Vol d'avancement-** :

$$\mathrm{M}z\mathrm{G} = n \times \mathrm{C}ouple$$

Le couple transmis à la structure correspond à l'opposé du couple sur l'arbre du rotor. Dans la modélisation, les deux sont comptés positivement car C$ouple$ est vu comme reçu par le rotor et MzG comme reçu par la structure.

$$\mathrm{P}z\mathrm{G} \times 1000 = \mathrm{F}z\mathrm{G} \times \sqrt{-\frac{v^2}{2} + \frac{1}{2} \times \sqrt{v^4 + \left(\frac{\mathrm{F}z\mathrm{G}}{1.2 \times \pi \times l^2}\right)^2}}$$

Calcul de la puissance nécessaire au rotor pour fournir une poussée FzG, d'après la théorie de Froude.

$$\mathrm{P}x\mathrm{G} \times 1000 = \frac{1}{2} \times 1.2 \times \mathrm{C}x\mathrm{A}tot \times v^2 \times \sqrt{-\frac{v^2}{2} + \frac{1}{2} \times \sqrt{v^4 + (\frac{\mathrm{F}z\mathrm{G}}{1.2 \times \pi \times l^2})^2)}}$$

Calcul de la puissance nécessaire pour faire avancer l'aéronef à une vitesse v, avec 1.2 la masse volumique de l'air considérée.

$$n = \mathrm{P}tot/\mathrm{PM}eca$$

L'équation de rendement du rotor.

$$\mathrm{PM}eca = \mathrm{O}mega \times \mathrm{C}ouple \times 2 \times \frac{\pi}{1000 \times 60}$$

Le lien entre les différentes variables du port d'énergie mécanique de rotation

$$\mathrm{P}tot = \mathrm{P}z\mathrm{G} + \mathrm{P}x\mathrm{G}$$

La puissance totale du rotor est fonction de la puissance de sustentation et d'avancement.

$$\mathrm{MA}ero\mathrm{C} = f(\mathrm{P}tot)$$

La masse du rotor est fonction de la puissance transitant en son sein. L'équation n'est pas explicitée et celle qui est utilisée dans le manuscrit n'est pas liée à une étude de la variation du paramètre sur la flotte mondiale des hélicoptères pour des raisons de confidentialité.

$$\mathrm{C}x\mathrm{AE}lem = f(l, \mathrm{P}tot)$$

Le coefficient de traînée, multiplié par la surface mise en jeu dans la traînée, est fonction de la taille du rotor et donc de la longueur des pales et de la puissance transitant par le rotor. L'équation n'est pas explicitée et celle qui est utilisée dans le manuscrit n'est pas liée à une étude du paramètre sur la flotte mondiale des hélicoptères pour des raisons de confidentialité.

— **Mode 2 -Vol stationnaire-** :

$$\mathrm{M}z\mathrm{G} = n \times Couple$$

Le couple transmis à la structure correspond à l'opposé du couple sur l'arbre du rotor. Dans la modélisation, les deux sont comptés positivement car $Couple$ est vu comme reçu par le rotor et MzG comme reçu par la structure.

$$\mathrm{P}z\mathrm{G} \times 1000 = \frac{\mathrm{F}z\mathrm{G}^{\frac{3}{2}}}{\sqrt{2 \times 1.2 \times \pi \times l^2}}$$

Calcul de la puissance nécessaire au rotor pour fournir une poussée FzG, d'après la théorie de Froude.

$$n = \mathrm{P}tot / \mathrm{PM}eca$$

L'équation de rendement du rotor

$$\mathrm{PM}eca = Omega \times Couple \times 2 \times \frac{\pi}{1000 \times 60}$$

Le lien entre les différentes variables du port d'énergie mécanique de rotation

$$\mathrm{P}tot = \mathrm{P}z\mathrm{G}$$

La puissance totale du rotor est fonction de la puissance de sustentation et d'avancement, mais pour ce mode, la puissance d'avancement est nulle.

$$\mathrm{C}x\mathrm{AE}lem = f(l, \mathrm{P}tot)$$

Le coefficient de traînée, multiplié par la surface mise en jeu dans la traînée, est fonction de la taille du rotor et donc de la longueur des pales et de la puissance transitant par le rotor. L'équation n'est pas explicitée et celle qui est utilisée dans le manuscrit n'est pas liée à une étude du paramètre sur la flotte mondiale des hélicoptères pour des raisons de confidentialité.

— **Mode 3 -A l'arrêt-** :

$$\emptyset$$

Pas d'équations comportementales, car le rotor est à l'arrêt.

Les autres AO ne sont pas décrits dans ce manuscrit car l'objectif premier est de décrire la méthodologie de synthèse et non pas l'ensemble des modèles utilisés. Néanmoins, tous les autres AO sont modélisés de la même façon que «RotorArtZ» et sont utilisés dans les résultats présentés dans ce document.

5.7 Analyse des architectures physiques

5.7.1 L'exemple du groupe électrogène

La méthode qui a été présentée précédemment permet la création de modèles physiques appelés les architectures physiques. Ces modèles sont résolus à l'aide du logiciel Ibex[2]. Il se trouve qu'en complexifiant au fur et à mesure les modèles, les limites du logiciel Ibex ont été atteintes. L'analyse physique des solutions ne peut se faire que sur des problèmes ayant uniquement des variables du même ordre de grandeur ou alors sur des problèmes avec peu d'équations complexes.
L'exemple que nous avons utilisé jusqu'à présent n'est actuellement pas adapté à une résolution avec Ibex. En effet, les modèles CSP pouvant être résolus avec Ibex doivent contenir des équations simples ou alors des variables de décision homogènes, c'est-à-dire avec des ordres de grandeur similaires. Néanmoins, pour présenter un exemple, la synthèse d'un système groupe électrogène va être étudiée pour cette dernière étape. Les équations mises en jeu sont plus simples car elles se focalisent uniquement sur des conversions d'énergies et pas sur des bilans d'efforts et des calculs de puissances associées, comme pour l'exemple du drone précédemment traité.

Cet exemple est traité du début à la fin. Cela permet au lecteur de comprendre l'application de la méthodologie sur un exemple complet.

Identification des situations de vie et des parties prenantes

Le système à concevoir est un groupe électrogène. Appliquons la méthodologie pour la capture du besoin en faisant une première liste des parties prenantes et des situations de vie.
Parties prenantes :

- Environnement extérieur
- Réseau à alimenter
- Utilisateur
- Régulation

Situations de vie :

- Phase de démarrage
- Phase de recharge
- Phase de production électrique

Un diagramme d'environnement par phase de vie est ensuite réalisé. Trois diagrammes d'environnement sont donnés sur les Figures suivantes.
La Figure 5.5 représente l'environnement extérieur vu par le système lors de la phase de démarrage.
La Figure 5.6 représente l'environnement extérieur vu par le système lors de la phase de recharge.
La Figure 5.7 représente l'environnement extérieur vu par le système lors de la phase de production électrique.
La Figure 5.8 représente l'environnement extérieur vu par le système, toutes phases de vie confondues.

2. http://www.ibex-lib.org/ dernier accès le 30/11/2017

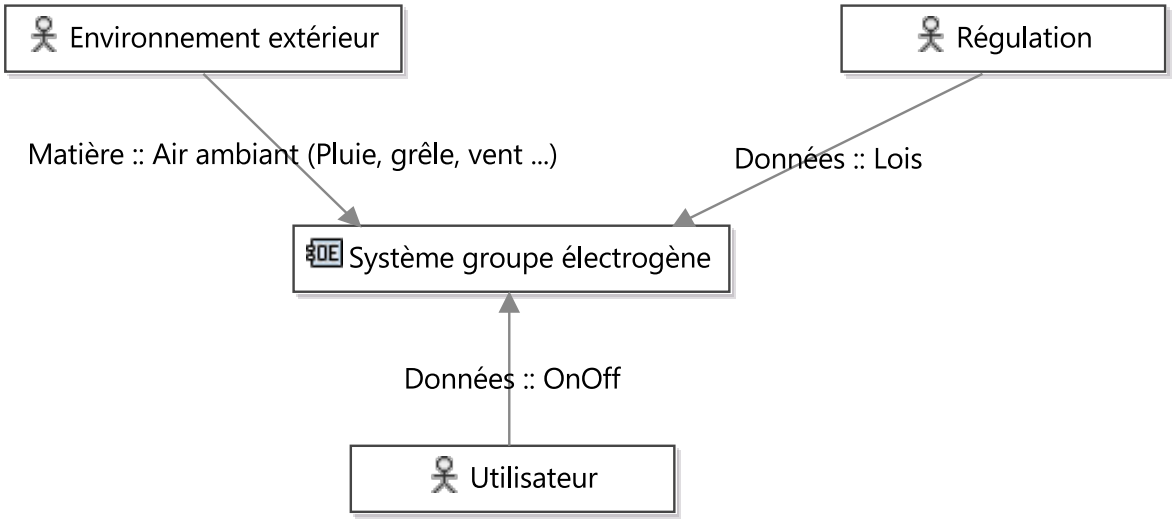


FIGURE 5.5 – Diagramme d'environnement pour la phase de démarrage.

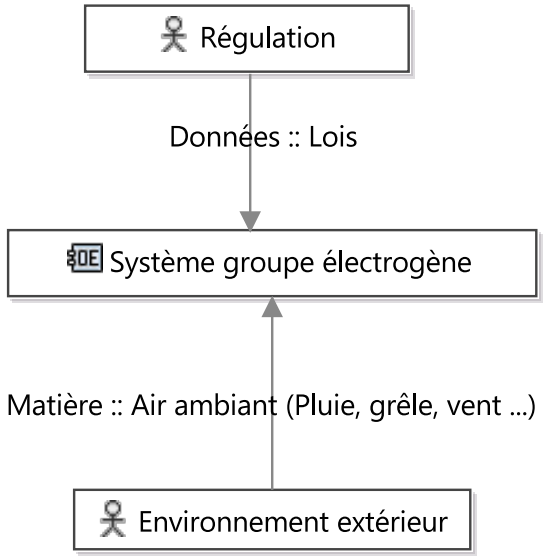


FIGURE 5.6 – Diagramme d'environnement pour la phase de recharge.

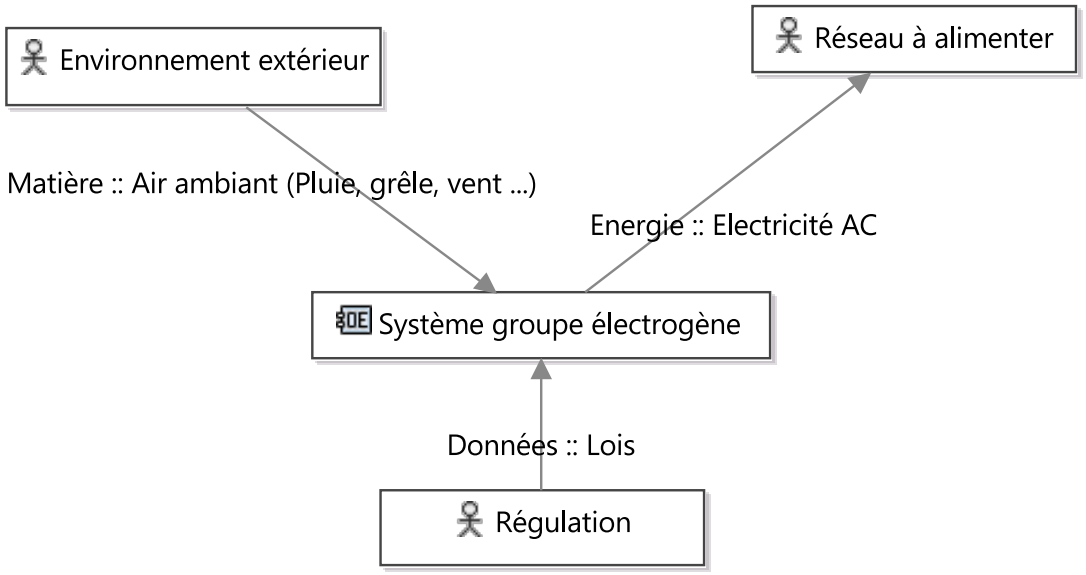


FIGURE 5.7 – Diagramme d'environnement pour la phase de production d'électricité.

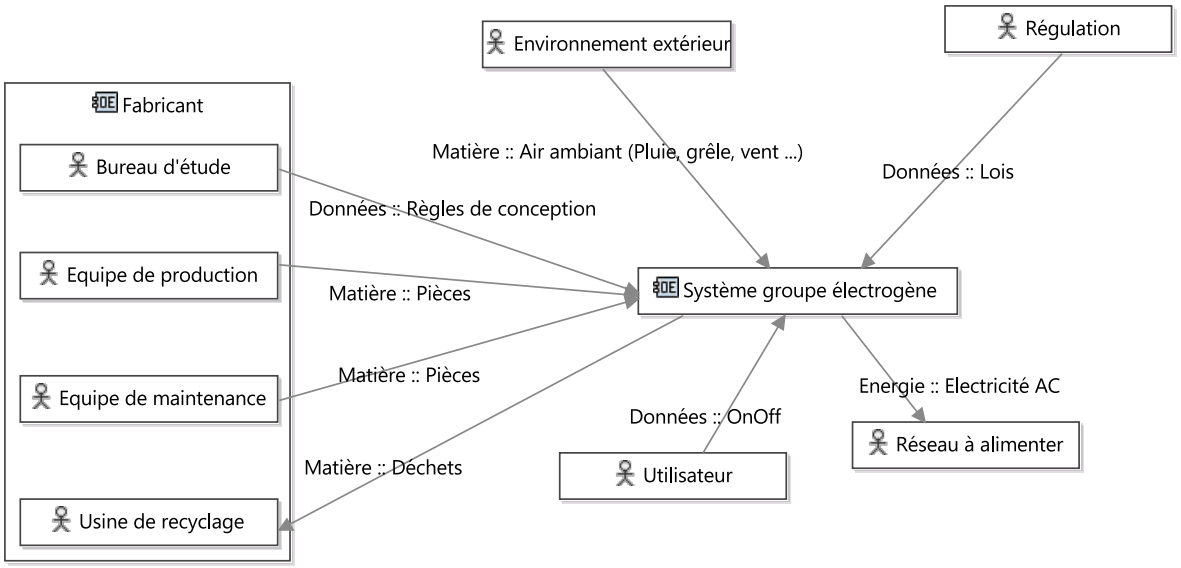


FIGURE 5.8 – Diagramme d'environnement du système à travers toutes les situations de vie.

On peut remarquer que plusieurs nouvelles parties prenantes ont été rajoutées. Ces parties prenantes sont impliquées dans de nouvelles phases de vie du système : la phase de conception, la phase de fabrication, la phase de maintenance et la phase de recyclage. Les transitions entre ces différentes situations de vie sont illustrées sur un cycle de vie présenté sur la Figure 5.9. Cela permet d'exprimer d'un point de vue opérationnel comment le système va évoluer entre ses différentes situations de vie.

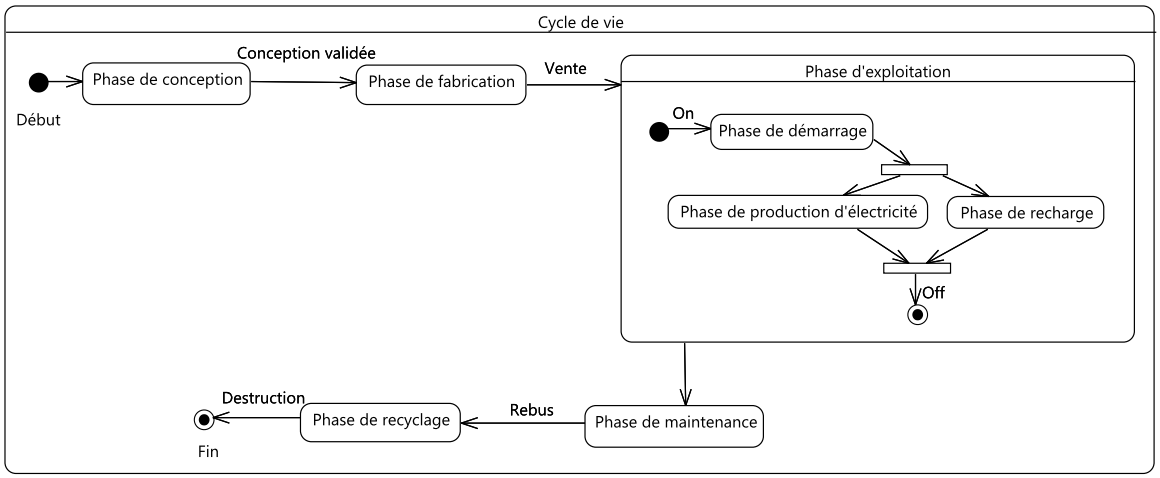


FIGURE 5.9 – Cycle de vie du système groupe électrogène.

Expression du besoin

Maintenant que les situations de vie et les parties prenantes pertinentes ont été déterminées, le besoin de chaque partie prenante dans chaque situation de vie doit être exprimé. Ce processus de capture du besoin est itératif. C'est-à-dire que lors de l'interview des parties prenantes pour capturer le besoin, de nouvelles situations de vie peuvent émerger avec de nouvelles parties prenantes qui leur sont associées. Pour cet exemple aucune interview n'a été menée, nous considérons que toutes les parties prenantes pertinentes ont été détectées.

Voici les listes complètes des parties prenantes et des situations de vie considérées :
Parties prenantes :

- Environnement extérieur
- Réseau à alimenter
- Utilisateur
- Régulation
- Bureau d'étude
- Équipe de production
- Équipe de maintenance
- Usine de recyclage

Situations de vie :

- Phase de démarrage
- Phase de recharge
- Phase de production électrique
- Phase de conception
- Phase de fabrication
- Phase de maintenance
- Phase de recyclage

Exprimons les besoins de chaque partie prenante avec le formalisme attendu. Commençons par l'environnement extérieur :

- L'environnement extérieur doit recevoir moins de 0.5 kg de CO_2 par heure en phase de démarrage.
- L'environnement extérieur doit recevoir moins de 0.5 kg de CO_2 par heure en phase de production électrique.
- L'environnement extérieur doit recevoir moins de 0.5 kg de CO_2 par heure en phase de recharge.

Exprimons maintenant les besoins du réseau à alimenter :

- Le réseau à alimenter doit recevoir un courant alternatif de 230V en phase de production électrique.
- Le réseau à alimenter doit recevoir un courant alternatif de 3kW pendant une heure en phase de production électrique.
- Le réseau électrique doit pouvoir se connecter au système à l'aide d'une connectique normalisée en phase de production électrique.

Exprimons maintenant les besoins de l'utilisateur :

- L'utilisateur doit pouvoir démarrer le système sans apport d'énergie en phase de démarrage.
- L'utilisateur doit pouvoir éteindre le système à la fin de la phase de production électrique.

Exprimons maintenant les besoins de l'équipe de production :

- L'équipe de production doit avoir moins de 15 sous-systèmes à intégrer pour construire le système en phase de fabrication.

— L'équipe de production doit transporter uniquement des sous-systèmes de moins de 20 kilogrammes en phase de fabrication.

Exprimons maintenant les besoins de l'équipe de maintenance :

— L'équipe de maintenance doit intervenir sur le système au plus toutes les 5000 heures de fonctionnement en phase de maintenance.

Exprimons maintenant les besoins de l'usine de recyclage :

— L'usine de recyclage doit pouvoir traiter l'ensemble du système en phase de recyclage.

Le bureau d'étude n'émet pas d'exigences vis-à-vis du produit à concevoir.

Une fois les besoins exprimés par les parties prenantes, ils doivent être transformés du point de vue du système. Il s'agit d'un passage du monde du client au monde du produit. Les exigences sont exprimées textuellement dans un premier temps :

TABLEAU 5.3 – Transformation des besoins en exigences pour l'exemple du groupe électrogène

Besoins	**Exigences**
B.1 L'environnement extérieur doit recevoir moins de 0.5 kg de CO_2 par heure en phase de démarrage.	E.1.1 Le système groupe électrogène doit émettre moins de 0.5 kg de CO_2 par heure en phase de démarrage.
B.2 L'environnement extérieur doit recevoir moins de 0.5 kg de CO_2 par heure en phase de production électrique.	E.2.1 Le système groupe électrogène doit émettre moins de 0.5 kg de CO_2 par heure en phase de production électrique.
B.3 L'environnement extérieur doit recevoir moins de 0.5 kg de CO_2 par heure en phase de recharge.	E.3.1 Le système groupe électrogène doit émettre moins de 0.5 kg de CO_2 par heure en phase de recharge.
B.4 Le réseau à alimenter doit recevoir un courant alternatif de 230 V en phase de production électrique.	E.4.1 Le système groupe électrogène doit émettre un courant alternatif de 230 V en phase de production électrique.
B.5 Le réseau à alimenter doit recevoir une puissance de 3 kW pendant une heure en phase de production électrique.	E.5.1 Le système groupe électrogène doit émettre un courant alternatif de 3 kW pendant une heure en phase de production électrique.
B.6 Le réseau électrique doit pouvoir se connecter au système à l'aide d'une connectique normalisée en phase de production électrique.	E.6.1 Le système groupe électrogène doit contenir une prise de courant normalisée en phase de production électrique.
B.7 L'utilisateur doit pouvoir démarrer le système sans apport d'énergie en phase de démarrage.	E.7.1 Le système groupe électrogène doit contenir une interface homme/machine pour démarrer le système en phase de démarrage. E.7.2 Le système groupe électrogène doit démarrer en 5 secondes en phase de démarrage.

continue sur la page suivante

continue depuis la page précédente	
Besoins	**Exigences**
B.8 L'utilisateur doit pouvoir éteindre le système à la fin de la phase de production électrique.	E.8.1 Le système groupe électrogène doit contenir une interface homme/machine pour éteindre le système en fin de phase de production électrique.
B.9 L'équipe de production doit avoir moins de 15 sous-systèmes à intégrer pour construire le système en phase de fabrication.	E.9.1 Le système groupe électrogène doit contenir moins de 15 sous systèmes en phase de fabrication.
B.10 L'équipe de production doit transporter uniquement des sous-systèmes de moins de 20 kilogrammes en phase de fabrication.	E.10.1 Le système groupe électrogène doit être composé uniquement de sous-systèmes de moins de 20 kg en phase de fabrication.
B.11 L'équipe de maintenance doit intervenir sur le système au plus toutes les 5000 heures de fonctionnement en phase d'exploitation.	E.11.1 Le système groupe électrogène doit avoir une occurrence de panne inférieure à une toutes les 5000 heures de fonctionnement durant sa phase d'exploitation.
B.12 L'usine de recyclage doit pouvoir traiter l'ensemble du système en phase de recyclage.	E.12.1 Le système groupe électrogène doit être composé uniquement de composant acceptés par l'usine de recyclage durant sa phase de conception.

Une fois que les exigences textuelles ont été écrites, il serait pertinent d'utiliser la méthode PESTEL introduite par AGUILAR [1967] pour vérifier que tous les domaines pouvant faire naître des exigences ont bien été pris en compte. Dans le cadre de cet exemple la méthode n'est pas appliquée, nous nous contenterons des exigences déjà exprimées.
Il est maintenant nécessaire de trier ces exigences dans les trois classes définies par la NASA [2007].

TABLEAU 5.4 – Tri des exigences pour l'exemple du groupe électrogène

	Exigence fonctionnelle	**Exigence de performance**	**Exigence de contrainte**
E.1.1 Le système groupe électrogène doit émettre moins de 0.5 kg de CO_2 par heure en phase de démarrage.		**X**	
E.2.1 Le système groupe électrogène doit émettre moins de 0.5 kg de CO_2 par heure en phase de production électrique.		**X**	
continue sur la page suivante			

continue depuis la page précédente

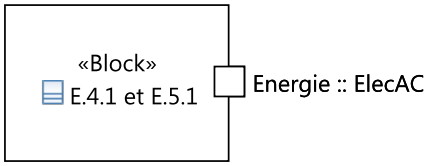
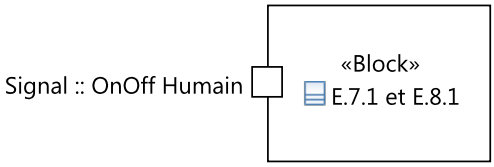
	Exigence fonctionnelle	Exigence de performance	Exigence de contrainte
E.3.1 Le système groupe électrogène doit émettre moins de 0.5 kg de CO_2 par heure en phase de recharge.		**X**	
E.4.1 Le système groupe électrogène doit émettre un courant alternatif de 230 V en phase de production électrique.	**X**	**X**	
E.5.1 Le système groupe électrogène doit émettre un courant alternatif de 3 kW pendant une heure en phase de production électrique.	**X**	**X**	
E.6.1 Le système groupe électrogène doit contenir une prise de courant normalisée en phase de production électrique.			**X**
E.7.1 Le système groupe électrogène doit contenir une interface homme/-machine pour démarrer le système en phase de démarrage.	**X**		**X**
E.7.2 Le système groupe électrogène doit démarrer en 5 secondes en phase de démarrage.		**X**	
E.8.1 Le système groupe électrogène doit contenir une interface homme/-machine pour éteindre le système en fin de phase de production électrique.	**X**		**X**
E.9.1 Le système groupe électrogène doit contenir moins de 15 sous systèmes en phase de fabrication.			**X**
E.10.1 Le système groupe électrogène doit être composé uniquement de sous-systèmes de moins de 20 kg en phase de fabrication.		**X**	

continue sur la page suivante

continue depuis la page précédente			
	Exigence fonctionnelle	**Exigence de performance**	**Exigence de contrainte**
E.11.1 Le système groupe électrogène doit avoir une occurrence de panne inférieure à une toutes les 5000 heures de fonctionnement durant sa phase d'exploitation.		**X**	
E.12.1 Le système groupe électrogène doit être composé uniquement de composant acceptés par l'usine de recyclage durant sa phase de conception.		**X**	

Toutes les exigences précédemment exprimées sont triées dans le but d'être ensuite transformées soit en boîtes noires fonctionnelles, soit en équations de contrainte qui seront utilisées par le modèle mathématique de synthèse architecturale.
Exprimons maintenant les exigences fonctionnelles sous forme de boîtes noires. Ce sont ces exigences qui vont s'appliquer sur les architectures logiques.

TABLEAU 5.5 – Transformation en boîtes noires pour l'exemple du groupe électrogène

Exigences fonctionnelles	**Boîtes noires**
E.4.1 et E.5.1 Le système groupe électrogène doit émettre un courant alternatif de 230 V en phase de production électrique. Et le système groupe électrogène doit émettre un courant alternatif de 3 kW pendant une heure en phase de production électrique.	
E.7.1 et E.8.1 Le système groupe électrogène doit contenir une interface homme/machine pour démarrer le système en phase de démarrage. Et le système groupe électrogène doit contenir une interface homme/machine pour éteindre le système en fin de phase de production électrique.	

Sont introduits deux types de port : Energie : : ElecAC et Signal : : OnOff. Le premier émet un flux de courant alternatif et le deuxième un flux de données pour le démarrage et l'arrêt

du système groupe électrogène.
Une fois que toutes les boîtes noires communiquant uniquement vers l'extérieur du système ont été définies elles peuvent être agrégées et il est aussi possible de rajouter des ports non-fonctionnels à la définition du système sous forme de boîte noire pour prendre en compte des interfaces avec l'environnement. Dans le cadre de cet exemple nous rajouterons un port air. Le port CO_2 n'a pas été rajouté, mais il aurait pu l'être surtout que des exigences portent sur l'émission de CO_2. Ces exigences ne seront pas traitées dans le reste de l'exemple. Au final nous obtenons une boîte noire du système groupe électrogène comme présentée sur la Figure 5.10. Le comportement du système groupe électrogène ne

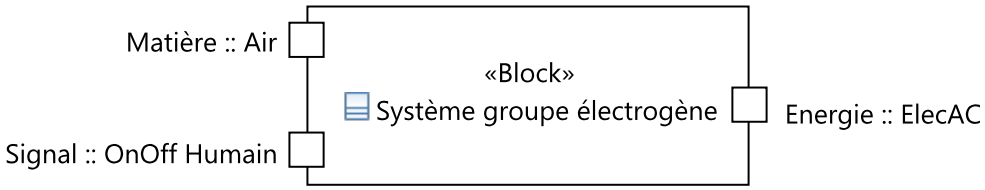


FIGURE 5.10 – Ports d'interface du système groupe électrogène.

va pas être le même durant les phases de vie de la phase d'exploitation. Il est nécessaire de définir maintenant les boîtes noires du système pour toutes les phases de vie qui seront prises en compte dans la synthèse des architectures. Ici deux phases sont considérées : la phase de démarrage et la phase de production électrique combinée à la phase de charge. La Figure 5.11 représente la boîte noire du système durant la phase de démarrage et la Figure 5.12 représente la boîte noire du système durant la phase de production électrique couplée à la phase de charge. Durant la phase de démarrage (Situation de vie 1), aucune

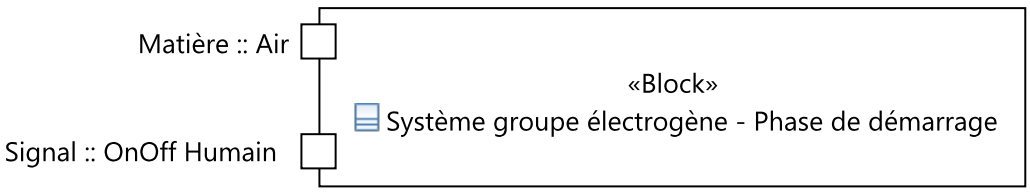


FIGURE 5.11 – Ports d'interface du système groupe électrogène durant la phase de démarrage (Situation de vie 1).

sortie de puissance électrique n'est attendue. Par contre, une commande provenant de l'utilisateur est requise. Durant la phase de production électrique (Situation de vie 2),

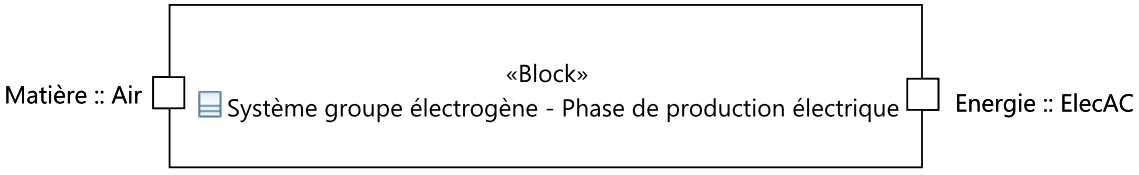


FIGURE 5.12 – Ports d'interface du système groupe électrogène durant la phase de production électrique (Situation de vie 2).

la puissance électrique est attendue en sortie, mais aucune commande n'est requise de l'utilisateur.

Il est temps de se pencher sur les autres types d'exigences et plus particulièrement sur les exigences de performance. Ces exigences vont s'appliquer sur les architectures physiques et pas sur les architectures logiques. Pour rappel les architectures physiques se focalisent sur l'instanciation des variables d'échange et des variables d'état du système. Il est donc nécessaire de définir quelles sont les exigences de performance qui vont s'appliquer sur des variables d'échange ou sur des variables d'état.

TABLEAU 5.6 – Allocation des exigences de performance pour l'exemple du groupe électrogène

	Variable d'échange	**Variable d'état**
E.1.1 Le système groupe électrogène doit émettre moins de 0.5 kg de CO_2 par heure en phase de démarrage.	**X**	
E.2.1 Le système groupe électrogène doit émettre moins de 0.5 kg de CO_2 par heure en phase de production électrique.	**X**	
E.3.1 Le système groupe électrogène doit émettre moins de 0.5 kg de CO_2 par heure en phase de recharge.	**X**	
E.4.1 Le système groupe électrogène doit émettre un courant alternatif de 230 V en phase de production électrique.	**X**	
E.5.1 Le système groupe électrogène doit émettre un courant alternatif de 3 kW pendant une heure en phase de production électrique.	**X**	
E.7.2 Le système groupe électrogène doit démarrer en 5 secondes en phase de démarrage.		**X**
E.10.1 Le système groupe électrogène doit être composé uniquement de sous-systèmes de moins de 20 kg en phase de fabrication.		**X**
E.11.1 Le système groupe électrogène doit avoir une occurrence de panne inférieure à une toutes les 5000 heures de fonctionnement durant sa phase d'exploitation.		**X**
E.12.1 Le système groupe électrogène doit être composé uniquement de composant acceptés par l'usine de recyclage durant sa phase de conception.		**X**

Au niveau système nous définissons quatre variables : une variable d'échange et trois variables d'état. C'est sur ces variables que sont appliquées les exigences de performance. Comme vu précédemment, le système est défini par ses ports, mais aussi par des va-

riables d'échange, contenues dans les ports, et des variables d'état. Les exigences de performance sont exprimées sous la forme de contraintes sur les variables d'échange pour des performances fonctionnelles et sur des variables d'état pour des performances non-fonctionnelles. Les variables d'échange du système groupe électrogène sont présentées dans le Tableau 5.7. Les variables d'état du système groupe électrogène sont présentées dans le Tableau 5.8. A noter que les ports «Matière : : Air» et «Signal : : OnOff» ne contiennent aucune variable dans cette modélisation.

TABLEAU 5.7 – Définition physique des ports du système à concevoir.

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque situation de vie		Est associative ?
Port Énergie : : ElecAC						
WElecAC	kW	0	20	S. de vie 1 S. de vie 2	= 0 >= 3	Oui

Les variables définies dans le tableau 5.7, sont explicitées maintenant :

— WElecAC : la puissance électrique émise lors de la phase de production électrique.

TABLEAU 5.8 – Définition physique des variables d'état du système drone.

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque situation de vie		Est associative ?	Impose ou collecte
Système groupe électrogène							
t	s	0	3600	S. de vie 1 S. de vie 2	= 5 = 3600	Oui	Impose
M	kg	0	50	S. de vie 1 S. de vie 2	∅ ∅	Oui	Collecte
Jconso	J	0	100000	S. de vie 1 S. de vie 2	∅ ∅	Oui	Collecte

Les variables définies dans le tableau 5.8, sont explicitées maintenant :

— t : le temps de fonctionnement, 5 secondes pour la phase de démarrage et 1h00 pour la phase de production électrique,

— M : la masse totale du groupe électrogène,

— Jconso : la consommation totale en carburant du groupe électrogène.

Il n'y a pas d'équations comportementales propres au système. C'est-à-dire que tout son comportement est uniquement décrit par les contraintes qui ont été appliquées sur les variables dans le tableau 5.8.

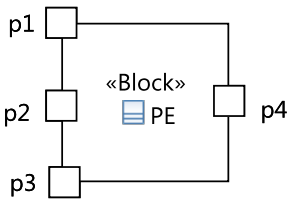
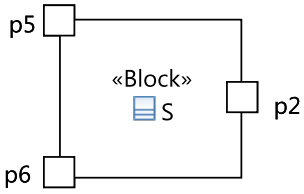
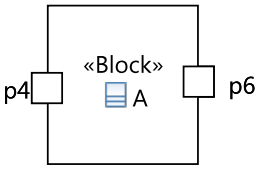
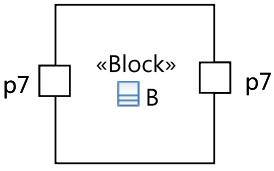
Les exigences de contrainte doivent porter sur le fait de forcer ou non les occurrences de certains AO. Ici le système doit contenir une prise pour se connecter au réseau à alimenter.

Ce sous-système ne sera pas modélisé car il doit forcément faire partie du système et il n'a pas un gros impact sur le reste de l'architecture. Le système va contenir un sous-système «Interface Homme Machine». Le système aura moins de 15 sous-systèmes. En plus de ces contraintes venant directement des exigences, nous rajoutons une contrainte pour augmenter l'espace des solutions : nous autorisons les architectures à avoir jusqu'à deux moteurs thermiques. En effet la synthèse assistée par ordinateur que nous avons développée permet de sortir, de manière exhaustive, toutes les architectures minimalistes pour répondre au besoin. Dans le cas de l'exemple, un seul moteur thermique aurait été nécessaire, mais comme nous avons forcé l'occurrence de l'AO moteur thermique entre un et deux, nous aurons donc plus de solutions.

Construction de la base de données des AO

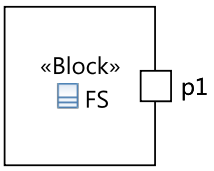
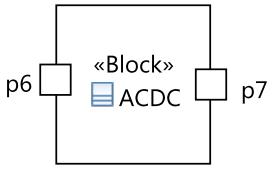
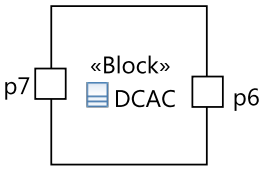
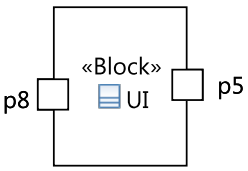
Maintenant que nous avons défini le système à concevoir vu de l'extérieur, il est temps de construire la base de données des sous-systèmes qui va permettre de construire l'intérieur du système. Le tableau 5.9 présente les AO qui sont disponibles pour créer le système. Seule la vue logique est affichée pour le moment.

TABLEAU 5.9 – Base de données des AO pour l'exemple du système groupe électrogène

Description	AO
PE pour Piston Engine ou moteur à pistons en français est la modélisation simplifiée d'un moteur thermique. Cet AO est composé de trois ports d'entrée et un port de sortie. p1 correspond au port Matière : : Carburant. p2 correspond au port Energie : : Mécanique rotation commande. p3 correspond au port Matière : : Air. p4 correspond au port Energie : : Mécanique rotation production.	
S pour Starter ou démarreur en français est la modélisation simplifiée d'un démarreur. Cet AO est composé de deux ports d'entrée et d'un port de sortie. p5 correspond au port Signal : : OnOff machine. p6 correspond au port Energie : : ElecAC. p2 correspond au port Energie : : Mécanique rotation commande.	
A pour Alternator ou générateur en français est la modélisation simplifiée d'un générateur. Cet AO est composé d'un port d'entrée et d'un port de sortie. p4 correspond au port Energie : : Mécanique rotation production. p6 correspond au port Energie : : ElecAC.	
B pour Battery ou encore batterie en français est la modélisation simplifiée d'une batterie. Cet AO est composé d'un port d'entrée et d'un port de sortie. p7 correspond au port Energie : : ElecDC.	

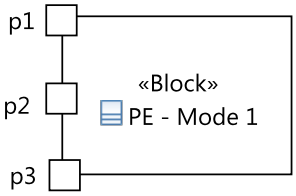
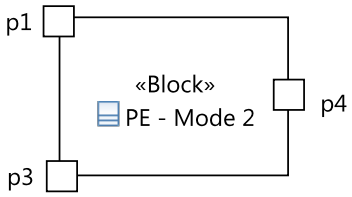
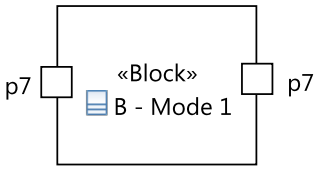
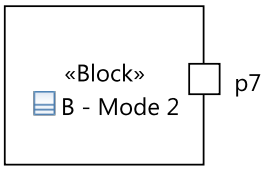
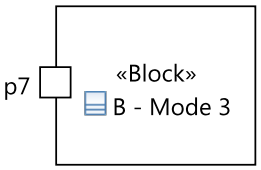
continue sur la page suivante

continue depuis la page précédente

Description	**AO**
FS pour Fuel System ou encore système carburant en français est la modélisation simplifiée d'un système carburant. Cet AO est composé d'un port de sortie. p1 correspond au port Matière : : Carburant.	
ACDC pour ACDC converter ou encore convertisseur ACDC en français est la modélisation simplifiée d'un convertisseur ACDC. Cet AO est composé d'un port d'entrée et d'un port de sortie. p6 correspond au port Energie : : ElecAC. p7 correspond au port Energie : : ElecDC.	
DCAC pour DCAC converter ou encore convertisseur DCAC en français est la modélisation simplifiée d'un convertisseur DCAC. Cet AO est composé d'un port d'entrée et d'un port de sortie. p7 correspond au port Energie : : ElecDC. p6 correspond au port Energie : : ElecAC.	
UI pour User Interface ou encore interface Homme machine en français est la modélisation simplifiée d'une interface Homme machine. Cet AO est composé d'un port d'entrée et d'un port de sortie. p8 correspond au port Signal : : OnOff Humain. p5 correspond au port Signal : : OnOff machine.	

Tous ces AO ont été présentés avec une vue globale, c'est-à-dire en comptabilisant tous les ports, qu'ils soient actifs ou non. Maintenant nous allons définir les différents modes de fonctionnement des AO en choisissant quels sont les ports actifs ou non dans chaque mode de fonctionnement pour chaque AO. Le résultat de cette activité est présenté dans le tableau 5.10. Le mode arrêt complet avec aucun port actif n'est pas présenté dans le tableau, mais il existe bien. Les AO qui n'ont que deux modes : tous les ports actifs et tous les ports inactifs ne sont pas présentés dans le tableau.

TABLEAU 5.10 – Les différents AO avec leurs modes de fonctionnement respectifs

Description	AO
Le premier mode du moteur thermique est son mode de démarrage. Il reçoit de l'air, du carburant et une énergie mécanique de rotation pour démarrer.	
Le second mode du moteur thermique est son mode nominal. Il produit une énergie mécanique de rotation avec de l'air et du carburant.	
Le premier mode de la batterie est un mode de recharge et de débit en simultané. Elle reçoit une puissance électrique et débite une autre puissance électrique.	
Le deuxième mode de la batterie est un mode de débit pur. Elle débit une puissance électrique.	
Le troisième mode de la batterie est un mode de charge pure. Elle reçoit une puissance électrique.	

A ce stade la base de données des AO a été entièrement définie d'un point de vue logique. Il reste maintenant à décrire sa définition physique. L'objectif est de définir pour chaque AO et pour chacun de ses ports ses variables d'échange, ses variables d'état et ses équations comportementales. Le tableau 5.11 présente toutes les variables d'échange des ports des AO de la base de données.

TABLEAU 5.11 – Définition physique des ports des AO de la base de données pour l'exemple du système groupe électrogène.

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque mode	Est associative ?
AO Moteur thermique (PE)					
continue sur la page suivante					

continue depuis la page précédente

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque mode		Est associative ?
Port Matière : : Carburant (p1)						
Qc	kg/h	0	10	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui
Port Energie : : Mécanique rotation commande (p2)						
$Wmeca_2$	kW	0	20	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui
Port Energie : : Mécanique rotation production (p4)						
$Wmeca_4$	kW	0	20	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui
AO Démarreur (S)						
Port Energie : : ElecAC (p6)						
$WelecAC_6$	kW	0	20	Mode 1 Mode 2	∅ ∅	Oui
Port Energie : : Mécanique rotation commande (p2)						
$Wmeca_7$	kW	0	20	Mode 1 Mode 2	∅ ∅	Oui
AO Générateur (A)						
Port Énergie : : Mécanique rotation production (p4)						
$Wmeca_8$	kW	0	20	Mode 1 Mode 2	∅ ∅	Oui
Port Énergie : : ElecAC (p6)						
$WelecAC_9$	kW	0	20	Mode 1 Mode 2	∅ ∅	Oui
AO Batterie (B)						
Port Énergie : : ElecDC (p6)						
$WelecDCi_{10}$	kW	0	20	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui
Port Énergie : : ElecDC (p6)						
$WelecDCo_{11}$	kW	0	20	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui
AO Système carburant (FS)						
Port Matière : : Carburant (p1)						
Qc_{12}	kg/h	0	10	Mode 1 Mode 2	∅ ∅	Oui
AO Convertisseur ACDC (ACDC)						
Port Énergie : : ElecAC (p6)						
$WelecAC_{13}$	kW	0	20	Mode 1 Mode 2	∅ ∅	Oui

continue sur la page suivante

continue depuis la page précédente

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque mode		Est associative ?
Port Énergie : : ElecDC (p7)						
$WelecDC_{14}$	kW	0	20	Mode 1 Mode 2	∅ ∅	Oui
AO Convertisseur DCAC (DCAC)						
Port Énergie : : ElecDC (p7)						
$WelecDC_{15}$	kW	0	20	Mode 1 Mode 2	∅ ∅	Oui
Port Énergie : : ElecAC (p6)						
$WelecAC_{16}$	kW	0	20	Mode 1 Mode 2	∅ ∅	Oui

Les AO et les ports qui n'apparaissent pas dans le tableau ne sont pas modélisés dans la vue physique d'un point de vue des variables d'échange.
Les variables définies dans le tableau 5.11, sont explicitées maintenant :

— Q_c : le débit de carburant en entrée du moteur thermique et en sortie du système carburant,

— Wmeca : les puissances mécaniques de rotation,

— WelecAC : les puissances électriques des courants alternatifs,

— WelecDC : les puissances électriques des courants continus.

Le tableau 5.12 présente toutes les variables d'état des AO de la base de données.

TABLEAU 5.12 – Définition physique des variables d'état des AO de la base de données pour l'exemple du groupe électrogène.

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque mode		Est associative ?	Somme, max ou NA
AO Moteur thermique (PE)							
Wc	kW	0	50	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui	Somme
n	—	0	1	Mode 1 Mode 2 Mode 3	= 0.45 = 0.45 = 0.45	—	—
t	s	0	3600	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	—	Imposée
Jconso	kJ	0	100000	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui	Somme

continue sur la page suivante

continue depuis la page précédente

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque mode	Est associative ?	Somme, max, imposée ou NA
PCI	kJ/kg	0	47300	Mode 1 = 47300 Mode 2 = 47300 Mode 3 = 47300	—	—
M	kg	4	50	Mode 1 ∅ Mode 2 ∅ Mode 3 ∅	Oui	Max
AO Démarreur (S)						
n	—	0	1	Mode 1 = 0.9 Mode 2 = 0.9	—	—
M	kg	0.2	50	Mode 1 ∅ Mode 2 ∅	Oui	—
AO Générateur (A)						
n	—	0	1	Mode 1 = 0.9 Mode 2 = 0.9	—	—
M	kg	0.2	50	Mode 1 ∅ Mode 2 ∅	Oui	—
AO Batterie (B)						
t	s	0	3600	Mode 1 ∅ Mode 2 ∅ Mode 3 ∅	—	Imposée
J_1	kJ	-100000	100000	Mode 1 >= 0 Mode 2 >= 0 Mode 3 >= 0	Oui	—
J_2	kJ	-100000	100000	Mode 1 <= 0 Mode 2 <= 0 Mode 3 <= 0	Oui	—
J_3	kJ	-100000	100000	Mode 1 >= 0 Mode 2 >= 0 Mode 3 >= 0	Oui	—
M	kg	0	50	Mode 1 ∅ Mode 2 ∅ Mode 3 ∅	Oui	Somme
AO Système carburant (FS)						
t	s	0	3600	Mode 1 ∅ Mode 2 ∅	—	Imposée
M	kg	0	50	Mode 1 ∅ Mode 2 ∅	Oui	Somme
AO Convertisseur ACDC (ACDC)						
n	—	0	1	Mode 1 = 0.9 Mode 2 = 0.9	—	—

continue sur la page suivante

continue depuis la page précédente

Variable	Unité	Borne min	Borne max	Contraintes associées à chaque mode		Est associative ?	Somme, max, imposée ou NA
M	kg	0.2	50	Mode 1 Mode 2	∅ ∅	Oui	Max
AO Convertisseur DCAC (DCAC)							
n	—	0	1	Mode 1 Mode 2	= 0.9 = 0.9	—	—
M	kg	0.2	50	Mode 1 Mode 2	∅ ∅	Oui	Max
AO Interface Homme machine (UI)							
M	kg	0	50	Mode 1 Mode 2	∅ ∅	Oui	Max

Les variables définies dans le tableau 5.12, sont explicitées maintenant :

- W_c : les puissances de carburant transmises,
- n : les rendements des AO,
- t : le temps,
- Jconso : les énergies de carburant qui ont été consommées,
- PCI : le pouvoir calorifique inférieur du carburant,
- M : les masses des AO,
- Jconso : les énergies électriques qui ont été consommées ou produites.

Les équations comportementales des AO sont données ci-après par mode de fonctionnement :

Moteur thermique (PE)

- **Mode 1 -Démarrage-** :

$$\frac{W_c \times 3600}{Q_c \times PCI} - 1 = 0$$

L'équation normalisée liant la puissance carburant au débit de carburant.

$$\frac{n \times W_c}{Wmeca_2} - 1 = 0$$

L'équation normalisée de rendement du moteur entre la puissance carburant et la puissance fournie au démarrage.

$$\frac{Jconso}{W_c \times t} - 1 = 0$$

L'équation normalisée permettant le calcul de l'énergie de carburant consommée durant la phase de démarrage.

— **Mode 2 -Fonctionnement nominal-** :

$$\frac{\mathrm{W}_c \times 3600}{\mathrm{Q}_c \times \mathrm{PCI}} - 1 = 0$$

L'équation normalisée liant la puissance carburant au débit de carburant.

$$\frac{n \times \mathrm{W}_c}{\mathrm{W}meca_4} - 1 = 0$$

L'équation normalisée de rendement du moteur entre la puissance carburant et la puissance fournie dans le mode nominal.

$$\frac{\mathrm{J}conso}{\mathrm{W}_c \times t} - 1 = 0$$

L'équation normalisée permettant le calcul de l'énergie de carburant consommée durant la phase de démarrage.

$$\frac{\mathrm{M} + 3.3563}{3.9076 \times \mathrm{W}meca_4} - 1 = 0$$

L'équation normalisée de masse du moteur qui provient d'une étude sur les moteurs thermiques existant.

— **Mode 2 -A l'arrêt-** :

$$\emptyset$$

Pas d'équations comportementales, car le moteur est à l'arrêt.

Démarreur (S)

— **Mode 1 -Fonctionnement nominal-** :

$$\frac{n \times \mathrm{W}elec\mathrm{AC}_6}{\mathrm{W}meca_7} - 1 = 0$$

L'équation normalisée de rendement du démarreur.

$$\frac{\mathrm{M} - 0.72}{2.4 \times \mathrm{W}meca_7} - 1 = 0$$

L'équation normalisée de masse du démarreur qui provient d'une étude sur les moteurs électriques existants.

— **Mode 2 -A l'arrêt-** :

$$\emptyset$$

Pas d'équations comportementales, car le moteur est à l'arrêt.

Générateur (A)

— **Mode 1 -Fonctionnement nominal-** :

$$\frac{n \times \mathrm{W}elec\mathrm{AC}_8}{\mathrm{W}meca_9} - 1 = 0$$

L'équation normalisée de rendement du générateur.

$$\frac{\mathrm{M} - 0.72}{2.4 \times \mathrm{W}meca_7} - 1 = 0$$

L'équation normalisée de masse du générateur qui provient d'une étude sur les générateurs existants.

— **Mode 2 -A l'arrêt-** :

$$\emptyset$$

Pas d'équations comportementales, car le moteur est à l'arrêt.

Batterie (B)

— **Mode 1 -Débit et stockage-** :

$$\frac{\mathrm{J}_1}{(\mathrm{W}elec\mathrm{DC}i_1 0 - \mathrm{W}elec\mathrm{DC}o_1 1) * t} - 1 = 0$$

L'équation normalisée de l'énergie électrique stockée dans la batterie.

$$\frac{\mathrm{M}}{0.0057 \times \mathrm{J}_1} - 1 = 0$$

L'équation normalisée de masse de la batterie qui provient d'une étude sur les batteries existantes.

— **Mode 2 -Débit-** :

$$\frac{\mathrm{J}_2}{-\mathrm{W}elec\mathrm{DC}o_1 1 * t} - 1 = 0$$

L'équation normalisée de l'énergie électrique stockée dans la batterie.

$$(\mathrm{J}_3/\mathrm{J}_2) + 1 >= 0$$

L'équation normalisée exprimant le fait que la batterie doit permettre le démarrage du système sans énergie extérieure.

— **Mode 3 -Stockage-** :

$$\frac{\mathrm{J}_3}{-\mathrm{W}elec\mathrm{DC}i_1 0 * t} - 1 = 0$$

L'équation normalisée de l'énergie électrique stockée dans la batterie.

$$\frac{\mathrm{M}}{0.0057 \times \mathrm{J}_3} - 1 = 0$$

L'équation normalisée de masse de la batterie qui provient d'une étude sur les batteries existantes.

— **Mode 4 -Arrêt-** :

$$\emptyset$$

Pas d'équations comportementales, car le moteur est à l'arrêt.

Système carburant (FS)

— **Mode 1 -Débit-** :

$$\frac{\mathrm{M} \times 3600}{\mathrm{Q}c_1 2 \times t} - 1 = 0$$

L'équation normalisée de la masse de carburant utilisé.

— **Mode 2 -A l'arrêt-** :

$$\emptyset$$

Pas d'équations comportementales, car le moteur est à l'arrêt.

Convertisseur ACDC (ACDC)

— **Mode 1 -Fonctionnement nominal-** :

$$\frac{n \times \mathrm{W}elec\mathrm{AC}_1 3}{\mathrm{W}elec\mathrm{DC}_1 4} - 1 = 0$$

L'équation normalisée de rendement du convertisseur ACDC.

$$\frac{\mathrm{M} - 1.0458 * \mathrm{W}elec\mathrm{AC}_1 3}{0.6625} - 1 = 0$$

L'équation normalisée de masse du convertisseur ACDC qui provient d'une étude sur les convertisseurs existants.

— **Mode 2 -A l'arrêt-** :

$$\emptyset$$

Pas d'équations comportementales, car le moteur est à l'arrêt.

Interface Homme machine (UI)

— **Mode 1 -Démarrage-** :

$$\mathrm{M} - 1 = 0$$

L'équation normalisée de masse de l'interface Homme machine.

— **Mode 2 -A l'arrêt-** :

$$\emptyset$$

Pas d'équations comportementales, car l'interface Homme machine n'est pas utilisée.

A ce stade la base de données des AO est totalement définie. Les AO ont été définis par leurs vues logique et physique.
Le système à concevoir a aussi été défini complètement d'un point de vue logique et d'un point de vue physique.
Une contrainte a été rajoutée pour générer des solutions avec un et deux moteurs thermiques.
La synthèse automatique peut être lancée.

La synthèse des architectures logiques

Dans un premier temps, la synthèse automatique génère trois méta-architectures comme présenté sur la Figure 5.13. La méta-architecture A débite directement d'un alternateur alors que la solution B débite de la batterie à travers un convertisseur DCAC. La troisième méta-architecture est un mélange des deux premières.

Dans un second temps, les occurrences de chaque AO sont calculées et une base de données étendue est créée. Elle contient deux moteurs thermiques, deux démarreurs et

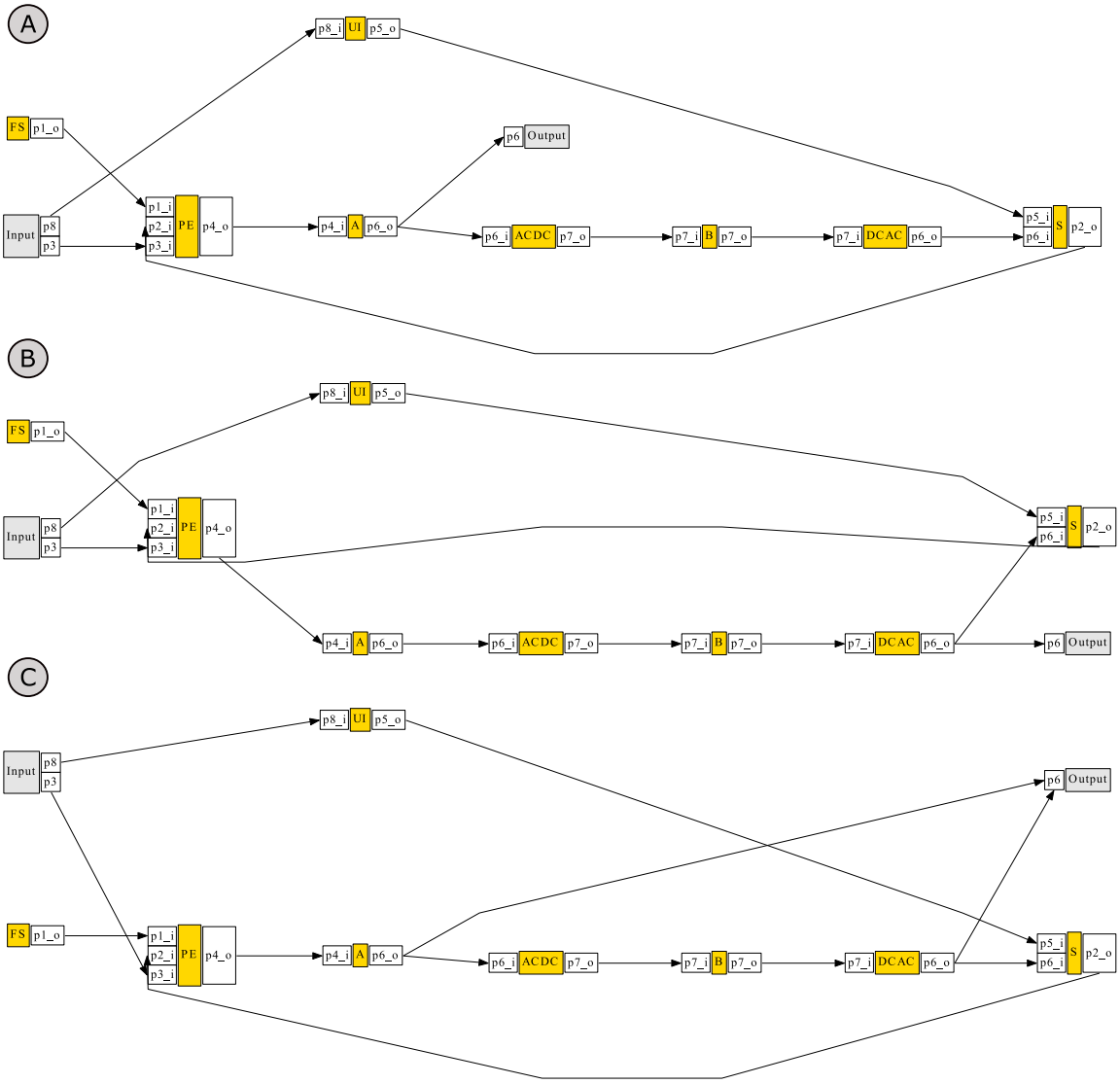


FIGURE 5.13 – Représentation des trois meta-architectures générées (A, B et C).

deux générateurs pour répondre à la contrainte de pouvoir générer des architectures avec deux moteurs thermiques.

Dans un troisième temps la synthèse des architectures logiques et la détection des isomorphismes produit dix solutions toutes différentes présentées sur la Figure 5.14. Les

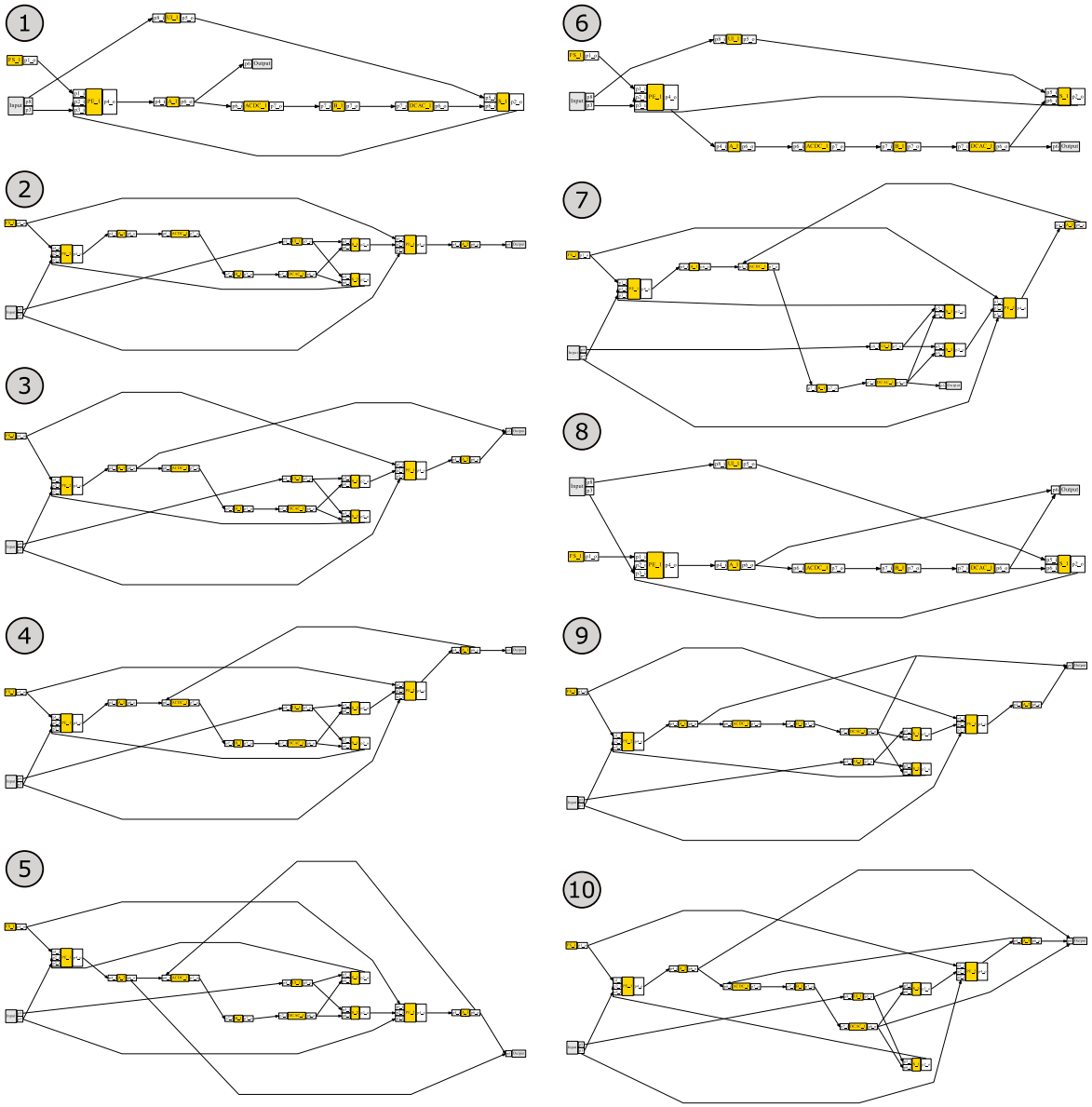


FIGURE 5.14 – Toutes les architectures logiques générées

architectures logiques proviennent des méta-architectures suivant la répartition suivante :

— A : 1, 2, 3, 4, 5
— B : 6, 7
— C : 8, 9, 10

Pour une analyse plus en détail de ces architectures logiques, merci de vous reporter aux travaux de HARTMANN et collab. [2018].

L'analyse physique et la génération des architectures physiques

Dans le cadre de cet exemple, la fonction objectif à minimiser est la masse du système. Chaque architecture logique est instanciée tout en essayant de minimiser la masse du

système groupe électrogène. Seulement neuf architectures sur les dix générées lors de la synthèse des architectures logiques ont pu être instanciées. Les résultats partiels de l'analyse physique sur les variables M et WE*lec*AC du système groupe électrogène sont donnés sur la Figure 5.15. L'architecte avec ces résultats va maintenant pouvoir écarter les

Architecture	Total weight (kg)	Power (kW)
1	22.6	3.0
3	23.5	3.4
4	23.5	3.4
5	23.5	3.4
6	34.9	3.0
7	33.0	3.0
8	22.6	3.3
9	23.6	3.4
10	23.6	3.4

FIGURE 5.15 – Résultat partiel de l'analyse physique sur les variables M et WE*lec*AC du système groupe électrogène.

architectures 6 et 7 qui sont les plus lourdes et se focaliser sur une étude plus détaillée des sept architectures restantes.

5.8 Conclusion du chapitre

A la fin du processus de synthèse, toutes les architectures qui n'étaient pas viables physiquement, mais uniquement d'un point de vue de la logique, ont été rejetées. A ce moment, toutes les solutions architecturales ont été prédimensionnées, c'est-à-dire que leurs paramètres clés, comme la masse et la puissance des différents sous-sytèmes, ont été déterminés. L'utilisation des CSP sur des variables entières représentatives de l'architecture physique permettent d'instancier les architectures logiques. Même si des limitations sont pour l'instant associées à cette méthode de résolution, pour des équations simples, il est possible d'obtenir des résultats. L'utilisation de CSP permet donc d'apporter une solution à la question de recherche **Question 4**.
Si lors de la synthèse des architectures physiques une méthode d'optimisation est utilisée plutôt qu'une simple méthode de recherche de toutes les solutions, alors on peut faire une sélection sur un critère unique des concepts architecturaux. Il est possible d'instancier toutes les architectures logiques tout en les optimisant. Le tri de ces solutions instanciées permet alors de sélectionner celles qui sont les plus pertinentes. Cela répond donc à la question de recherche **Question 5**.
Une étude plus détaillée sur certaines architectures prometteuses doit être menée comme cela est fait habituellement lors d'une phase d'avant-projet. Néanmoins, l'apport de la méthodologie assistée par ordinateur est le nombre beaucoup plus important d'alternatives étudiées, et le fait que celles qui sont ensuite sélectionnées présentent une plus grande légitimité. La condition évidente est que les modèles utilisés pour les AO aient un sens.

Chapitre 6

Conclusion

Sommaire

6.1 Implémentation de la synthèse automatisée

L'ensemble du cadre de synthèse est outillé par un logiciel développé pour l'occasion. Dans cette section, sont présentées les actions que doit réaliser l'architecte pour utiliser le logiciel factuellement, sans faire aucune allusion aux modèles ou au cadre de synthèse. Les interactions entre l'utilisateur et le logiciel se font principalement avec des fichiers Excel. Le transfert des données et l'enchaînement des actions de la synthèse sont illustrés sur les Figures 6.1 et 6.2. Sur ces deux figures, sont représentés les différents éléments du logiciel, l'utilisateur, l'enchaînement de certaines actions et les échanges de données. Ces échanges de données sont d'ailleurs numérotés et sont explicités ci-après.

(1) représente le remplissage de la base de données des AO et la définition du système à concevoir d'un point de vue logique. Ici, aucune définition physique à l'aide d'équations n'est demandée.

(2) représente le remplissage de la base de données des AO et la définition du système à concevoir d'un point de vue physique. Les différentes variables du modèle, ainsi que les équations comportementales de chaque AO et du système attendu sont renseignées.

(3) représente la définition, par l'utilisateur, des différents modes de fonctionnement des AO de la base de données, ainsi que les différentes situations de vie du système à concevoir.

(4) représente les occurrences imposées par l'utilisateur à certains AO.

(5), (6) et (7) sont des fichiers au format txt, qui servent à remplir les définitions des objets C++. Chaque concept, chaque modèle présenté dans ce manuscrit dispose d'une définition similaire sous la forme d'une classe C++. A titre d'illustration, un exemple d'une classe est donné sur le code source 6.1. Ce dernier correspond à la définition des attributs de la classe «Port».

(8) représente un code data Minizinc correspondant à la base de données précédemment définie et à la définition du système à concevoir. Le fichier modèle Minizinc a été écrit une seule fois. C'est lui qui contient les contraintes qui vont permettre la synthèse.

(9) représente un fichier txt contenant toutes les méta-architectures.

(10) représente des fichiers modèles et data Minizinc contenant les résultats de la synthèse des méta-architectures et les occurrences des AO définies par l'utilisateur.

(11) représente un fichier txt résultat contenant les occurrences nécessaires pour chaque AO, dans le but de permettre le fonctionnement de chaque méta-architecture.

(12) représente un code data Minizinc correspondant à la base de données précédemment définie et à la définition du système à concevoir. Le fichier modèle Minizinc est, cette fois-ci, aussi généré à la volée car il contient des contraintes fonctions des résultats précédents. Il s'agit des contraintes de parité sur la cardinalité de certaines connexions.

(13) représente un fichier txt résultat contenant les architectures logiques.

(14) représente des fichiers bch, qui sont les architectures physiques, produits par un code C++ et envoyés à Ibex pour une résolution du système d'équations.

(15) représente des fichiers txt pour chaque architecture logique avec toutes les variables calculées.

(16) représente un affichage à l'utilisateur de toutes les solutions architecturales triées par rapport à une variable à minimiser, renseignée par l'utilisateur.

```
1 class Port
```

```
{
   //Les variables (attributs) sont prives.
      protected:
   //Le type du port
      std::string m_type;
   //La direction du port
      std::string m_direction;
   //Le composant auquel le port appartient
      std::string m_component;
   //Est-ce que le port est obligatoirement connecte ou non ?
      std::string m_obligation;
   //La multiplicite minimale du port
      std::string m_nbMin;
   //La multiplicite maximale du port
      std::string m_nbMax;
   //Est-ce que la cardinalite du port doit etre paire ou non ?
      std::string m_nbPair;
   //Les variables contenues dans le port
      std::vector<std::string> m_variables;
   //Les contraintes associees aux variables dans les differents modes de ↩
       fonctionnement de son composant
      std::vector<std::vector<std::vector<std::string>>> m_constraints;
   //Savoir si la variable est associative ou non
      std::vector<std::string> m_associations;
   //La borne min de la variable
      std::vector<std::string> m_min_var;
   //La borne max de la variable
      std::vector<std::string> m_max_var;
};
```

Code 6.1 – La classe «port» en C++

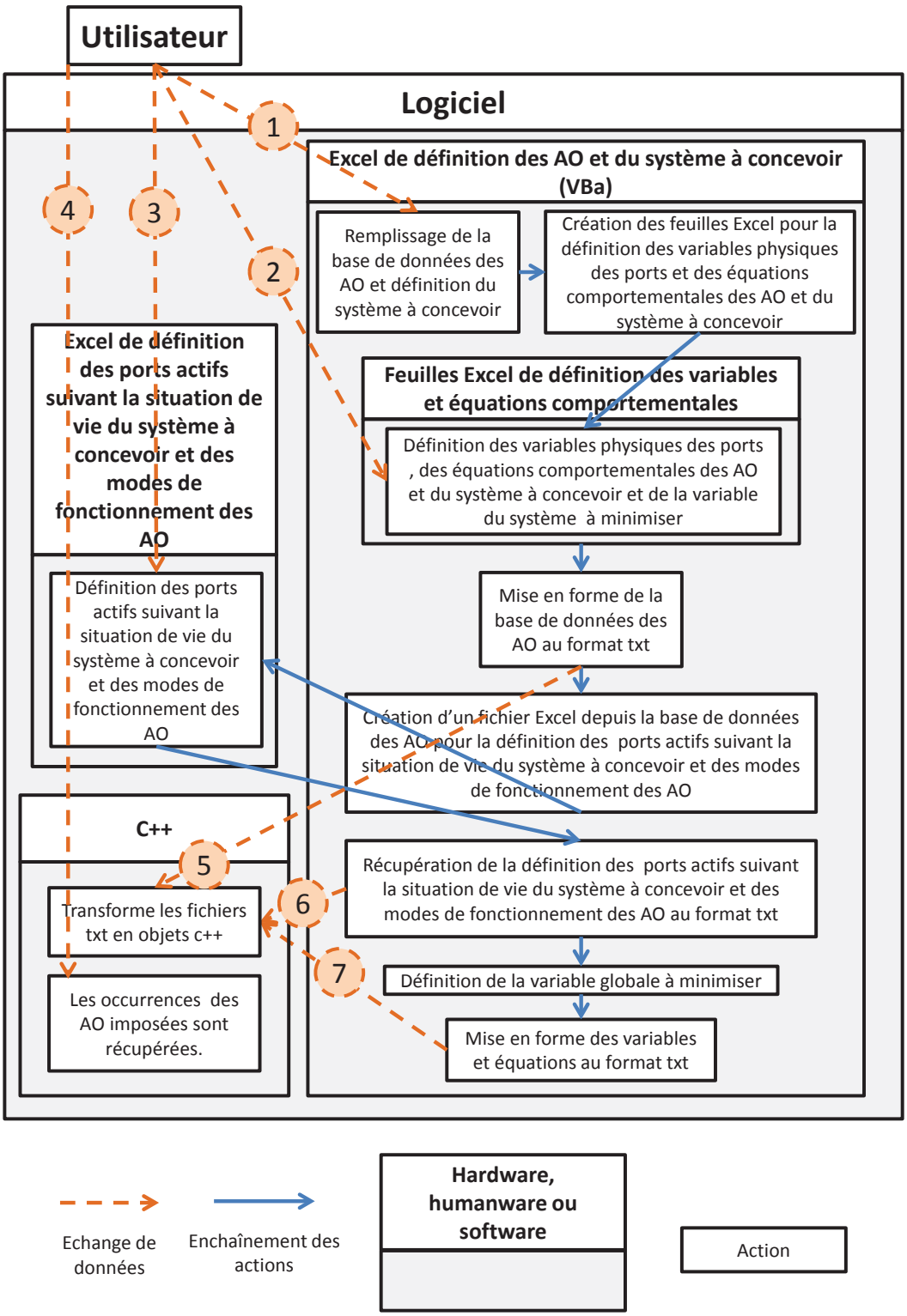


FIGURE 6.1 – La méthodologie outillée (1/2)

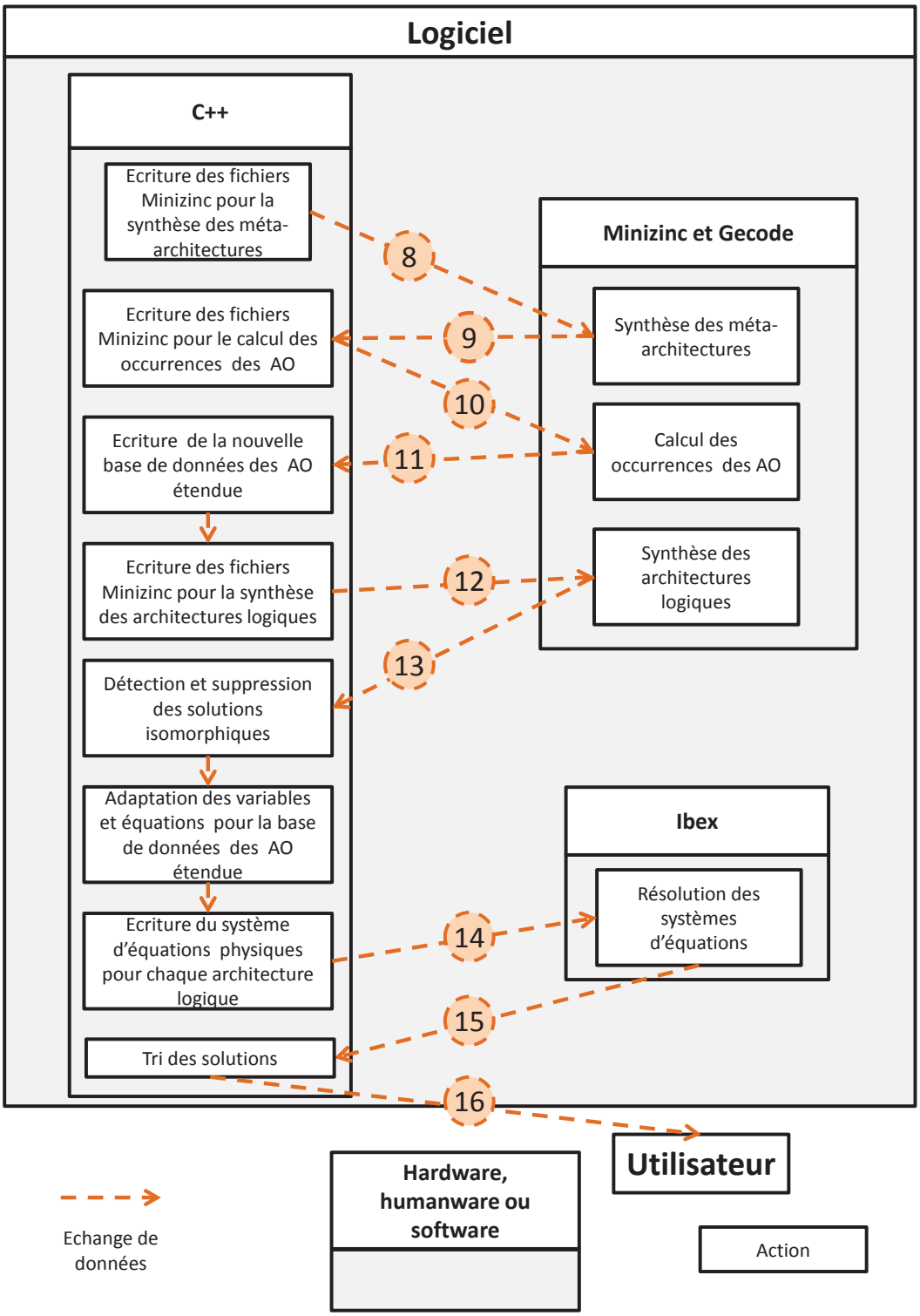


FIGURE 6.2 – La méthodologie outillée (2/2)

6.2 Apport scientifique

Tout au long de ce manuscrit, une méthode semi-automatisée de synthèse architecturale a été présentée. Cette méthode reprend les concepts de l'ingénierie système et permet la transformation automatique du problème de conception en un ensemble de CSP qui seront résolus à l'aide de solveurs discret et continu. Jusqu'à présent, peu de travaux ont traité le problème d'une façon similaire à celle présentée ici. Les travaux les plus proches sont ceux de MÜNZER et collab. [2013] et de HELMS et SHEA [2012], qui reposent, pour l'un, sur l'utilisation de solveurs SAT et, pour l'autre, sur une grammaire architecturale permettant la création des solutions à l'aide de règles.

L'utilisation des matrices d'autorisation et d'obligation, couplée à l'usage de modèles CSP, est une nouveauté dans le champ bibliographique de la synthèse. De même, le découpage de la résolution logique en trois phases est aussi nouveau avec, en premier lieu, la génération de méta-architectures, puis seulement ensuite le calcul du nombre d'occurrences nécessaires pour chaque AO, et enfin la génération des architectures logiques. Cette articulation en trois parties permet de générer, par famille de concepts, les solutions, et surtout de ne pas demander à l'utilisateur de rentrer, pour chaque AO de la base de données, le nombre d'occurrences autorisées. Ce nombre est déduit des méta-architectures pour qu'elles puissent être fonctionnelles. De plus, même si le concept de situation de vie est bien connu en ingénierie système, la prise en compte des situations de vie du système à concevoir et des modes de fonctionnement des AO est une nouveauté dans les méthodes automatiques de synthèse. En effet, habituellement, lors de la génération des architectures logiques, seul la situation de vie nominale est pris en compte. Or, dans certains cas, l'étude de la seule situation de vie nominale n'est pas suffisante pour dimensionner un système, ou même pour assurer qu'il sera en mesure de fournir tous les flux nécessaires à chaque situation de vie. Ceci implique que le système à concevoir doit être défini dans chacune des situations de vie qu'il va rencontrer, et indique aussi de définir les différents modes de fonctionnement des AO de la base de données. Les architectures viables seront celles en mesure de fonctionner dans toutes les situations de vie. La détection des isomorphismes se fait à l'aide d'un algorithme novateur par le nombre de différences qu'il prend en compte entre deux architectures. En effet, contrairement à plusieurs algorithmes, celui présenté dans ce manuscrit permet de traiter les graphes MIMO et prend en compte les trois matrices DSM utilisées pour décrire une architecture logique : la matrice des connexions internes et les matrices d'entrée et de sortie. Si deux solutions ont une même architecture logique globale, cela ne veut pas dire que les mêmes AO seront dans le même mode de fonctionnement dans ces deux solutions. Ceci induit des solutions différentes au regard des situations de vie du système, même si l'architecture globale est la même. L'algorithme de détection des isomorphismes prend en compte cette différence.
Concernant les architectures physiques, comme la distinction a été faite dès les architectures logiques, les différentes situations de vie sont aussi prises en compte. Cela permet la création d'un modèle pour le prédimensionnement du système par situation de vie, tout en reliant chacune de ces situations de vie aux variables d'état qui sont fixes vis-à-vis des situations de vie du système. Les différentes variables du système pourront avoir des valeurs différentes suivant la situation de vie. Celles qui sont liées à des exigences de performance doivent être fixées ou, dans d'autres termes, spécifiées. Chaque AO doit être composé d'un ensemble d'équations comportementales par mode de fonctionnement.

La rigueur de fonctionnement imposée par le traitement informatique des données permet de cadrer la définition du système à concevoir. De ce fait, même si l'architecte,

utilisateur final de la méthodologie semi-automatisée, n'utilise pas les résultats de la synthèse, il aura défini les composants utilisables et son système à concevoir d'une manière plus formelle et proche d'autres méthodes d'architecture système, telles que celle de KROB [2014].

D'un point de vue industriel, les travaux ont débouché sur la création d'un démonstrateur qui a pu être déroulé sur un cas complet pertinent. Ce démonstrateur ouvre la voie à un nouveau type d'outil d'aide à la spécification et à la conception architecturale. Il permet à des ingénieurs dans le domaine de la mécanique de modéliser leurs systèmes à l'aide d'outils de l'ingénierie système. L'essor actuel des systèmes drones est une opportunité pour le développement des logiciels de synthèse automatique car une grande variété de solutions existe et une synthèse manuelle ne pourra pas permettre le parcours de tout l'espace de conception.

6.3 Pistes à explorer

Malgré les apports de cette thèse à la communauté scientifique, il reste du travail à mettre en place pour arriver à un outillage complet pouvant permettre le passage de la feuille blanche à un ensemble d'architectures géométriques.
Concernant la génération des architectures logiques, l'algorithme pourrait être amélioré pour être plus efficace. L'objectif serait de regrouper les trois phases de génération en une seule et d'éviter les isomorphismes à la source au lieu de le faire à la fin du processus de synthèse des architectures logiques. En effet, actuellement, dès lors que le nombre d'occurrences d'un AO augmente un peu, ses différentes instances peuvent être permutées, générant ainsi un grand nombre d'isomorphismes. La détection d'isomorphismes est un champ à part entière des mathématiques, et plusieurs algorithmes optimisés ont vu le jour. Malheureusement, aucun de ces algorithmes, du moins ceux évalués lors de la revue bibliographique, ne permet de prendre en compte des graphes MIMO. De plus, des modifications auraient dû être apportées pour prendre en compte toutes les situations de vie d'une même architecture avec ces algorithmes issus de la recherche mathématique. C'est pour cette raison que durant les travaux de thèse, un algorithme de détection des isomorphismes sur des graphes MIMO et représentant différentes situations de vie a été créé. La rencontre de cet algorithme non-optimisé avec des algorithmes provenant de recherches mathématiques pourrait améliorer grandement le temps de calcul nécessaire pour la synthèse architecturale proposée dans ce manuscrit.
Concernant l'analyse des architectures physiques, les équations différentielles pourraient être prises en compte pour obtenir des résultats plus fins. Néanmoins, avec les technologies actuelles de CSP, il n'est pas possible de résoudre des équations différentielles. La solution serait donc de rajouter une étape de simulation sur un autre logiciel en récupérant la définition des AO actuelle avec les variables d'état instanciées, mais en l'enrichissant d'équations comportementales incluant de la dynamique. Les nouvelles solutions provenant de la synthèse n'auront pas forcément le même niveau de sécurité que les architectures utilisées dans le passé, qui ont eu le temps de maturer et il pourrait être très pertinent de mettre en place des exigences sur le niveau de sécurité à atteindre. Ces exigences pourraient être rangées dans les exigences de performance sous la forme d'une probabilité de défaillance pour le système global, calculée à l'aide des probabilités de défaillance des AO sélectionnés pour instancier le système à concevoir.
Concernant la génération des architectures géométriques, la totalité du travail reste à faire. Ce sous-processus devrait être mené en deux étapes avec, dans un premier temps, la création de squelettes géométriques correspondant aux architectures physiques. Dans

un second temps, les modèles géométriques des AO pourraient être accrochés sur ces squelettes ayant le bon placement géométrique et les bonnes dimensions. Cette phase permettrait de représenter les solutions d'une manière très parlante pour un architecte devant rendre des comptes aux responsables du projet sur lequel il travaille, mais l'utilité principale résiderait dans le calcul des premières estimations sur le centre de masse.
Concernant la représentation des architectures logiques, le logiciel Graphviz est utilisé et fournit une représentation sous la forme de fichiers PDF qui peuvent être difficiles à exploiter pour des solutions complexes avec beaucoup d'AO. Un autre format, où une exploration plus aisée serait possible, pourrait être souhaitable. Un graphe cliquable et navigable serait mieux adapté. Par exemple, si pour chaque AO, un clic pouvait faire apparaître sa définition complète avec toutes les variables instanciées, cela améliorerait grandement la compréhension des résultats.

Toutes ces pistes d'amélioration sont réalisables, mais nécessiteraient plus de temps et la mise en place d'une équipe d'informaticiens, qui pourraient reprendre les algorithmes pour créer des codes plus optimisés.

Bibliographie

AGUILAR, F. 1967, *Scanning the business environment*, Studies of the modern corporation, Macmillan. URL https://books.google.fr/books?id=o-M9AAAAIAAJ. 48, 135

AL-HAKIM, L., A. KUSIAK et J. MATHEW. 2000, «A graph-theoretic approach to conceptual design with functional perspectives», *Computer-Aided Design*, vol. 32, nº 14, doi :10.1016/S0010-4485(00)00075-0, p. 867–875, ISSN 0010-4485. URL http://www.sciencedirect.com/science/article/pii/S0010448500000750. 25

ALVAREZ, C., H. KOMOTO, T. VAN BEEK et T. TOMIYAMA. 2012, «Architecture-centric design approach for multi-disciplinary product development», . 37

BAHILL, A. T. et B. GISSING. 1998, «Re-evaluating systems engineering concepts using systems thinking», *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 28, nº 4, p. 516–527. URL http://ieeexplore.ieee.org/abstract/document/725338/. 27

BALLU, A., H. FALGARONE, N. CHEVASSUS et L. MATHIEU. 2006, «A new Design Method based on Functions and Tolerance Specifications for Product Modelling», *CIRP Annals - Manufacturing Technology*, vol. 55, nº 1, doi :10.1016/S0007-8506(07)60384-9, p. 139–142, ISSN 0007-8506. URL http://www.sciencedirect.com/science/article/pii/S0007850607603849. 26

BARBEDIENNE, R., Y. BEN MESSAOUD, J.-Y. CHOLEY, O. PENAS, A. OUSLIMANI et A. RIVIERE. 2015, «SAMOS for Spatial Architecture based on Multi-physics and Organisation of Systems in conceptual design», dans *Systems Engineering (ISSE), 2015 IEEE International Symposium on*, IEEE, p. 135–141. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7302746. 16, 17

VAN BEEK, T. J., M. S. ERDEN et T. TOMIYAMA. 2010, «Modular design of mechatronic systems with function modeling», *Mechatronics*, vol. 20, nº 8, doi :10.1016/j.mechatronics.2010.02.002, p. 850–863, ISSN 0957-4158. URL http://www.sciencedirect.com/science/article/pii/S0957415810000310. 25

BEN HAMIDA, S., M. JANKOVIC, M. CALLOT, A. MONCEAUX et C. ECKERT. 2015, «Towards a decision support framework for system architecture design», URL http://oro.open.ac.uk/43911/. 29

BEN MESSAOUD, Y., M. HAMMADI, T. HASSINE et O. HAMMAMI. 2014, «Multidisciplinary optimization of a quad-rotor by integrating multi-level models», dans *Mecatronics (MECATRONICS), 2014 10th France-Japan/8th Europe-Asia Congress on*, IEEE, p. 337–342. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7018616. 26

BONEV, M., L. HVAM, J. CLARKSON et A. MAIER. 2015, «Formal computer-aided product family architecture design for mass customization», *Computers in Industry*, vol. 74, doi :10.1016/j.compind.2015.07.006, p. 58–70, ISSN 0166-3615. URL http://www.sciencedirect.com/science/article/pii/S0166361515300245. 15, 23

BONNET, S. 2016, «Acquisition and Early Evaluation of Architecture with Arcadia and Capella», . 29, 44

BRACEWELL, R. H., R. V. CHAPLIN et D. A. BRADLEY. 1992, «Schemebuilder and layout : computer based tools to aid the design of mechatronic systems», dans *IMech Conference on Mechatronics—The Integration of Engineering Design, Dundee, UK*, Citeseer, p. 1–7. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.161.8243&rep=rep1&type=pdf. 18, 23

BURGE, S. 2013, «The systems engineering tool box», *London, UK*, p. 1. URL http://www.burgehugheswalsh.co.uk/uploaded/documents/fm-tool-box-v1.0.pdf. 19

CAGAN, J., M. I. CAMPBELL, S. FINGER et T. TOMIYAMA. 2005, «A Framework for Computational Design Synthesis : Model and Applications», *Journal of Computing and Information Science in Engineering*, vol. 5, nº 3, doi :10.1115/1.2013289, p. 171–181, ISSN 1530-9827. URL http://dx.doi.org/10.1115/1.2013289. 9, 14, 15, 23, 26, 29, 34, 94

CHAKRABARTI, A. et L. BLESSING. 1996, «Special issue : representing functionality in design», *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing*, vol. 10, nº 04, p. 251–253. URL http://journals.cambridge.org/abstract_S0890060400001608. 17

CHANDRASEKARAN, B. et J. R. JOSEPHSON. 2000, «Function in device representation», *Engineering with computers*, vol. 16, nº 3, p. 162–177. 50

CHAPON, D. et G. BOUCHEZ. 2009, «On the link between Architectural Description Models and Modelica Analyses Models», p. 784–789, doi :10.3384/ecp09430079. URL http://www.ep.liu.se/ecp_article/index.en.aspx?issue=043;article=92. 17, 112

CHENOUARD, R. 2007, *Résolution par satisfaction de contraintes appliquée à l'aide à la décision en conception architecturale*, phdthesis, Arts et Métiers ParisTech. URL https://tel.archives-ouvertes.fr/tel-00486662/document. 27, 112

CHENOUARD, R., L. GRANVILLIERS et R. SOTO. 2008, «Model-driven constraint programming», dans *Proceedings of the 10th international ACM SIGPLAN conference on Principles and practice of declarative programming*, ACM, p. 236–246. URL http://dl.acm.org/citation.cfm?id=1389479. 2, 16

CHIKHAOUI, Z. 2013, *Contribution à la modélisation énergétique des hélicoptères en vue de la maîtrise de leurs comportements dynamiques*, thèse de doctorat, Paris, ENSAM. URL http://www.theses.fr/2013ENAM0061. 16

CHIOU, S.-J. et K. SRIDHAR. 1999, «Automated conceptual design of mechanisms», *Mechanism and Machine Theory*, vol. 34, nº 3, doi :10.1016/S0094-114X(98)00037-8, p. 467–495, ISSN 0094-114X. URL http://www.sciencedirect.com/science/article/pii/S0094114X98000378. 21

CHRISTOPHE, F., A. BERNARD et E. COATANÉA. 2010, «Rfbs : A model for knowledge representation of conceptual design», *CIRP Annals Manufacturing Technology*, vol. 59, nº 1, doi :10.1016/j.cirp.2010.03.105, p. 155–158, ISSN 0007-8506. URL http://www.sciencedirect.com/science/article/pii/S000785061000106X. 20

CHRISTOPHE, F., S. RAIVO, A. BERNARD et E. COATANÉA. 2009, «Opas : Ontology processing for assisted synthesis», ASME, San Diego, doi :10.1115/DETC2009-87776. 2, 14, 19

CORTELLESSA, V. et L. FRITTELLA. 2007, «A framework for automated generation of architectural feedback from software performance analysis», dans *Formal Methods and Stochastic Models for Performance Evaluation*, Springer, p. 171–185. URL http://link.springer.com/chapter/10.1007/978-3-540-75211-0_13. 24

CRAWLEY, E., O. DE WECK, S. EPPINGER, C. MAGEE, J. MOSES, W. SEERING, J. SCHINDALL, D. WALLACE et D. WHITNEY. 2004, «Engineering systems monograph», *Massachusetts Institute of Technology, Tech. Rep.* URL http://mit.sustech.edu/NR/rdonlyres/0708296E-EAFB-4FA9-9710-56E7B20AB34A/0/arch_esd.pdf. 4

DAUPHIN-TANGUY, G. 1999, «Les bond graphs et leur application en mécatronique», *Techniques de l'ingénieur*, , nº s7222. 19

DAUPHIN-TANGUY, G. 2010, «La méthodologie bond graph - Principes et langage», *REE Revue de l'Electricité et de l'Electronique*, vol. 2, p. 89–98, ISSN 1265-6534. 16, 35

DAVID F. WYATT, D. C. W. 2011, «Supporting product architecture design using computational design synthesis with network structure constraints», *Research in Engineering Design*, vol. 23, nº 1, doi :10.1007/s00163-011-0112-y, p. 17–52, ISSN 0934-9839. 23, 33

DEMOLY, F. 2010, *Conception intégrée et gestion d'informations techniques : application à l'ingénierie du produit et de sa séquence d'assemblage*, thèse de doctorat, Université de technologie de Belfort-Montbéliard. 26

DEMOLY, F., N. TROUSSIER, B. EYNARD, H. FALGARONE, B. FRICERO et S. GOMES. 2011, «Proactive assembly oriented design approach based on the deployment of functional requirements», *Journal of Computing and Information Science in Engineering*, vol. 11, nº 1, p. 014 501. URL http://computingengineering.asmedigitalcollection.asme.org/article.aspx?articleid=1402300. 26, 27

DENG, Y.-M. et W. F. LU. 2009, «A Software System Embodying the Support for Conceptual Design Synthesis», IEEE, ISBN 978-0-7695-3642-2, p. 615–620, doi :10.1109/SNPD.2009.50. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5286601. 28

DEWULF, S. et D. MANN. 2014, «CreaTRIZ Systematic Creativity for Innovation : Access the world's ...», URL https://www.yumpu.com/en/document/view/27009095/creatriz-systematic-creativity-for-innovation-access-the-worlds-. 25

FAISANDIER, A. 2015a, «Concevoir les architectures fonctionnelle et physique des systèmes complexes - Techniques de l'Ingénieur», URL http://www.techniques-ingenieur.fr/fiche-pratique/genie-industriel-th6/deployer-l-innovation-dt30/concevoir-les-architectures-fonctionnelle-et-physique-des-systemes-complexes-0271/. 20, 49

FAISANDIER, A. 2015b, *System Notion and Engineering of Systems – Volume: 1, Engineering and Architecting Multidisciplinary Systems*, vol. 1, Sinergy'Com, ISBN 979-10-91699-06-8. 4, 6, 15

FALGARONE, H. et N. CHEVASSUS. 2006, *Structural and Functional Analysis for Assemblies*, Springer London, London, ISBN 978-1-84628-210-2, p. 87–96, doi :10.1007/1-84628-210-1_7. URL https://doi.org/10.1007/1-84628-210-1_7. 40

FRANÇOIS CHRISTOPHE, F. M. 2014, «A methodology supporting syntactic, lexical and semantic clarification of requirements in systems engineering», *International Journal of Product Development*, vol. 19, nº 4, doi :10.1504/IJPD.2014.062973, p. 173–190, ISSN 1741-8178. 49

GADEYNE, K., G. PINTE et K. BERX. 2014, «Describing the design space of mechanical computational design synthesis problems», *Advanced Engineering Informatics*, vol. 28, nº 3, doi :10.1016/j.aei.2014.03.004, p. 198–207, ISSN 1474-0346. URL http://www.sciencedirect.com/science/article/pii/S1474034614000329. 24, 27

GALLAHER, A. 2017, «Method for estimating inertial properties of rotorcraft in conceptual design», dans *Proceedings of the 73rd Annual Forum*, AHS International, Fort Worth. 9

GANE, V. et J. HAYMAKER. 2012, «Design Scenarios : Enabling transparent parametric design spaces», *Advanced Engineering Informatics*, vol. 26, nº 3, doi :10.1016/j.aei.2012.04.008, p. 618–640, ISSN 1474-0346. URL http://www.sciencedirect.com/science/article/pii/S147403461200047X. 26

GANSNER, E. R. et S. C. NORTH. 2000, «An open graph visualization system and its applications to software engineering», *SOFTWARE - PRACTICE AND EXPERIENCE*, vol. 30, nº 11, p. 1203–1233. 94

GERO, J. S. 1990, «Design prototypes : a knowledge representation schema for design», *AI magazine*, vol. 11, nº 4, p. 26. URL http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/854. 20, 22, 33

GIAMPÀ, F., M. MUZZUPAPPA, S. RIZZUTI et OTHERS. 2004, «Design by function : a methodology to support designer creativity», dans *DS 32 : Proceedings of DESIGN 2004, the 8th International Design Conference, Dubrovnik, Croatia*. URL https://www.designsociety.org/publication/19757/design_by_function_a_methodology_to_support_designer_creativity. 21, 49

HAMIDA, S. B., A. GRANDOU, M. JANKOVIC, C. ECKERT, A. HUET et J.-C. BOCQUET. 2015, «A Comparative Case Study of Functional Models to Support System Architecture Design», *Procedia Computer Science*, vol. 44, doi :10.1016/j.procs.2015.03.058, p.

325–335, ISSN 18770509. URL http://linkinghub.elsevier.com/retrieve/pii/S187705091500294X. 17

HAMPSON, K. 2015, «Technical Evaluation of the Systems Modeling Language (SysML)», *Procedia Computer Science*, vol. 44, doi :10.1016/j.procs.2015.03.054, ISSN 1877-0509. URL http://www.sciencedirect.com/science/article/pii/S1877050915002902. 17, 95

HAN, Y.-H. et K. LEE. 2006, «A case-based framework for reuse of previous design concepts in conceptual synthesis of mechanisms», *Computers in Industry*, vol. 57, nº 4, doi : 10.1016/j.compind.2005.09.005, p. 305–318, ISSN 01663615. URL http://linkinghub.elsevier.com/retrieve/pii/S0166361505001752. 14

HARTMANN, C., R. CHENOUARD, E. MERMOZ et A. BERNARD. 2018, «A framework for automatic architectural synthesis in conceptual design phase», *A paraître*. 76, 152

HARTMANN, C., E. MERMOZ, R. CHENOUARD et A. BERNARD. 2017, «A conceptual framework leading to different dynamic systems architectures», dans *Proceedings of the 73rd Annual Forum*, AHS International, Fort Worth. 44

HELMS, B. 2012, *Object-Oriented Graph Grammars for Computational Design Synthesis*, thèse de doctorat, TECHNISCHE UNIVERSITÄT MÜNCHEN. 17, 33, 34, 35

HELMS, B. et K. SHEA. 2012, «Computational Synthesis of Product Architectures Based on Object-Oriented Graph Grammars», *Journal of Mechanical Design*, vol. 134, nº 2, doi :10.1115/1.4005592, p. 021 008–021 008, ISSN 1050-0472. URL http://dx.doi.org/10.1115/1.4005592. 32, 160

HENDERSON, R. M. et K. B. CLARK. 1990, «Architectural innovation : The reconfiguration of existing», *Administrative Science Quarterly*, vol. 35, p. 9–30. 23

HIRTZ, J., R. B. STONE, D. A. MCADAMS, S. SZYKMAN et K. L. WOOD. 2002, «A functional basis for engineering design: Reconciling and evolving previous efforts», *Research in Engineering Design*, vol. 13, nº 2, doi :10.1007/s00163-001-0008-3, p. 65–82, ISSN 0934-9839, 1435-6066. URL http://link.springer.com/article/10.1007/s00163-001-0008-3. 5, 18, 19, 33, 78, 125

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. 2005, *International classification for standards: ICS*, International Organization for Standardization, Genève, Switzerland, ISBN 978-92-67-10405-8. 21

IWASAKI, Y., R. FIKES, M. VESCOVI et B. CHANDRASEKARAN. 1993, «How things are intended to work : Capturing functional knowledge in device design», dans *IJCAI*, Citeseer, p. 1516–1522. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.5252&rep=rep1&type=pdf. 18

JANTHONG, N., D. BRISSAUD et S. BUTDEE. 2010, «Combining axiomatic design and case-based reasoning in an innovative design methodology of mechatronics products», *CIRP Journal of Manufacturing Science and Technology*, vol. 2, nº 4, doi :10.1016/j.cirpj.2010.05.005, p. 226–239, ISSN 1755-5817. URL http://www.sciencedirect.com/science/article/pii/S1755581710000581. 20

JOHNSON, T., A. KERZHNER, C. J. PAREDIS et R. BURKHART. 2012, «Integrating models and simulations of continuous dynamics into sysml», *Journal of Computing and Information Science in Engineering*, vol. 12, nº 1, p. 011 002. 39

KANG, S. W. et C. TUCKER. 2015, «An automated approach to quantifying functional interactions by mining large-scale product specification data», *Journal of Engineering Design*, doi :10.1080/09544828.2015.1083539, p. 1–24, ISSN 0954-4828, 1466-1837. URL http://www.tandfonline.com/doi/full/10.1080/09544828.2015.1083539. 19

KÖNIGSEDER, C. et K. SHEA. 2015, «Comparing Strategies for Topologic and Parametric Rule Application in Automated Computational Design Synthesis», *Journal of Mechanical Design*, doi :10.1115/1.4031714, ISSN 1050-0472. URL http://mechanicaldesign.asmedigitalcollection.asme.org/article.aspx?doi=10.1115/1.4031714. 39

KOMOTO, H., S. KONDOH, K. MASUI et A. TEZUKA. 2014, «Parameter-oriented Visualization of a Modelica Model with a Numerical Data Integration Feature», *Procedia CIRP*, vol. 21, doi :10.1016/j.procir.2014.03.138, p. 40–45, ISSN 2212-8271. URL http://www.sciencedirect.com/science/article/pii/S2212827114006787. 39, 112

KOMOTO, H. et T. TOMIYAMA. 2010a, «Computational Support for System Architecting», doi : 10.1115/DETC2010-28683, p. 25–34. URL http://dx.doi.org/10.1115/DETC2010-28683. 37

KOMOTO, H. et T. TOMIYAMA. 2010b, «A system architecting tool for mechatronic systems design», *CIRP Annals - Manufacturing Technology*, vol. 59, nº 1, doi :10.1016/j.cirp.2010.03.104, p. 171–174, ISSN 0007-8506. URL http://www.sciencedirect.com/science/article/pii/S0007850610001058. 37

KOMOTO, H. et T. TOMIYAMA. 2011, «Multi-disciplinary system decomposition of complex mechatronics systems», *CIRP Annals - Manufacturing Technology*, vol. 60, nº 1, doi :10.1016/j.cirp.2011.03.102, p. 191–194, ISSN 0007-8506. URL http://www.sciencedirect.com/science/article/pii/S000785061100103X, 00017. 37

KOMOTO, H. et T. TOMIYAMA. 2012, «A framework for computer-aided conceptual design and its application to system architecting of mechatronics products», *Computer-Aided Design*, vol. 44, nº 10, doi :10.1016/j.cad.2012.02.004, p. 931–946, ISSN 0010-4485. URL http://www.sciencedirect.com/science/article/pii/S0010448512000401. 37

KROB, D. 2014, «Éléments de systémique. Architecture des systèmes», dans *Complexité-Simplexité*, édité par A. Berthoz et J.-L. Petit, Collège de France, ISBN 978-2-7226-0330-1. URL http://books.openedition.org/cdf/3388, dOI : 10.4000/books.cdf.3388. 17, 18, 29, 30, 44, 47, 49, 50, 73, 161

KRYSSANOV, V., H. TAMAKI et K. UEDA. 1999, «Synthesis and emergence in engineering design problem solving», dans *Environmentally Conscious Design and Inverse Manufacturing, 1999. Proceedings. EcoDesign '99 : First International Symposium On*, p. 690–695, doi :10.1109/ECODIM.1999.747699. 12

LI, C. L., K. W. CHAN et S. T. TAN. 1999, «A configuration space approach to the automatic design of multiple-state mechanical devices», *Computer-Aided Design*, vol. 31, nº 10, doi :10.1016/S0010-4485(99)00058-5, p. 621–653, ISSN 0010-4485. URL http://www.sciencedirect.com/science/article/pii/S0010448599000585. 14

LO, M. 2013, *Contribution à l'évaluation d'architectures en Ingénierie Système : application en conception de systèmes mécatroniques*, phdthesis, Université Montpellier II - Sciences et Techniques du Languedoc. URL https://tel.archives-ouvertes.fr/tel-00918890/document. 26, 112

M ISHII, T. T. 1996, «A synthetic reasoning method based on a physical phenomenon knowledge base», . 38

MAKKONEN, P. et J.-G. PERSSON. 1994, «Configuration and Dimensional Synthesis in Mechanical Design : an Application for Planar Mechanisms», *CIRP Annals - Manufacturing Technology*, vol. 43, nº 1, doi :10.1016/S0007-8506(07)62183-0, p. 145–148, ISSN 0007-8506. URL http://www.sciencedirect.com/science/article/pii/S0007850607621830, 00004. 24

MARRIOTT, K., P. J. STUCKEY, L. D. KONINCK et H. SAMULOWITZ. 2014, *A minizinc tutorial*, Technical report, 2015. http ://www. minizinc. org/downloads/doc-latest/minizinc-tute. pdf. URL http://www.minizinc.org/tutorial/minizinc-tute.pdf. 86

MATAR, J. 2013, *Une méthodologie d'aide à la décision en conception architecturale (proposition d'une approche outillée)*, thèse de doctorat, Ecole Centrale de Nantes. 2, 17

MERRIS, R. 1994, «Laplacian matrices of graphs: a survey», *Linear Algebra and its Applications*, vol. 197-198, doi :10.1016/0024-3795(94)90486-3, p. 143–176, ISSN 00243795. URL http://linkinghub.elsevier.com/retrieve/pii/0024379594904863. 92

MHENNI, F., J.-Y. CHOLEY, O. PENAS, R. PLATEAUX et M. HAMMADI. 2014, «A SysML-based methodology for mechatronic systems architectural design», *Advanced Engineering Informatics*, vol. 28, nº 3, doi :10.1016/j.aei.2014.03.006, p. 218–231, ISSN 14740346. URL http://linkinghub.elsevier.com/retrieve/pii/S1474034614000342. 17

MILLER, G. A. 1956, «The magical number seven, plus or minus two : some limits on our capacity for processing information.», *Psychological Review*, vol. 63, nº 2, doi : 10.1037/h0043158, p. 81–97, ISSN 1939-1471, 0033-295X. URL http://doi.apa.org/getdoi.cfm?doi=10.1037/h0043158. 29

MÜNZER, C., B. HELMS et K. SHEA. 2013, «Automatically Transforming Object-Oriented Graph-Based Representations Into Boolean Satisfiability Problems for Computational Design Synthesis», *Journal of Mechanical Design*, vol. 135, nº 10, doi :10.1115/1.4024850, p. 101 001–101 001, ISSN 1050-0472. URL http://dx.doi.org/10.1115/1.4024850. 35, 36, 160

MOULLEC, M.-L., M. BOUISSOU, M. JANKOVIC, J.-C. BOCQUET, F. RÉQUILLARD, O. MAAS et O. FORGEOT. 2013, «Toward system architecture generation and performances assessment under uncertainty using Bayesian networks», *Journal of Mechanical Design*, vol. 135, nº 4, p. 041 002. URL http://appliedmechanics.asmedigitalcollection.asme.org/article.aspx?articleid=1685815. 39

NADEAU, J.-P. et Y. LEDOUX. 2014, «L'éco-innovation par la méthode eco-mal'in», *Techniques de l'ingénieur Éco-conception : concepts et méthodes*, vol. base documentaire : TIB566DUO., nº ref. article : ag6783. URL http://www.techniques-ingenieur.fr/base-documentaire/innovation-th10/eco-conception-concepts-et-methodes-42566210/l-eco-innovation-par-la-methode-eco-mal-in-ag6783/. 40

NASA. 2007, *NASA Systems Engineering Handbook*, ISBN 978-0-16-079747-7. 44, 49, 74, 76, 135

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. 1993, «Integration definition for function modeling (IDEF0)», . 25

OULD BOUAMAMA, B. et G. DAUPHINTANGUY. 2006, «Modélisation par bond graph application aux systèmes énergétiques», *Techniques de l'ingénieur. Génie énergétique*, , nº BE 8 281 1. 16

PAHL, G., W. BEITZ, J. FELDHUSEN, K.-H. GROTE, K. WALLACE et L. T. BLESSING, éd.. 2007, *Engineering design : a systematic approach*, 3e éd., Springer, London, ISBN 978-1-84628-318-5 978-1-84628-319-2. 5, 25, 33, 35, 50, 62, 74

PAILHÈS, J., M. SALLAOU, J.-P. NADEAU et G. FADEL. 2009, «Energy based functional decomposition in preliminary design», . 5, 22, 74

PAL, U., Y.-C. LIU et A. CHAKRABARTI. 2014, «Evaluating FuncSION: A software for automated synthesis of design solutions for stimulating ideation during mechanical conceptual design», *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 28, nº 03, doi :10.1017/S0890060414000183, p. 209–226, ISSN 0890-0604, 1469-1760. URL http://www.journals.cambridge.org/abstract_S0890060414000183. 31, 32

PAREDIS, C. J., Y. BERNARD, R. M. BURKHART, H.-P. KONING, S. FRIEDENTHAL, P. FRITZSON, N. F. ROUQUETTE et W. SCHAMAI. 2010, «5.5. 1 An Overview of the SysML-Modelica Transformation Specification», dans *INCOSE International Symposium*, vol. 20, Wiley Online Library, p. 709–722. URL http://onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.2010.tb01099.x/full. 39

PARK, G.-J. 2007, *Analytic Methods for Design Practice*, Springer Science & Business Media, ISBN 978-1-84628-472-4. 20

PEARCE, P. et S. FRIEDENTHAL. 2013, «A Practical Approach For Modelling Submarine Subsystem Architecture In SysML», . 31

RECHTIN, E. 1991, *Systems architecting : creating and building complex systems*, Prentice Hall, Englewood Cliffs, N.J, ISBN 0-13-880345-5. 20, 28

RENIER, R. et R. CHENOUARD. 2011, «DE SYSML A MODELICA AIDE A LA FORMALISATION DE MODELES DE SIMULATION EN CONCEPTION PRELIMINAIRE», dans *12ème Colloque National AIP PRIMECA*, France. URL https://hal.archives-ouvertes.fr/hal-00585100. 39, 112

SCARAVETTI, D. 2004, «Formalisation préalable d'un problème de conception, pour l'aide à la décision en conception préliminaire», . 31

SIMONS, A. J. 2003, «The Theory of Classification, Part 9: Inheritance and Self-Reference.», *The Journal of Object Technology*, vol. 2, nº 6, doi :10.5381/jot.2003.2.6.c2, p. 25, ISSN 1660-1769. URL http://www.jot.fm/contents/issue_2003_11/column2.html. 22

STONE, R. B. et K. L. WOOD. 1999, «Development of a Functional Basis for Design», *Journal of Mechanical Design*, vol. 122, nº 4, doi :10.1115/1.1289637, p. 359–370, ISSN 1050-0472. URL http://dx.doi.org/10.1115/1.1289637. 18

ULRICH, K. T. et W. P. SEERING. 1989, «Synthesis of schematic descriptions in mechanical design», *Research in Engineering Design*, vol. 1, nº 1, p. 3–18. URL http://link.springer.com/article/10.1007/BF01579999. 35

UMEDA, Y., M. ISHII, M. YOSHIOKA, Y. SHIMOMURA et T. TOMIYAMA. 1996, «Supporting conceptual design based on the function-behavior-state modeler», *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing*, vol. 10, nº 04, doi : 10.1017/S0890060400001621, p. 275, ISSN 0890-0604, 1469-1760. URL http://www.journals.cambridge.org/abstract_S0890060400001621. 14, 25, 37

UMEDA, Y. et T. TOMIYAMA. 1997, «Functional reasoning in design», *IEEE Expert*, vol. 12, nº 2, doi :10.1109/64.585103, p. 42–48, ISSN 0885-9000. 18, 20

WEILKIENS, T. 2006, *Systems engineering mit SysML/UML : Modellierung, Analyse, Design*, 1re éd., dpunkt.verl, Heidelberg, ISBN 978-3-89864-409-9. OCLC : 180005733. 27

WIXSON, J. R. 1999, «Function Analysis and Decomposistion using Function Analysis Systems Technique», dans *INCOSE International Symposium*, vol. 9, Wiley Online Library, p. 800–805. URL http://onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.1999.tb00241.x/abstract. 20, 76

YANG, K., Y. LI, Y. XIONG, J.-Y. YAN et H.-Z. NA. 2015, «A model for computer-aided creative design based on cognition and iteration», *Proceedings of the Institution of Mechanical Engineers, Part C : Journal of Mechanical Engineering Science*, doi : 10.1177/0954406215611438, ISSN 0954-4062, 2041-2983. URL http://pic.sagepub.com/lookup/doi/10.1177/0954406215611438. 25

YUAN, L., Y. LIU, Z. SUN, Y. CAO et A. QAMAR. 2016, «A hybrid approach for the automation of functional decomposition in conceptual design», *Journal of Engineering Design*, p. 1–28. URL http://www.tandfonline.com/doi/abs/10.1080/09544828.2016.1146237. 20

ZHENG, C., J. LE DUIGOU, M. BRICOGNE et B. EYNARD. 2016, «Multidisciplinary interface model for design of mechatronic systems», *Computers in Industry*, vol. 76, doi :10.1016/j.compind.2015.12.002, p. 24–37, ISSN 01663615. URL http://linkinghub.elsevier.com/retrieve/pii/S0166361515300658. 15

Annexe A

Annexes

A.1 Figures annexes

A.1.1 Synthèse des architectures hélicoptères du point de vue de la sustentation

Pour la construction de la base de données des AO, il est nécessaire de connaître les technologies utilisées dans les systèmes similaires à celui qui doit être conçu. Pour ce faire, une synthèse manuelle a été conduite et est présentée ci-après sur la Figure A.1. Toutes les technologies identifiées lors de cette synthèse manuelle ne sont pas traduites en AOs pour l'exemple fil rouge traité dans cette thèse. La puissance de calcul nécessaire pour réaliser une synthèse automatisée ne correspond pas au matériel disponible lors de la rédaction de ce manuscrit de thèse.

La synthèse manuelle est présentée sous la forme d'un arbre avec chaque branche correspondant à des sous-familles. La famille mère est l'«aéronef». Chaque sous-famille :

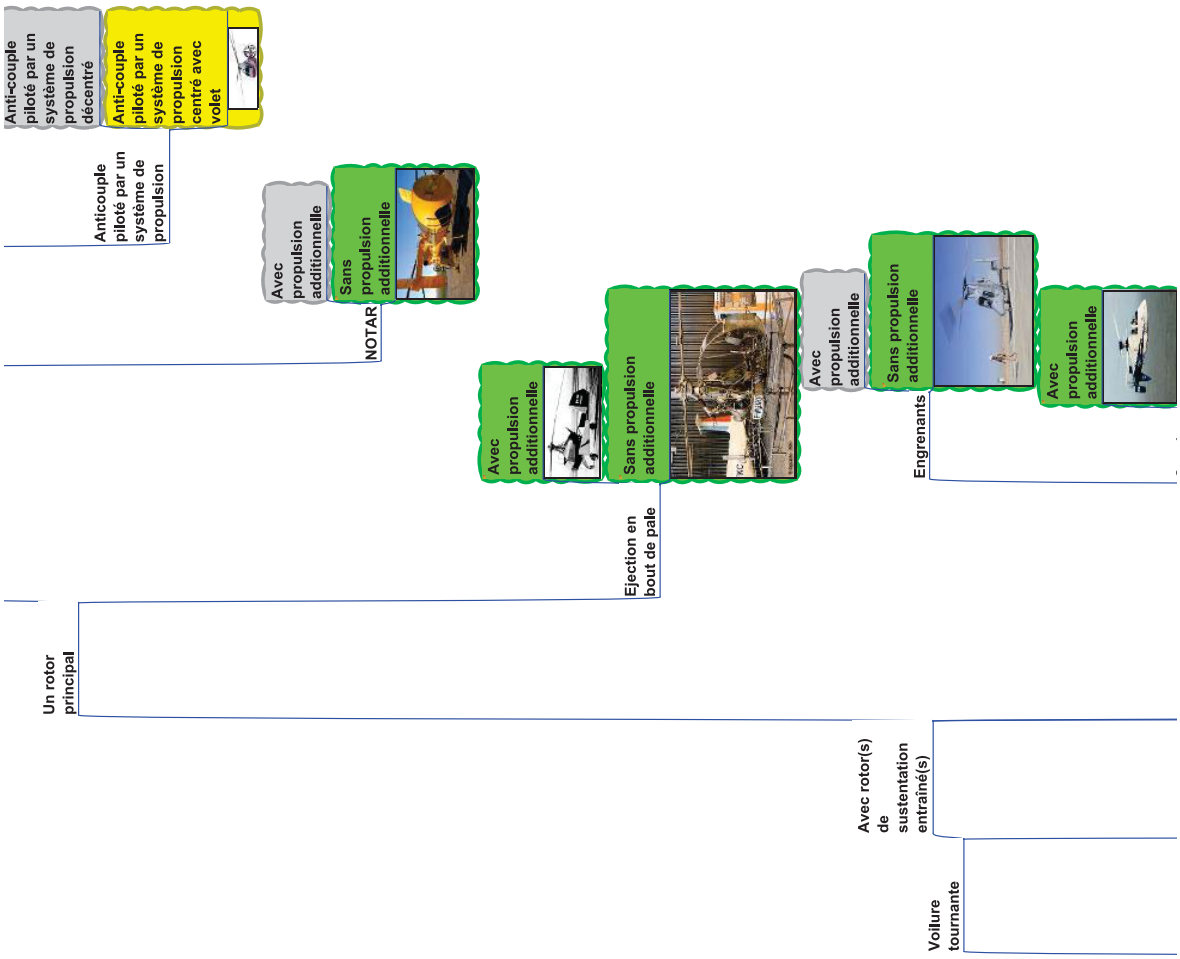
— qui n'est pas traitée, est surlignée en rouge,

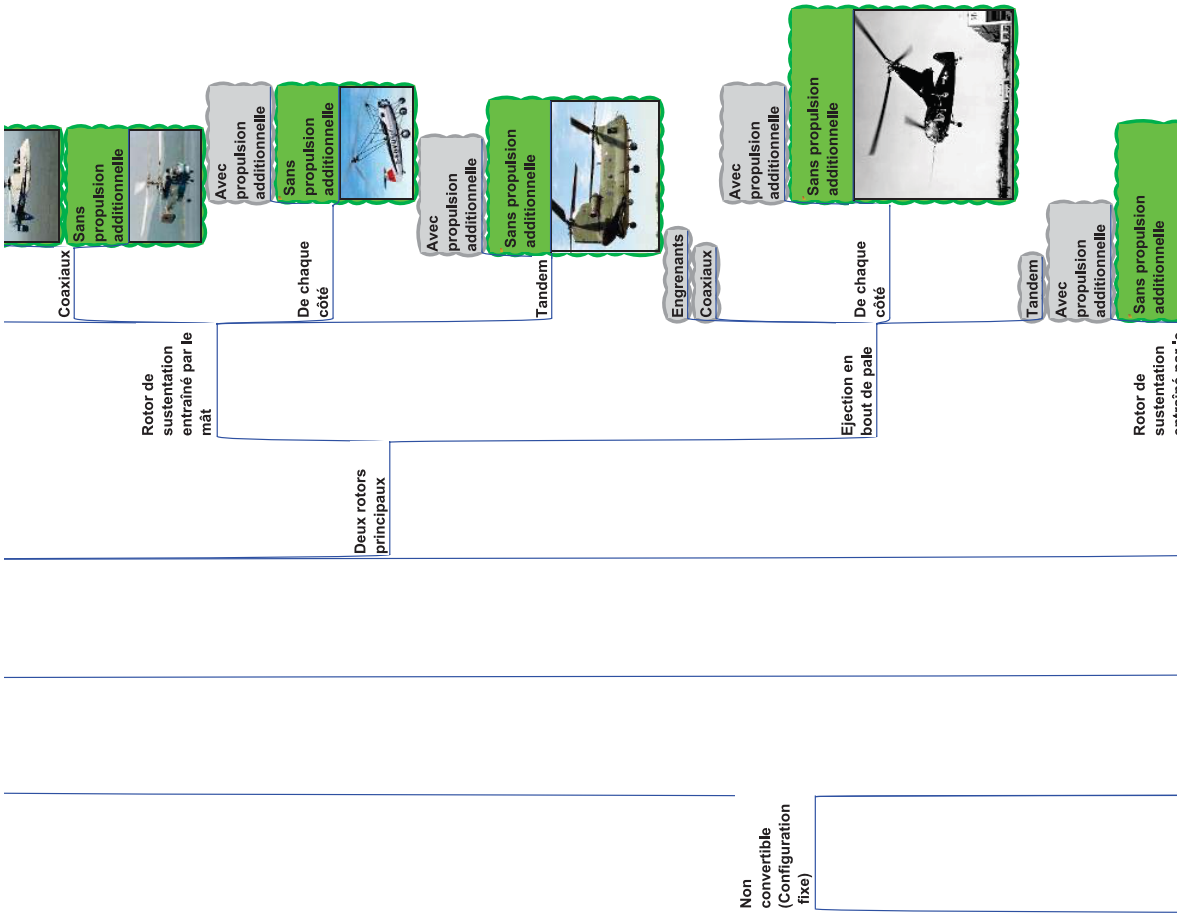
— qui arrive à la dernière feuille, mais sans implémentation dans l'industrie aéronautique, est surlignée en gris,

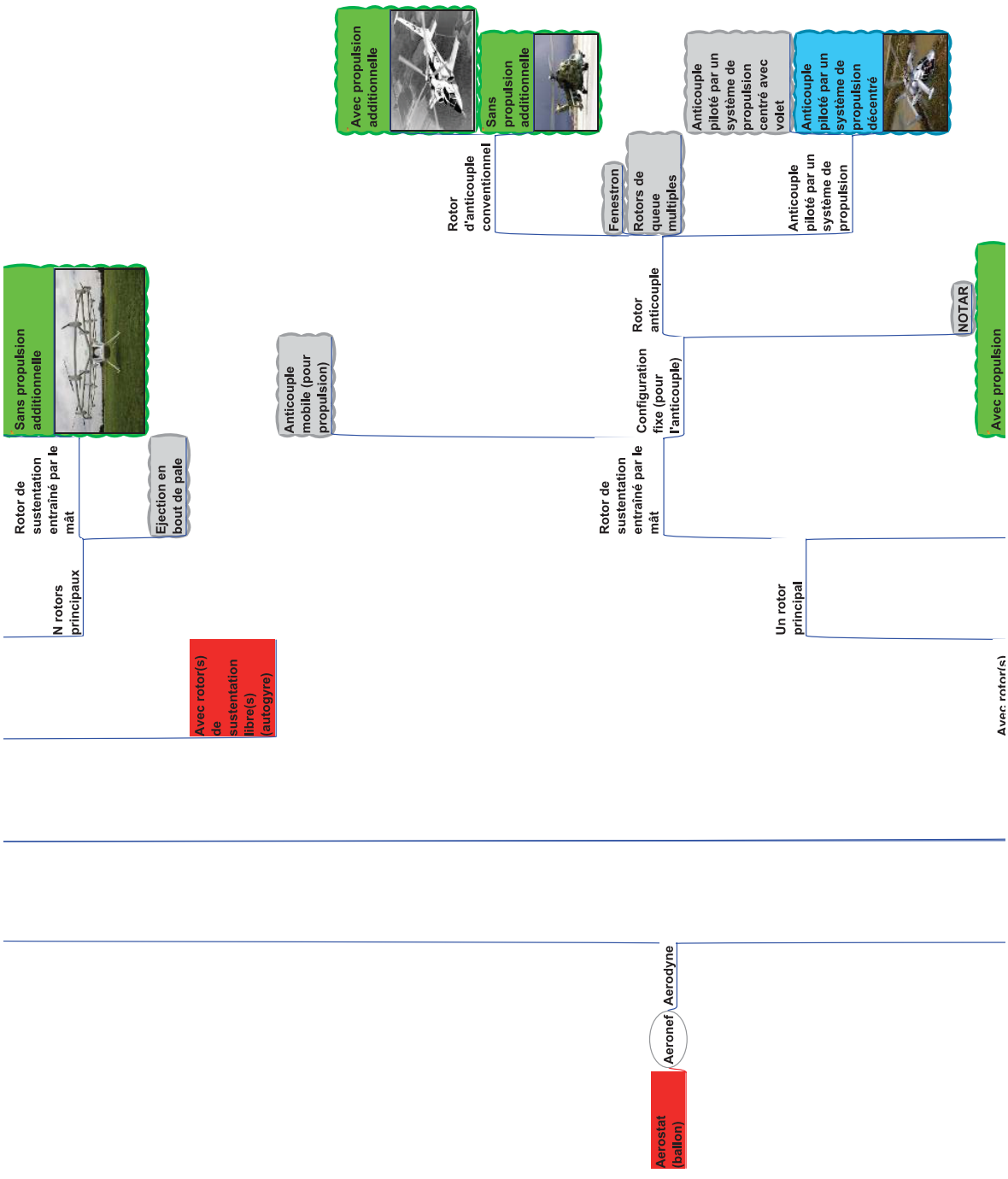
— qui arrive à la dernière feuille, avec une implémentation au stade de projet, est surlignée en jaune,

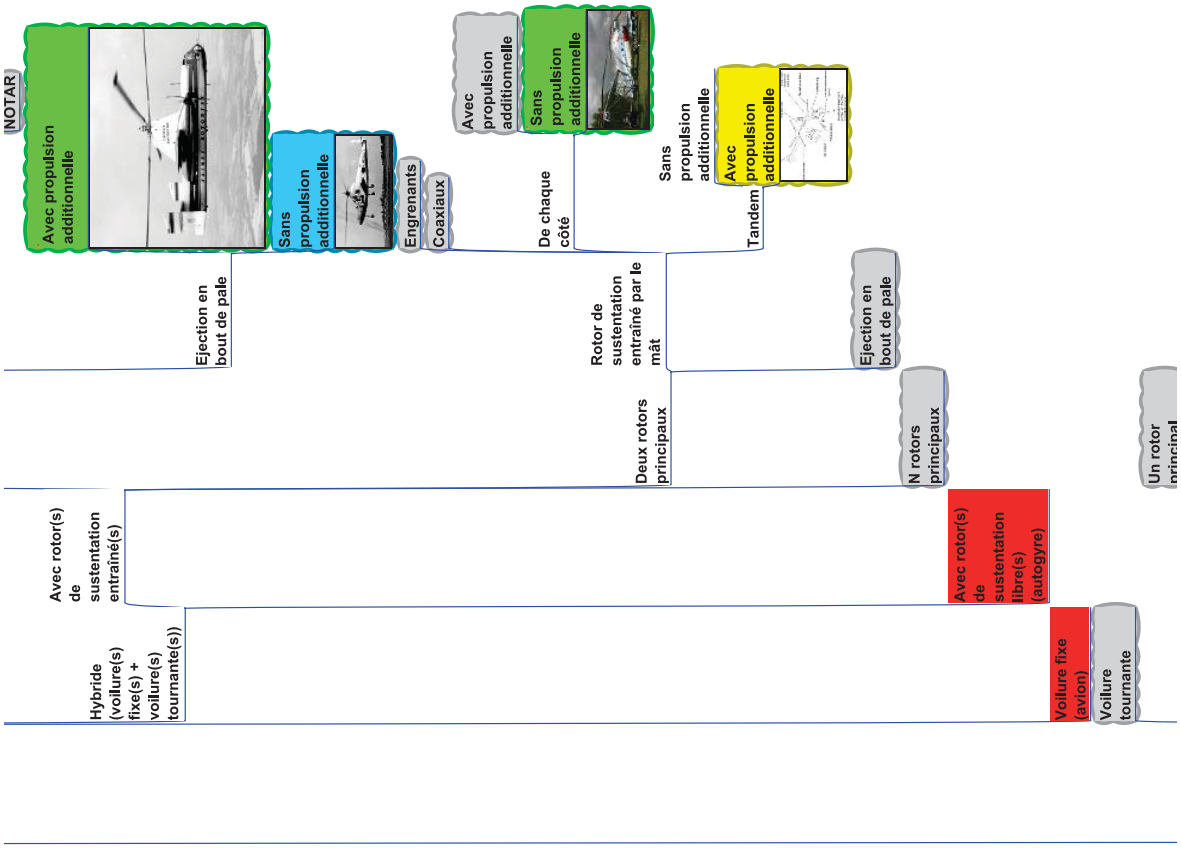
— qui arrive à la dernière feuille, avec une implémentation qui vole, est surlignée en vert ou bleu.

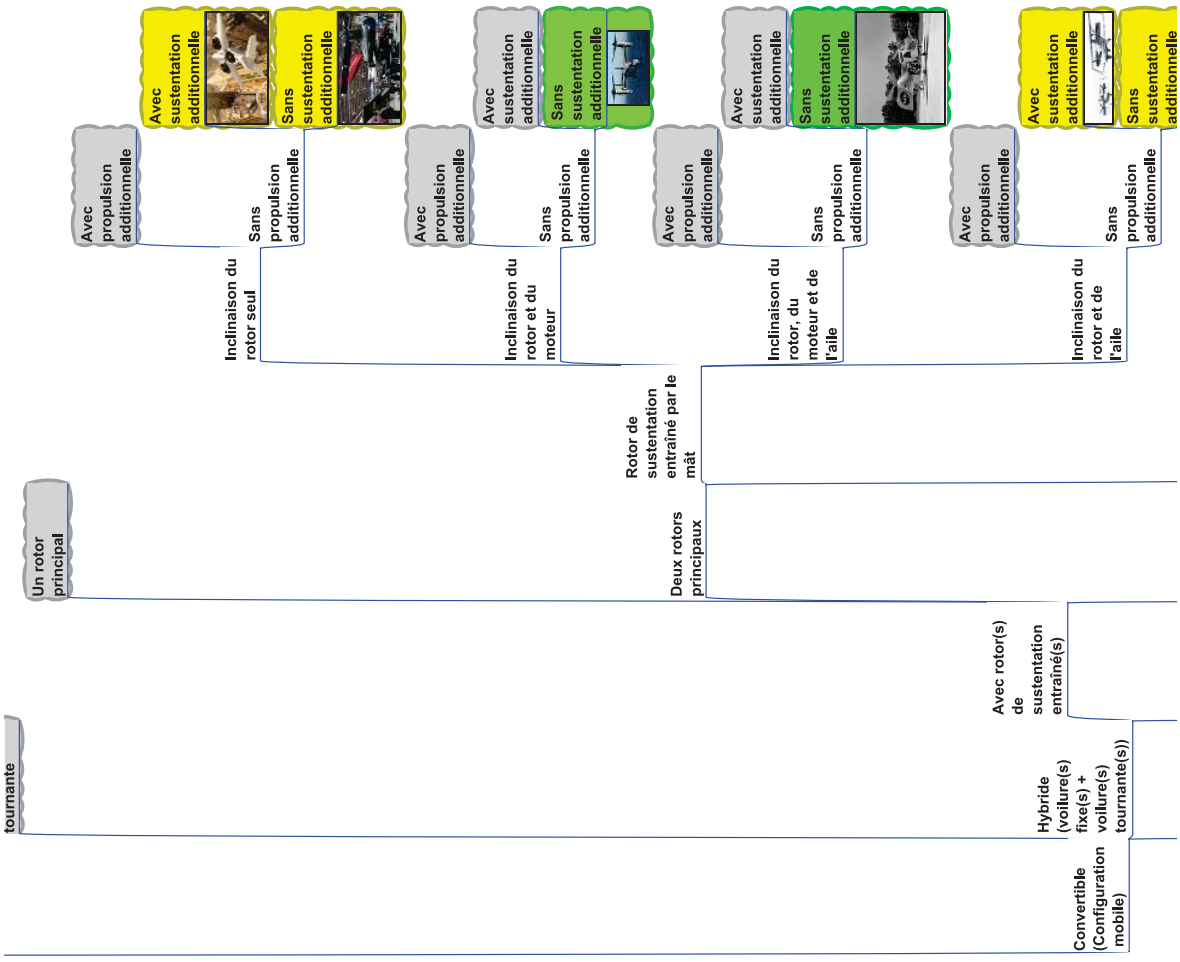
Un rotor
Rotor de sustentation entraîné par le mât
Configuration fixe (pour l'anticouple)
Anticouple mobile (pour propulsion)
Rotor anticouple
Rotor d'anticouple conventionnel
Avec propulsion additionnelle
Sans propulsion additionnelle
Fenestron
Avec propulsion additionnelle
Sans propulsion additionnelle
Rotors de queue multiples
Avec propulsion additionnelle
Sans propulsion additionnelle
Anti-couple piloté par un système de


Anti-couple piloté par un système de propulsion décentré
Anti-couple piloté par un système de propulsion centré avec volet
Anticouple piloté par un système de propulsion
Avec propulsion additionnelle
Sans propulsion additionnelle
NOTAR
Avec propulsion additionnelle
Sans propulsion additionnelle
Ejection en bout de pale
Un rotor principal
Avec propulsion additionnelle
Sans propulsion additionnelle
Engrenants
Avec propulsion additionnelle
Avec rotor(s) de sustentation entraîné(s)
Voilure tournante


Non convertible (Configuration fixe)
Deux rotors principaux
Rotor de sustentation entraîné par le mât
Coaxiaux
Sans propulsion additionnelle
Avec propulsion additionnelle
De chaque côté
Sans propulsion additionnelle
Avec propulsion additionnelle
Tandem
Sans propulsion additionnelle
Engrenants
Coaxiaux
Ejection en bout de pale
Avec propulsion additionnelle
Sans propulsion additionnelle
De chaque côté
Tandem
Avec propulsion additionnelle
Sans propulsion additionnelle


Aerostat (ballon)
Aeronef
Aerodyne
Avec rotor(s) de sustentation libre(s) (autogyre)
N rotors principaux
Rotor de sustentation entraîné par le mât
Sans propulsion additionnelle
Ejection en bout de pale
Un rotor principal
Rotor de sustentation entraîné par le mât
Configuration fixe (pour l'anticouple)
Anticouple mobile (pour propulsion)
Rotor anticouple
Rotor d'anticouple conventionnel
Fenestron
Rotors de queue multiples
NOTAR
Avec propulsion
Avec propulsion additionnelle
Sans propulsion additionnelle
Anticouple piloté par un système de propulsion
Anticouple piloté par un système de propulsion centré avec volet
Anticouple piloté par un système de propulsion décentré


Avec rotor(s) de sustentation entrainé(s)
Hybride (voilure(s) fixe(s) + voilure(s) tournante(s))
Ejection en bout de pale
NOTAR
Avec propulsion additionnelle
Sans propulsion additionnelle
Deux rotors principaux
Rotor de sustentation entrainé par le mât
Engrenants
Coaxiaux
Avec propulsion additionnelle
Sans propulsion additionnelle
De chaque côté
Sans propulsion additionnelle
Avec propulsion additionnelle
Tandem
Ejection en bout de pale
N rotors principaux
Avec rotor(s) de sustentation libre(s) (autogyre)
Voilure fixe (avion)
Voilure tournante
Un rotor principal


tournante
Convertible (Configuration mobile)
Hybride (voilure(s) fixe(s) + voilure(s) tournante(s))
Avec rotor(s) de sustentation entrainé(s)
Un rotor principal
Deux rotors principaux
Rotor de sustentation entrainé par le mât
Inclinaison du rotor seul
Inclinaison du rotor et du moteur
Inclinaison du rotor, du moteur et de l'aile
Inclinaison du rotor et de l'aile
Avec propulsion additionnelle
Sans propulsion additionnelle
Avec sustentation additionnelle
Sans sustentation additionnelle

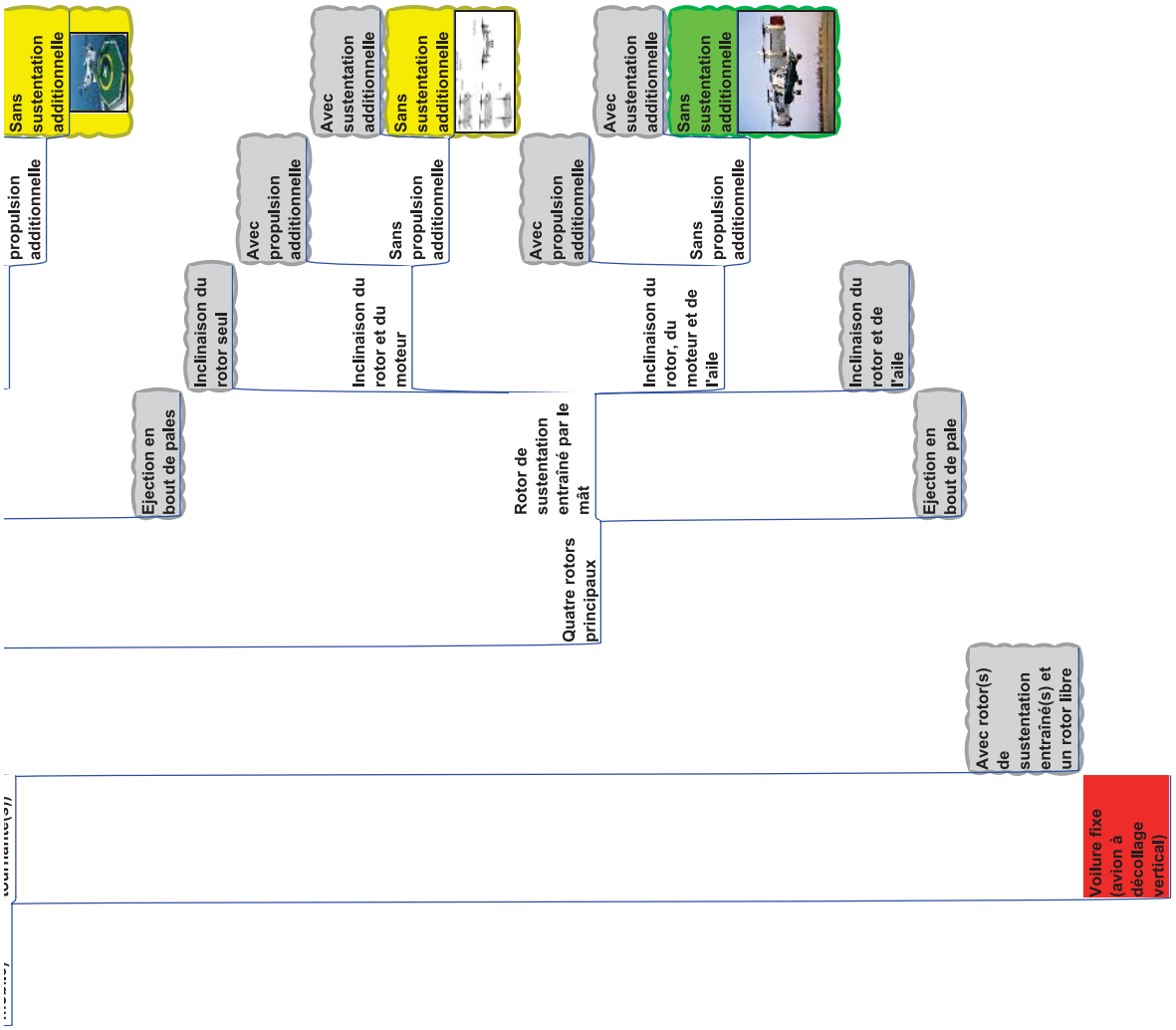


FIGURE A.1 – Synthèse des architectures hélicoptères.

A.1.2 Types de ports utilisés pour l'exemple du drone

Plusieurs types de ports sont utilisés dans l'exemple fil rouge de ce manuscrit. Les trois figures A.2, A.3 et A.4 affichent les types utilisés par classe. Seules, les feuilles des graphiques sont considérées par le logiciel de synthèse automatisée. Les autres nœuds sont uniquement organisationnels.

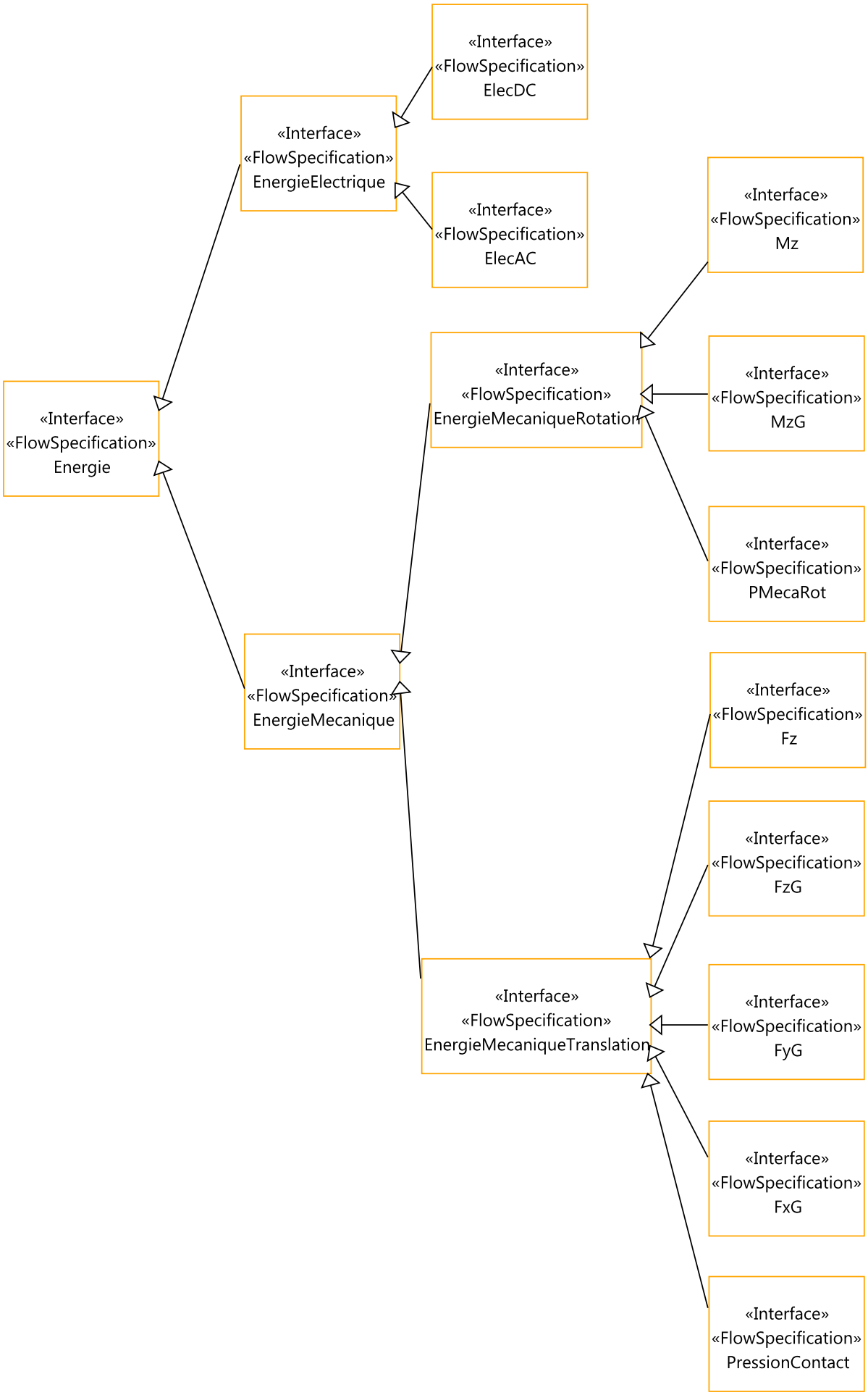


FIGURE A.2 – Ports de type énergie utilisés pour la synthèse.

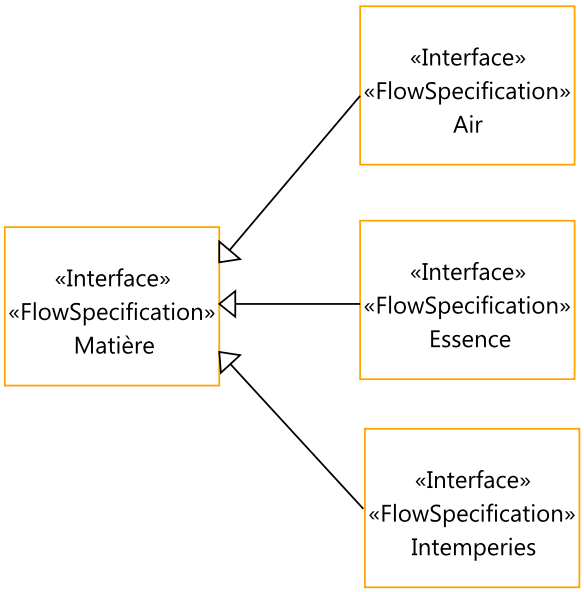


FIGURE A.3 – Ports de type matière utilisés pour la synthèse.

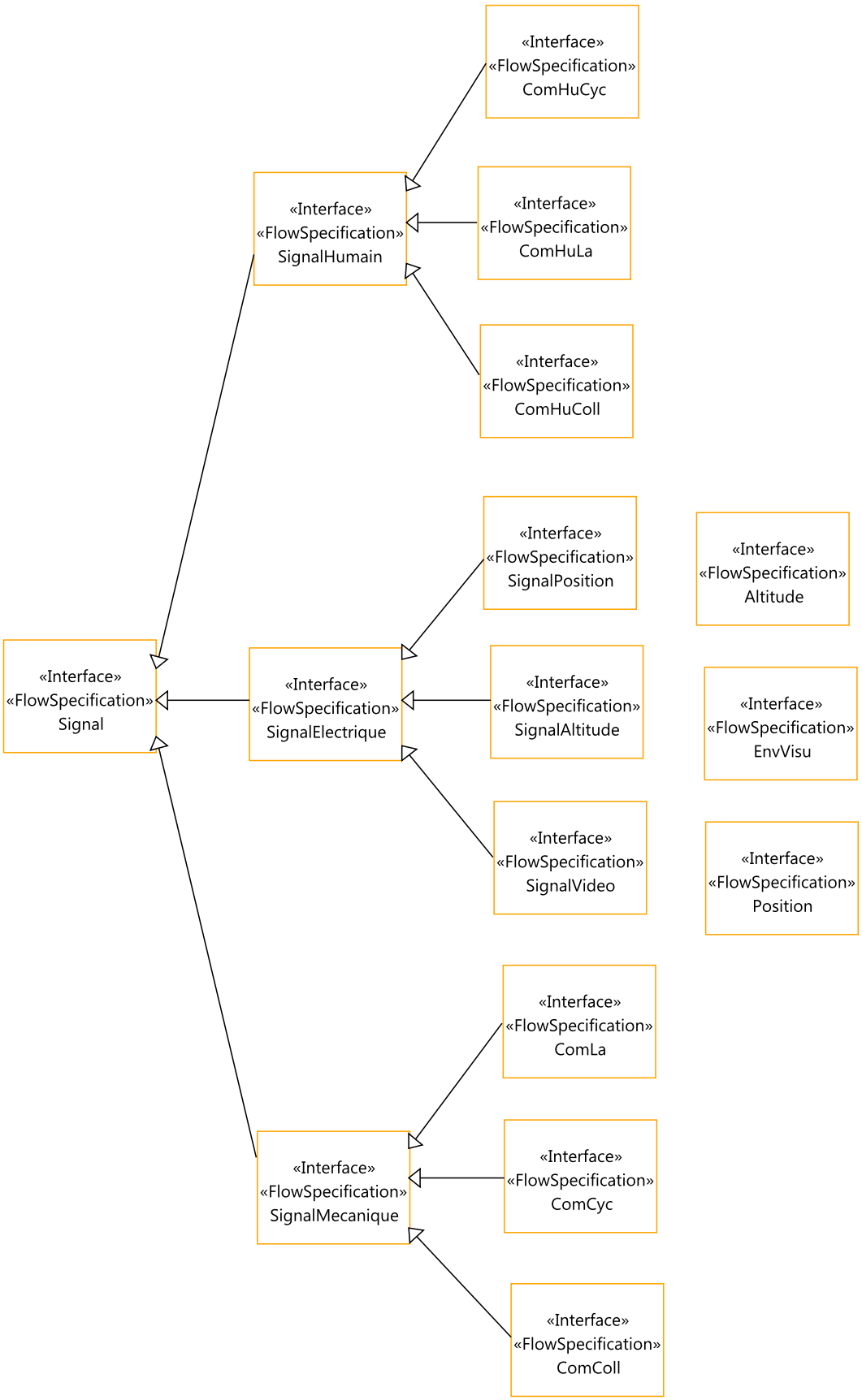


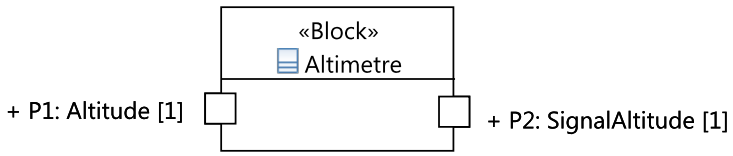
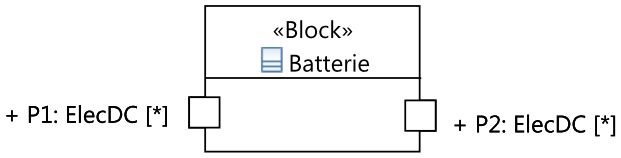
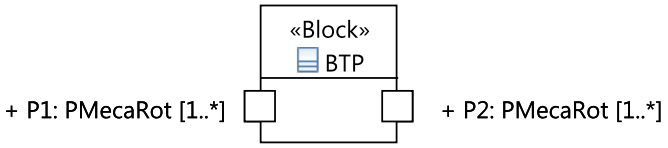
FIGURE A.4 – Ports de type signal utilisés pour la synthèse.

A.2 Tableaux annexes

A.2.1 AO logiques composant la base de données technologique

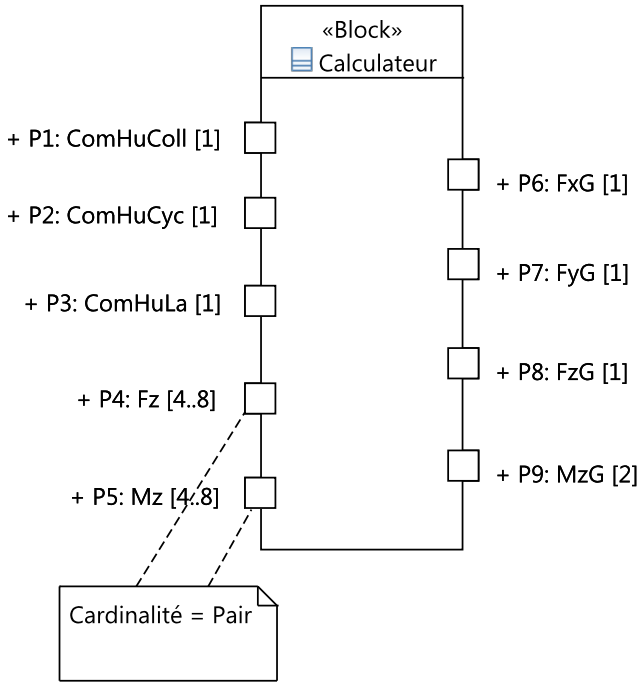
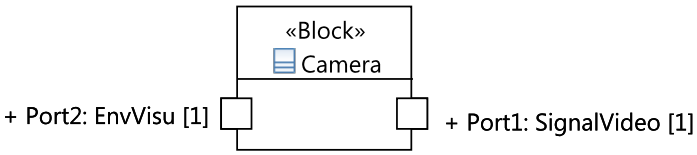
Dans le cadre de l'exemple fil rouge, les composants utilisés sont décrits sur le Tableau A.1. A noter que tous les composants consommant peu d'énergie électrique, n'ont pas de port d'entrée électrique, de façon à simplifier les modèles.

TABLEAU A.1 – Base de données des AO

Description	AO
L'altimètre reçoit, en entrée, l'altitude réelle et transmet l'information sous la forme d'un signal électrique.	
La batterie est un AO de stockage, elle reçoit et fournit donc le même flux en entrée et en sortie.	
La BTP est un réducteur permettant la réduction de la vitesse de rotation de ses arbres d'entrée jusqu'aux arbres de sortie. Dans l'exemple ci-contre, cette BTP peut recevoir plusieurs énergies mécaniques de rotation.	

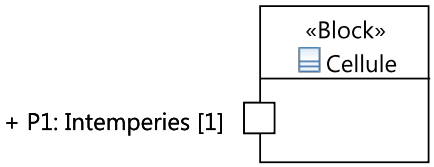
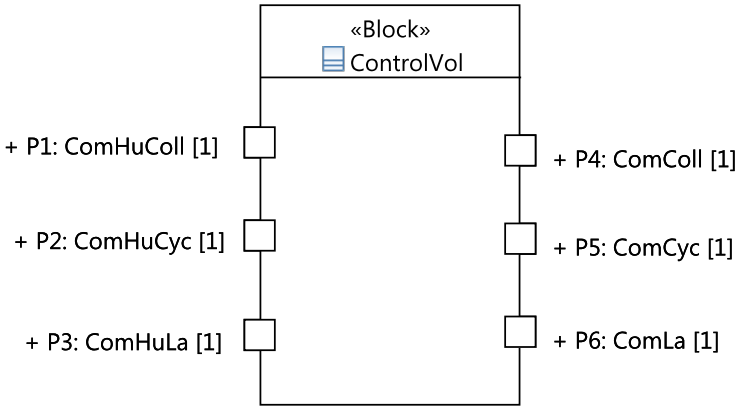
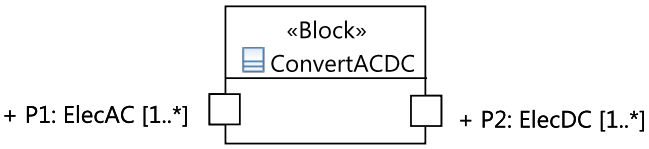
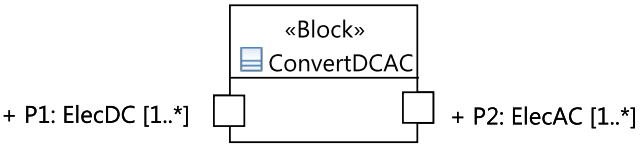
continue sur la page suivante

continue depuis la page précédente

Description	**AO**
Le calculateur est sous-entendu pour le contrôle d'aéronef multirotor avec un nombre de rotors pair. Il prend, en entrée, les commandes provenant du pilote, ainsi que la présence de quatre à huit fournisseurs de portance, et donne, en sortie, les efforts nécessaires pour mouvoir l'aéronef. La définition de cette boîte n'est pas vraiment physique. En effet, le calculateur ne reçoit pas directement les efforts de portance de chaque rotor pour fournir l'effort de portance global à la structure, au centre de gravité, mais il s'agit de la façon la plus aisée de représenter sous forme de boîte noire la fonction du calculateur. Physiquement, il s'agit d'un composant électronique qui va réguler, soit un moteur électrique, soit une turbine. La définition de ses ports peut faire penser aux classes abstraites du Dr. Helms (TUM – Allemagne).	
La caméra reçoit, en entrée, l'environnement visuel en direct pour fournir un signal vidéo.	

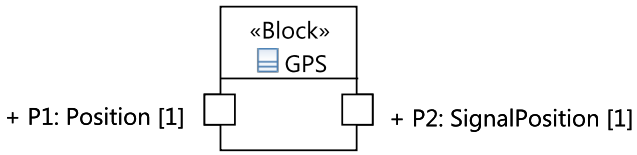
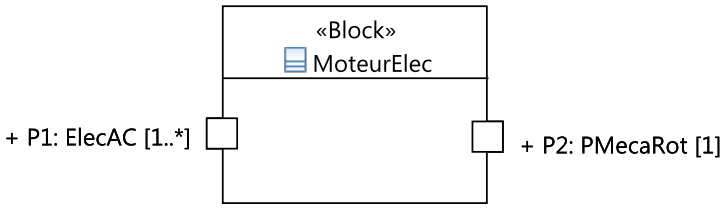
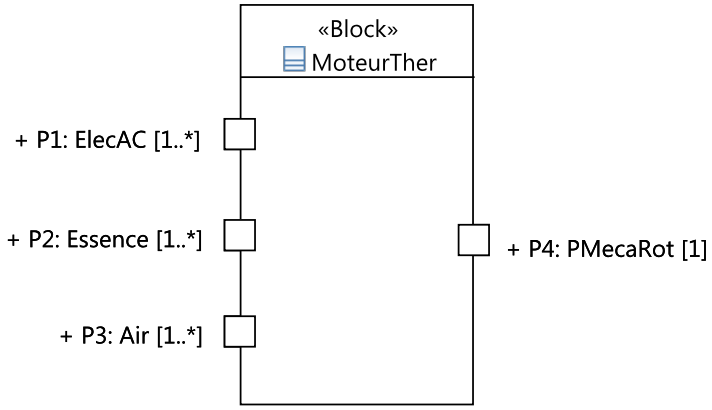
continue sur la page suivante

continue depuis la page précédente

Description	AO
La cellule est ajoutée dans le modèle car une exigence de contrainte avait émergé pour imposer son utilisation. L'ajout de cet AO a été fait en introduisant un port «Intempéries» qui correspond à l'entrée de la cellule.	
L'AO ControlVol représente sur un hélicoptère la partie commande de vol jusqu'à la sortie des plateaux de commande. Il prend, en entrée, les commandes provenant de l'utilisateur et donne, en sortie, les informations de pilotage directement au système qui va fournir la portance.	
Le convertisseur AC/DC convertit un courant alternatif en un courant continu.	
Le convertisseur DC/AC convertit un courant continu en un courant alternatif.	

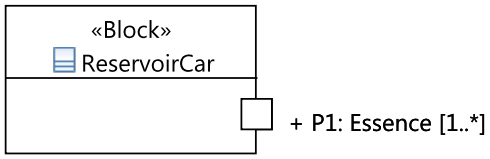
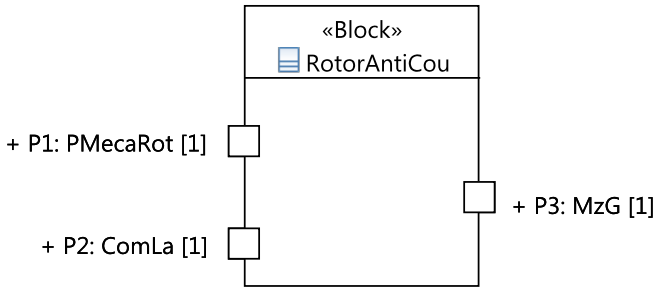
continue sur la page suivante

continue depuis la page précédente

Description	AO
Le GPS reçoit, en entrée, une position réelle et fournit, en sortie, un signal de position. A proprement parlé, il s'agit de deux valeurs : latitude et longitude, l'altitude étant mesurée par l'altimètre.	
Le moteur électrique reçoit une puissance électrique d'un courant continu et la transforme en une pièce mécanique de rotation associée à un couple et une vitesse.	
Le moteur thermique reçoit, en entrée, une puissance électrique permettant de le démarrer. C'est-à-dire que dans cette modélisation, le starter électrique est agrégé avec le moteur. Le moteur reçoit de l'essence et fournit une puissance mécanique de rotation. Un port «Air» a été rajouté, mais il n'est, a priori, pas forcément utile. Mais il est laissé pour l'exemple.	

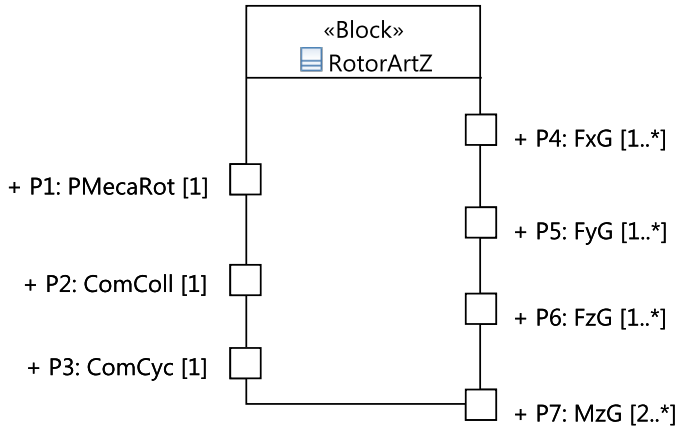
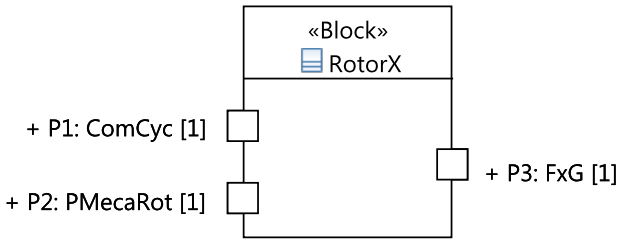
continue sur la page suivante

continue depuis la page précédente

Description	**AO**
Le réservoir de carburant est un distributeur. On remarque que l'action de remplissage du réservoir n'est pas prise en compte dans cette modélisation. Le point important concernant la situation de vie «remplissage du réservoir» est surtout de permettre géométriquement l'accès au goulot de remplissage. Ces contraintes ne sont pas traitées dans ce manuscrit.	
Le rotor anticouple est un rotor placé sur la queue de la machine, suivant l'axe de roulis de l'aéronef. Il reçoit une énergie mécanique de rotation et son rôle est de fournir un anticouple. Bien entendu, il fournit aussi une force suivant l'axe de tangage, mais pour une modélisation à notre niveau, où les angles d'orientation exacts des rotors ne sont pas étudiés, cette information est superflue.	

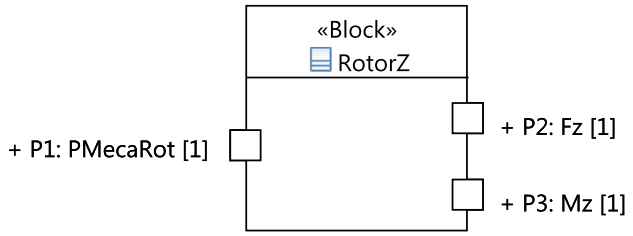
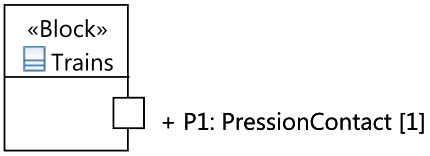
continue sur la page suivante

continue depuis la page précédente

Description	**AO**
Le rotor articulé reçoit les commandes mécaniques cyclique et collectif provenant du système de commande, ainsi qu'une puissance mécanique de rotation, et donne à la structure les efforts nécessaires pour se déplacer dans les airs. Ce rotor est orienté suivant l'axe de lacet.	
Le rotor de poussée est une hélice orientée suivant l'axe de roulis. Il permet la propulsion de l'aéronef. Il reçoit une commande cyclique et pas de collectif. c'est-à-dire que l'hélice ne produira pas de portance. Il reçoit bien entendu une puissance mécanique de rotation.	

continue sur la page suivante

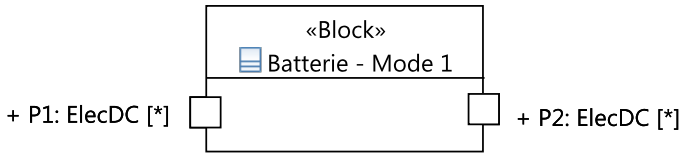
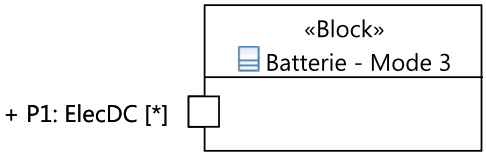
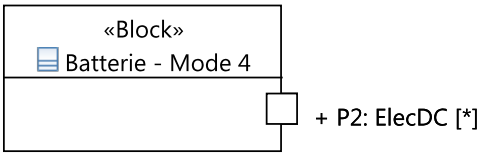
continue depuis la page précédente

Description	AO
Ce rotor est orienté suivant l'axe de lacet et n'est pas articulé. Cela veut dire que, seul, il ne peut pas produire autre chose que de la portance. L'AO calculateur est important, justement pour modérer les portances des différents rotors non articulés permettant la création des efforts de propulsion. La vitesse de rotation constitutive du flux est modulée pour générer plus ou moins de portance et forcément de couple.	
Le train permet la liaison au sol. Il impose une pression de contact sur la piste ou n'importe quel terrain sur lequel l'aéronef se pose.	

A.2.2 Modes des AO logiques composant la base de données technologique

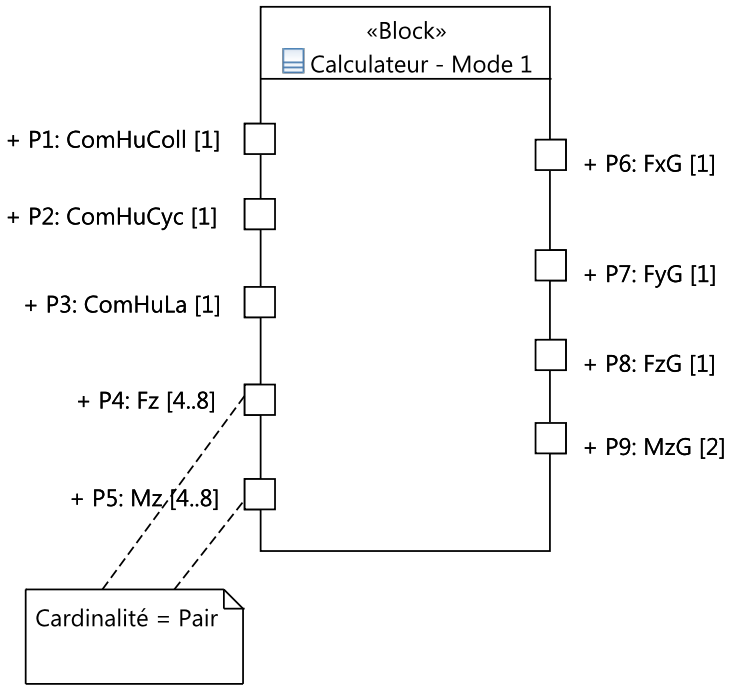
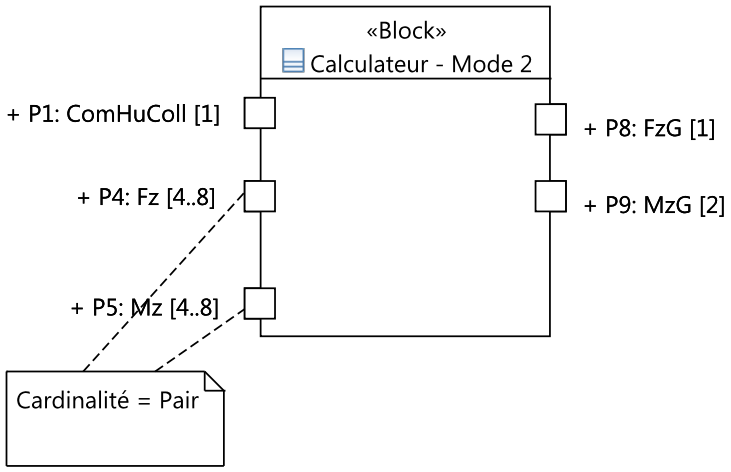
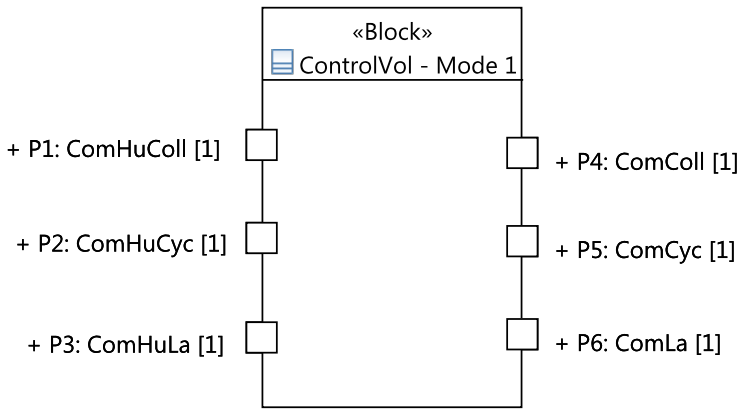
Dans le cadre de l'exemple fil rouge, les composants utilisés sont décrits sur le Tableau A.1. Or, il est aussi nécessaire de déterminer quels sont les ports actifs de ces AO suivant leurs modes de fonctionnement. Ces modes sont illustrés dans le Tableau A.2.

TABLEAU A.2 – Modes des AO

Description	AO
Le premier mode de la batterie est un mode de rechargement et de débit en même temps. Il s'agit d'un mode très courant en aéronautique, où l'ensemble de l'aéronef est alimenté par les batteries pendant que ces dernières se rechargent à l'aide de génératrices connectées aux moteurs.	
Le second mode de la batterie correspond au mode totalement désactivé. Il n'est pas traité. Le troisième mode de la batterie est un mode de rechargement.	
Le quatrième mode de la batterie est un mode de déchargement.	

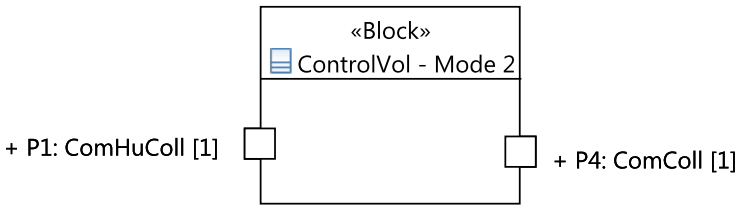
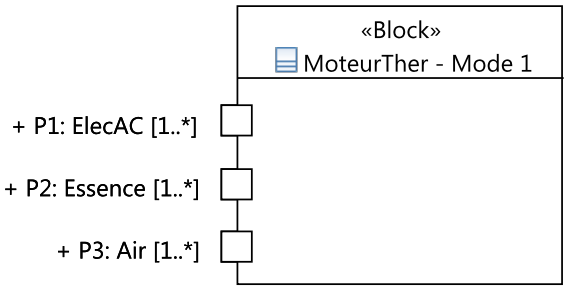
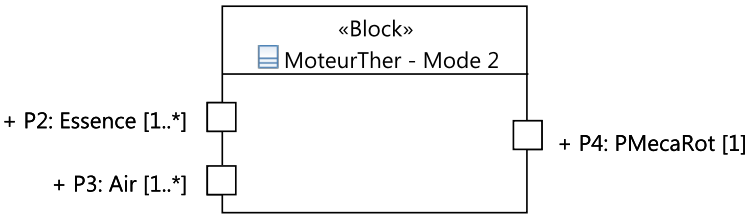
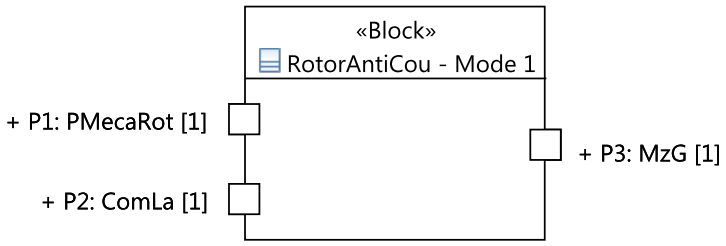
continue sur la page suivante

continue depuis la page précédente

Description	AO
Le premier mode du calculateur correspond à un mode de vol d'avancement. Toutes les commandes peuvent être reçues et toutes les directions peuvent être prises.	
Le second mode du calculateur correspond à un mode de vol stationnaire ou de montée/descente de l'aéronef.	
Le premier mode des commandes de vol correspond à un mode de vol d'avancement. Toutes les commandes peuvent être reçues et toutes les directions peuvent être prises.	

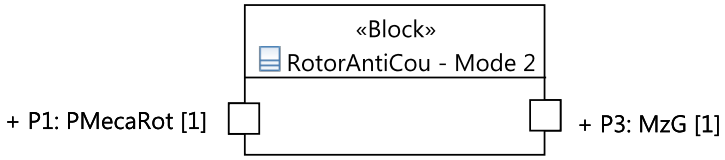
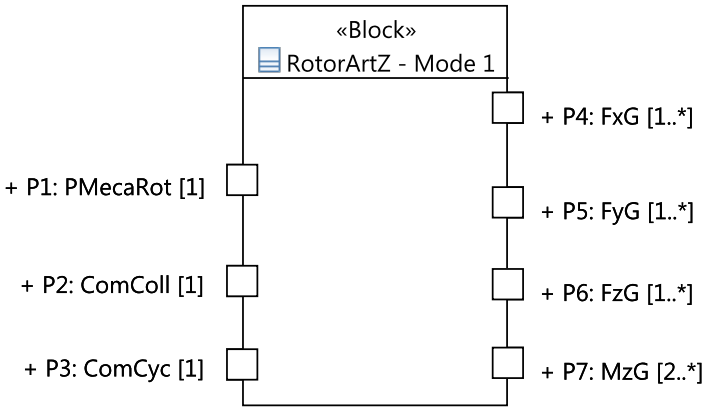
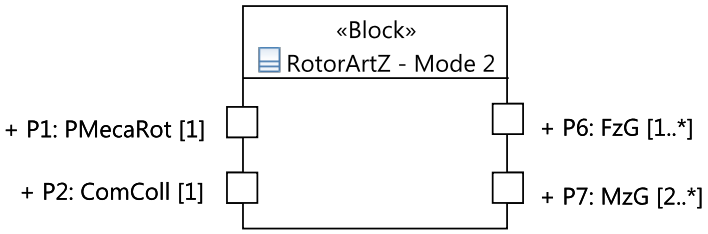
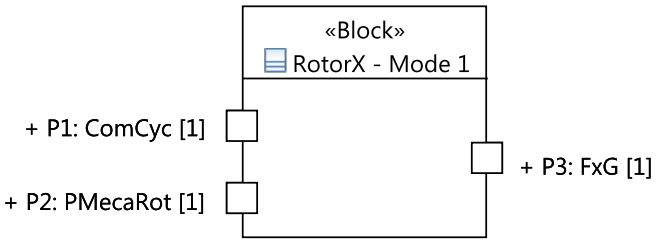
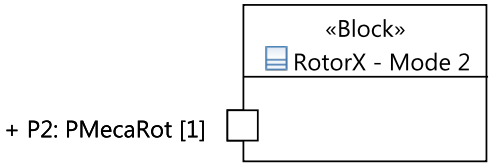
continue sur la page suivante

continue depuis la page précédente

Description	AO
Le second mode des commandes de vol correspond à un mode de vol stationnaire ou de montée/descente de l'aéronef.	
Le premier mode du moteur thermique correspond au démarrage. A ce moment, le starter reçoit de l'énergie électrique et du carburant, mais ne produit pas encore une puissance mécanique utilisable. Le mode de transition entre démarrage et utilisation nominale n'est pas étudié.	
Le second mode du moteur thermique correspond au cas nominal avec une production stabilisée de la puissance mécanique de rotation.	
Le premier mode du rotor anticouple correspond à une utilisation où du pas est appliqué aux pales de ce rotor. Du pas est appliqué lorsque le couple transmis du rotor principal à la structure augmente ou lorsque l'aéronef doit être commandé en lacet.	

continue sur la page suivante

continue depuis la page précédente

Description	AO
Le second mode du rotor anticouple correspond à la production d'un anticouple suffisant pour équilibrer l'aéronef, mais pas pour de la commande en lacet.	
Le premier mode du rotor articulé correspond à un vol d'avancement.	
Le second mode du rotor articulé correspond à un vol stationnaire.	
Le premier mode du rotor orienté dans l'axe de roulis correspond à un vol d'avancement.	
Le second mode du rotor orienté dans l'axe de roulis correspond à un vol stationnaire.	

A.2.3 Définition physique des variables d'échange de la base de données des AO

Tous les ports de la base de données des AO utilisée dans l'exemple fil rouge de ce manuscrit sont définis dans le tableau A.3.

TABLEAU A.3 – Définition physique des ports de tous les AO de la base de données.

Variable	Unité	Borne mini	Borne max	Contraintes associées à chaque mode		Est associative ?
AO RotorArtZ						
Port PMecaRot						
PMeca	kW	0	500	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui
Omega	rpm	100	400	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	—
Couple	Nm	0	50000	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui
Port FxG						
FxG	N	0	100000	Mode 1 Mode 2 Mode 3	∅ = 0 = 0	Oui
PxG	kW	0	500	Mode 1 Mode 2 Mode 3	∅ = 0 = 0	Oui
Port FzG						
FzG	N	0	100000	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
PzG	kW	0	500	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
Port MzG						
MzG	Nm	0	2000	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
AO RotorAntiCou						
Port PMecaRot						
PMeca	kW	0	500	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui
Omega	rpm	800	3200	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	—

continue sur la page suivante

continue depuis la page précédente

Variable	Unité	Borne mini	Borne max	Contraintes associées à chaque mode		Est associative ?
Couple	Nm	0	7000	Mode 1 Mode 2 Mode 3	∅ ∅ ∅	Oui
Port MzG						
MzG	Nm	-50000	0	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
AO Calculateur						
Port Fz						
Fz	N	0	100000	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
Pz	kW	0	500	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
Port Mz						
Mz	Nm	0	50000	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
Port FxG						
FxG	N	0	100000	Mode 1 Mode 2 Mode 3	∅ = 0 = 0	Oui
PxG	kW	0	500	Mode 1 Mode 2 Mode 3	∅ = 0 = 0	Oui
Port FzG						
FzG	N	0	100000	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
PzG	kW	0	500	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
Port MzG						
MzG	Nm	0	50000	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
AO RotorZ						
Port PMecaRot						
PMeca	kW	0	500	Mode 1 Mode 2	∅ ∅	Oui
Omega	rpm	100	400	Mode 1 Mode 2	∅ ∅	—

continue sur la page suivante

continue depuis la page précédente

Variable	Unité	Borne mini	Borne max	Contraintes associées à chaque mode		Est associative ?
Couple	Nm	0	50000	Mode 1 Mode 2	∅ ∅	Oui
Port Fz						
Fz	N	0	100000	Mode 1 Mode 2	∅ = 0	Oui
Pz	kW	0	500	Mode 1 Mode 2	∅ = 0	Oui
Port Mz						
Mz	Nm	0	50000	Mode 1 Mode 2	∅ = 0	Oui
AO RotorX						
Port PMecaRot						
PMeca	kW	0	500	Mode 1 Mode 2 Mode 3	∅ = 0 = 0	Oui
Omega	rpm	100	400	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	—
Couple	Nm	0	50000	Mode 1 Mode 2 Mode 3	∅ = 0 = 0	Oui
Port FxG						
FxG	N	0	100000	Mode 1 Mode 2 Mode 3	∅ = 0 = 0	Oui
PxG	kW	0	500	Mode 1 Mode 2 Mode 3	∅ = 0 = 0	Oui
AO MoteurTher						
Port ElecAC						
PElecAC	kW	0	500	Mode 1 Mode 2 Mode 3	∅ = 0 = 0	Oui
Port Essence						
Qc	kg/h	0	100000	Mode 1 Mode 2 Mode 3	∅ ∅ = 0	Oui
Port PMecaRot						
PMeca	kW	0	500	Mode 1 Mode 2 Mode 3	= 0 ∅ = 0	Oui

continue sur la page suivante

continue depuis la page précédente

Variable	Unité	Borne mini	Borne max	Contraintes associées à chaque mode		Est associative ?
Omega	rpm	100	400	Mode 1 Mode 2 Mode 3	= 0 ∅ = 0	—
Couple	Nm	0	50000	Mode 1 Mode 2 Mode 3	= 0 ∅ = 0	Oui
AO ReservoirCar						
Port Essence						
Qc	kg/h	0	100000	Mode 1 Mode 2	∅ = 0	Oui
AO MoteurElec						
Port ElecAC						
PElecAC	kW	0	500	Mode 1 Mode 2	∅ = 0	+
Port PMecaRot						
PMeca	kW	0	500	Mode 1 Mode 2	∅ = 0	Oui
Omega	rpm	100	400	Mode 1 Mode 2	∅ = 0	—
Couple	Nm	0	50000	Mode 1 Mode 2	∅ = 0	Oui
AO Batterie						
Port ElecDC						
PElecDC	kW	0	500	Mode 1 Mode 2 Mode 3 Mode 4	∅ = 0 ∅ = 0	Oui
Port ElecAC						
PElecAC	kW	0	500	Mode 1 Mode 2 Mode 3 Mode 4	∅ = 0 = 0 ∅	Oui
AO ConvertDCAC						
Port ElecDC						
PElecDC	kW	0	500	Mode 1 Mode 2	∅ = 0	Oui
Port ElecAC						
PElecAC	kW	0	500	Mode 1 Mode 2	∅ = 0	Oui
AO ConvertACDC						
Port ElecAC						
PElecAC	kW	0	500	Mode 1 Mode 2	∅ = 0	Oui

continue sur la page suivante

continue depuis la page précédente

Variable	Unité	Borne mini	Borne max	Contraintes associées à chaque mode		Est associative ?
Port ElecDC						
PElecDC	kW	0	500	Mode 1 Mode 2	∅ = 0	Oui
AO BTP						
Port PMecaRot						
PMeca	kW	0	500	Mode 1 Mode 2	∅ = 0	Oui
Omega	rpm	100	400	Mode 1 Mode 2	∅ = 0	—
Couple	Nm	0	50000	Mode 1 Mode 2	∅ = 0	Oui
Port PMecaRot						
PMeca	kW	0	500	Mode 1 Mode 2	∅ = 0	Oui
Omega	rpm	100	400	Mode 1 Mode 2	∅ = 0	—
Couple	Nm	0	50000	Mode 1 Mode 2	∅ = 0	Oui
AO Trains						
Port PressionContact						
P	MPa	0	5	Mode 1 Mode 2	∅ = 0	Oui