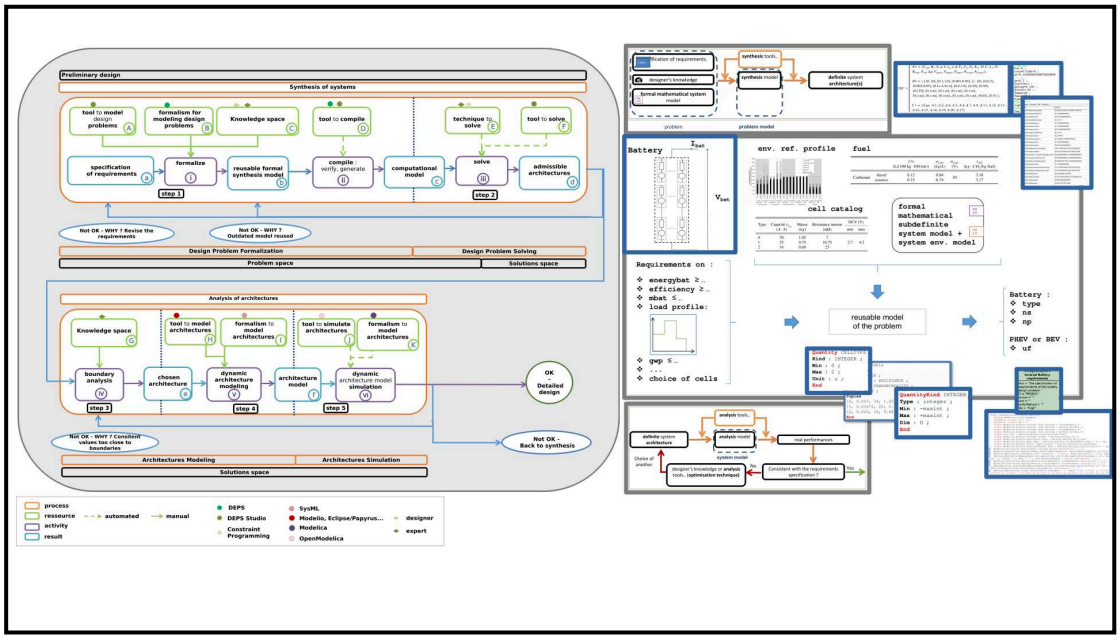
Modèles de synthèse pour la conception en génie électrique : application à l'électrification des véhicules

Sephora Diampovesa

Par **Sephora DIAMPOVESA**

Modèles de synthèse pour la conception en génie électrique : application à l'électrification des véhicules

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 10 juin 2021
Spécialité : Génie Électrique : Unité de recherche en Mécanique - Laboratoire Roberval

D2625

Modèles de synthèse pour la conception en génie électrique : application à l'électrification des véhicules

Spécialité : Génie Electrique

par Sephora DIAMPOVESA

Département Ingénierie Mécanique
Université de Technologie de Compiègne

Thèse de doctorat soumise pour l'obtention du grade
de *Docteure en Génie Electrique*

Soutenue le 10 juin 2021

Direction de la thèse : M. Arnaud HUBERT (Directeur), Professeur UTC
M. Pierre-Alain YVARS (Co-directeur), Professeur ISAE-Supméca

Jury : M. Jean BIGEON (Rapporteur), Directeur de recherche CNRS
M. Stéphane BRISSET (Rapporteur), Professeur Ecole centrale de Lille
Mme Manuela SÉCHILARIU (Examinatrice), Professeure UTC
Mme Elise VAREILLES (Examinatrice), Professeure ISAE-SUPAERO
M. Laurent ZIMMER (Invité), Ingénieur de recherche Dassault Aviation

Unité de recherche Roberval

Remerciements

Les travaux de recherche conduits durant ma thèse ont été réalisés au sein de l'unité de recherche en Mécatronique, Énergie, Électricité et Intégration (M2EI) de l'unité de recherche Roberval-CNRS, Département Ingénierie Mécanique de l'Université de Technologie de Compiègne (UTC).

Je remercie M. Pierre-Alain YVARS et M. Arnaud HUBERT, mes directeurs de thèse, pour leur encadrement continu et leurs conseils tout au long de ces années de travail, en particulier durant la rédaction de mon manuscrit et la préparation de ma soutenance. Je les remercie également pour m'avoir introduite à la recherche scientifique et respectivement pour leurs accueils au sein de ISAE-Supméca pour une formation et de l'équipe M2EI. Merci à M. Laurent Zimmer, ingénieur de recherche chez Dassault Aviation, et M. Pierre-Alain YVARS pour m'avoir permis d'utiliser les solveurs Constraint Explorer (CE), propriété de Dassault Aviation et DEPS Studio.

Je remercie toutes les personnes, membres de l'équipe, du laboratoire, de l'école doctorale et des ressources humaines qui ont participé d'une manière ou d'une autre à la bonne réussite à termes de ma thèse. Je remercie la région Hauts-de-France et le Fonds Européen de Développement Économique et Régional (FEDER) pour cette allocation doctorale relative au projet HetSpec.

Je remercie M. Jean BIGEON, directeur de recherche au CNRS et M. Stéphane BRISSET, professeur de l'Ecole centrale de Lille pour avoir acceptés d'être rapporteurs sur mes travaux. Je remercie M. Laurent ZIMMER, Mme Manuela SÉCHILARIU, professeure à l'UTC, Mme Elise VAREILLES, professeure à l'ISAE-SUPAERO pour avoir accepté d'examiner mes travaux lors de ma soutenance.

J'adresse ces derniers mots de remerciements à toute ma famille, mes parents et mes frères et sœurs pour leur patience, leur positivité sans faille et leur soutien inconditionnel. Un grand merci tout particulièrement à ma maman Jeannine Bazolélé à qui je dédie tous mes travaux de thèse dont ce manuscrit.

Abstract

The IEEE 1220 standard mainly recommands the use of analysis tools for designing new products, throughout the simulation of system architectures. However, limitations appear for obtaining new solutions as analysis rely on existing architectures designed according to a particular specification of requirements and which structures are usually not modified. The design will mainly consists in adapting their dimensions. The variability inherent to configuration and architectures generation problems cannot be specified naturally. Design problems in electrical engineering usually include mathematical aspects that are usually hard to deal with for solvers, such as mixed-type variables or catalog constraints and that are not properly considered by analysis tools most of the time. Non-functional requirements such as safety or eco-design are also usually not addressed in one design step through analysis. Thus, design is lengthened by additional analysis processes. The ever-increasing complexity of new systems necessitate to rethink their design approaches during preliminary design by enhancing architectures conceptualization, through synthesis. This thesis presents an equipped and operational approach for the preliminary design of systems in the framework of systems engineering by relying mainly on synthesis. This approach deals in particular with non-linear, mixed-type sizing, configuration and architecture generation problems with catalogs, with functional and non-functional requirements (including those of environmental type) of structured and decomposable systems. It also relies on the use of white box model to describe the system. It is completed by a validation step through analysis to reduce furthermore the space of admissible solutions. This approach focuses on the modeling of design problems in electrical engineering and of their solving by including non-functional requirements right from the start through a single process. Starting from the specification of the requirements, the proposed approach develops the construction of synthesis models of design problems and their solvings using the DEPS problem modeling language and the DEPS Studio environment, under development (www.depslink.com), based on Constraint Programming and Object-oriented modeling. Concepts related to this approach include in particular the issue of reusability of models, notions of subdefinite systems and of problem, knowledge and solutions spaces. Several design examples of electrical systems as electrical machines and of a Li-ion battery for Electric Vehicle act as practical study cases for this thesis.

Keywords : Preliminary design, subdefinite system, variability, design problems formalization, synthesis model, constraint programming, problem-space oriented approach, reusability, electrical systems.

Résumé

La norme IEEE 1220 recommande principalement l'utilisation d'outils d'analyse pour la conception de nouveaux produits à travers la simulation d'architectures de systèmes. Cependant, des limitations apparaissent pour l'obtention de nouvelles solutions, car l'analyse repose sur des architectures existantes conçues selon une spécification particulière des exigences et dont les structures ne sont généralement pas modifiées. La conception consistera principalement à adapter leurs dimensions. La variabilité inhérente aux problèmes de configuration et aux problèmes de génération d'architectures ne peut être spécifiée naturellement. Les problèmes de conception incluent généralement des aspects mathématiques difficiles à traiter par les solveurs, tels que les variables mixtes ou l'utilisation de catalogue et qui ne sont pas correctement pris en compte par les outils d'analyse. Les exigences non-fonctionnelles, telles que la sûreté de fonctionnement ou l'éco-conception, ne sont généralement pas non plus traitées en une seule étape avec l'analyse. Ainsi, la conception est allongée par des processus d'analyse supplémentaires. La complexité sans cesse croissante des nouveaux systèmes nécessite de repenser leurs approches de conception pendant la conception préliminaire en améliorant la conceptualisation des architectures, à travers la synthèse. Cette thèse présente une approche outillée et fonctionnelle pour la conception préliminaire de systèmes physiques en Génie Electrique dans le cadre de l'Ingénierie des Systèmes s'appuyant principalement sur la synthèse. Cette approche traite en particulier des problèmes non-linéaires, mixtes de dimensionnement, configuration et de génération d'architectures, à exigences fonctionnelles et non-fonctionnelles (notamment de type environnementales) avec catalogue, de systèmes structurés et décomposables. Elle repose également sur l'utilisation de modèles de type boîte blanche pour décrire le système. Elle est complétée par une étape de validation par analyse pour réduire encore l'espace des solutions admissibles. Elle se concentre sur la modélisation de problèmes de conception en Génie Electrique et sur leurs résolutions, en incluant dès le départ des exigences non-fonctionnelles au sein d'un processus unique. En partant de la spécification des exigences, l'approche proposée développe la construction de modèles de synthèse de problèmes de conception et leurs résolutions en utilisant le langage de modélisation de problème DEPS et l'environnement DEPS Studio, en cours de développement (www.depslink.com) et basés sur la Programmation Par Contraintes et la Modélisation orientée objet. Les concepts liés à cette approche incluent en particulier la question de la réutilisabilité des modèles, les notions de sous-définition, d'intégration de modèles, de systèmes sous-définis et de celles d'espaces du problème, des connaissances et des solutions. Plusieurs exemples de conception de systèmes électriques, tels que des machines électriques et une batterie Li-ion pour Véhicule Electrique servent de cas d'étude pratique pour cette thèse.

Mots-clés : Conception préliminaire, systèmes sous-définis, variabilité, formalisation de problèmes de conception, modèle de synthèse, d'architectures, programmation par contraintes, approche orientée problème, réutilisabilité, systèmes électriques.

Table des matières

Liste des acronymes

Introduction générale

Contexte des travaux de thèse

La conception des systèmes physiques complexes constitue un défi actuel important et en constante évolution. La concurrence économique très importante entre les différents acteurs industriels pousse ceux-ci à proposer régulièrement de nouveaux dispositifs toujours plus innovants. Le cahier des charges exprime le problème de conception d'un système et est établi à partir des besoins clients puis validé par l'ingénierie. Un problème de conception mène à de nombreuses interrogations avant, pendant et après la conception, à savoir : que cherche-t-on à construire ? Que doit réaliser le système ? Et de quelle manière ? A-t-on réalisé le bon système ? La résolution du problème de conception de ce système doit permettre de répondre à ces questions.

Les ingénieurs des bureaux d'études se doivent de satisfaire des cahiers des charges à complexité croissante et ce dans des délais toujours plus courts. À la complexité intrinsèque des systèmes à concevoir s'ajoute souvent la gestion de cahiers des charges aux exigences nombreuses et parfois contradictoires. Cette difficulté se matérialise en termes de performances, de qualité, d'intégration, de réduction des coûts / des déchets / de la pollution ou encore de sûreté ou de consommation énergétique. L'ingénierie se doit de gérer des exigences très hétérogènes, relatives aussi bien à son fonctionnement qu'à ses performances. A ces exigences s'ajoutent de plus en plus des aspects que l'on peut qualifier de non-fonctionnels et sociétaux, comme environnementaux ou humains [1]. La conception des systèmes physiques nécessite la synergie de plusieurs disciplines et compétences spécifiques. Le plus souvent, de forts couplages entre domaines physiques à savoir électrique, thermique, magnétique ou chimique s'ajoutent également.

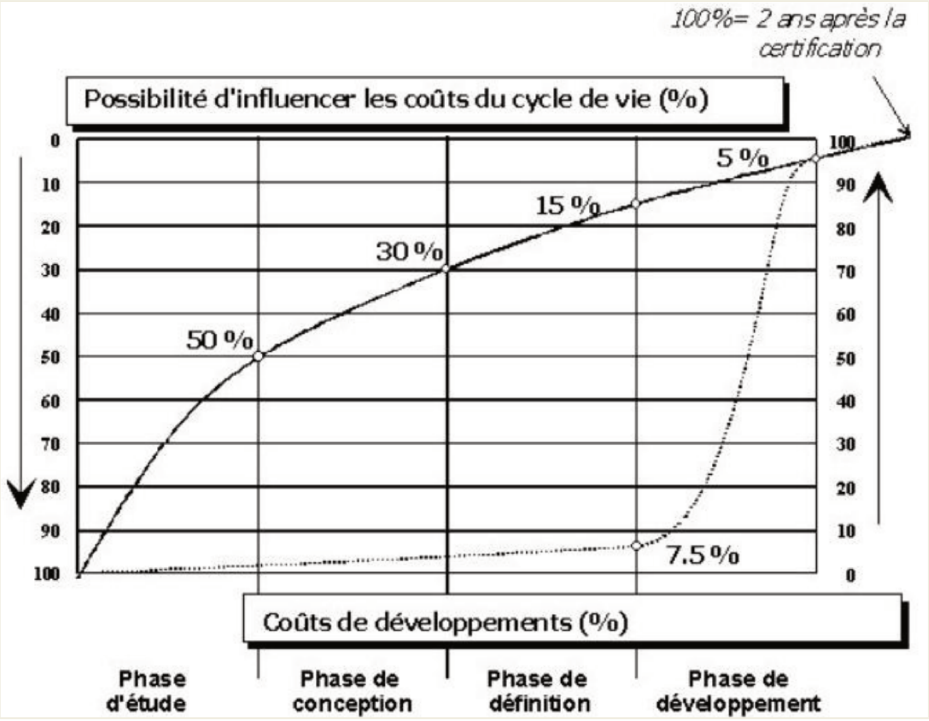
Pour répondre à la problématique de la conception, les concepteurs s'aideront de différents outils logiciels, techniques et langages qui permettront de la résoudre.

C'est lors des premières étapes de la conception que les architectures d'un système sont produites et que les principaux choix technologiques sont réalisés. Nous parlerons d'architectures pour désigner un système dont la structure est dimensionnée et la topologie connue, ie. son type, nombre de composants et leur agencement est connu ainsi que les dimensions du système. Les premières dimensions d'une architecture peuvent ou non déjà être fixées à ce stade. Ces premières étapes font référence à la conception préliminaire. Face à tous ces défis, les outils aidant à la conception préliminaire sont ainsi devenus primordiaux.

Plusieurs questions peuvent alors se poser. Par exemple, comment obtient-on actuellement des solutions technologiques en ingénierie ? De quelles aides disposent les concepteurs pour concevoir ? Ou encore, à quels types de problèmes de conception de systèmes a-t-on généralement à faire ? Les réponses à la plupart de ces questions dépendront fortement de l'approche, des langages et des outils utilisés lors de la conception préliminaire. D'une manière générale en GE, la conception préliminaire est très outillée lorsque elle est employée avec une approche d'analyse, mais très peu peu outillée avec une approche de synthèse [2]. Il s'agit pourtant des phases les plus décisives, comme l'indique la Fig. 1 issue d'une étude dans le domaine de l'aéronautique (où la "phase d'étude" et le début de la "phase de conception" peuvent être vues comme de la conception préliminaire). Celles-ci influent à près de 70% sur les coûts de fabrication globale d'un système durant son cycle de vie, bien qu'elles ne représentent qu'une faible part des coûts réels. La conception préliminaire est ainsi une étape clé aux intérêts économiques et stratégiques importants. L'intérêt serait d'orienter au plus tôt la conception vers les architectures admissibles, relativement au cahier des charges, en filtrant dès le départ celles qui n'y correspondent pas.

Pour les industriels du transport et de l'énergie en particulier, la plupart des outils développés pour la conception reposent principalement sur l'amélioration de solutions technologiques existantes [2]. Une telle approche de conception fait appel à des modèles dits d'analyse ou de simulation. On retrouve aussi principalement les approches utilisant des modèles d'analyse dans de nombreux travaux académiques. Toutefois, la principale limite de ces développements est de disposer d'une architecture initiale existante. Cela laisse peu de liberté de conception, puisque les composants d'un système et son agencement sont déjà fortement déterminés (la conception est alors réduite à du dimensionnement). On peut donc difficilement générer de nouvelles architectures pour le dispositif à concevoir et proposer des solutions très innovantes.

Figure 1 Influence de la conception préliminaire sur le coût global du d'un produit [3].



Objectifs des travaux de thèse

Le travail de recherche proposé dans cette thèse a pour objectif d'explorer une approche de conception de systèmes physiques outillée et intégrée, appliquée pour le GE qui ne reposerait pas sur de l'amélioration mais sur de l'obtention d'architectures, qui seraient correctes-par-construction en respectant le cahier des charges. Le processus permettant d'obtenir de telles architectures est dit de synthèse. Cette approche serait dédiée à la conception préliminaire et permettrait de prendre en compte des exigences à la fois fonctionnelles et non-fonctionnelles dès les étapes initiales de pré-conception. Cette approche vise principalement à couvrir les étapes depuis la formalisation d'un problème de pré-conception jusqu'à sa résolution. L'attention de ces travaux se portera en particulier sur l'aspect formalisation de ces problèmes. En particulier, les problèmes décrivant un choix d'architectures pour un système physique seront étudiés, car ils permettent le plus de liberté de conception et sont peu courants en Génie Electrique (GE).

De plus, ces travaux chercheront à construire des modèles de problèmes de conception dit de synthèse, qui seraient réutilisables dans d'autres projets de conception (ex : concevoir un système selon différents cahiers des charges). Egalement, cette approche explorera des

étapes d'analyse qui devrait permettre de mieux filtrer les solutions obtenues en les évaluant, notamment par simulation.

La construction de l'approche sera illustrée par des exemples d'application tirés du GE. Nous nous intéressons ainsi aux points suivants : quels concepts, formalismes, outils et techniques de résolution pourraient être utilisés pour concevoir une telle approche intégrée de conception reposant sur la synthèse ? C'est-à-dire, pour la formalisation d'une part et la résolution des problèmes d'autre part ? Pour la formalisation, quels types de modèles mathématiques seraient le plus adaptés pour représenter un système aux prémisses de la conception ? Comment aider les concepteurs à réaliser les bons choix de modélisation ? Comment s'assurer que les solutions obtenues à l'issue de cette approche respectent le cahier des charges ?

Organisation de la thèse

Ce manuscrit de thèse se compose de 4 chapitres.

Chapitre 1 Le premier chapitre est une introduction générale à la problématique de conception préliminaire des systèmes physiques complexes en ingénierie. Ce chapitre permettra de présenter les approches de conception actuelles, les techniques employées pour concevoir et les notions utiles pour la conception de systèmes physiques.

Chapitre 2 Nous proposons d'explorer la Programmation Par Contraintes (PPC) pour la conception des systèmes physiques dans le contexte de la synthèse. Nous nous intéresserons à la formalisation et la résolution de problèmes de conception du GE. Le caractère optimal d'un problème de conception mixtes sera aussi discuté. Il sera également abordée la nature des différents problèmes de conception, la gestion de contraintes contradictoires, la gestion des contraintes catalogue, l'implémentation de l'optimisation ou encore les choix de modélisation. Nous étudierons également les critères de performance et de rapidité de ces techniques en comparaison avec les outils classiques utilisés en conception optimale. La PPC sera finalement illustrée depuis la formalisation jusqu'à la résolution par trois cas d'études benchmark avec la prise en compte de catalogues de composants. Cela permettra d'aborder en pratique la formalisation et la résolution et d'en voir les limites. Le premier exemple traitera du dimensionnement optimal d'un moteur DC brushless. Le second exemple traitera de la génération d'architectures optimale d'un moteur AC/DC brushless à aimants perma-

nents (AP). Le troisième exemple traitera du dimensionnement optimal d'un transformateur de sécurité.

Chapitre 3 Ce chapitre présentera une approche structurée et réutilisable pour la conception. Elle sera principalement axée sur la formalisation des problèmes de conception. Une application liée à la conception d'une batterie de VE sera utilisée comme cas d'étude et illustration des solutions proposées. Nous proposons en particulier trois axes de modélisation pour un contrainte de choix de composants (contrainte catalogue). Des exigences non-fonctionnelles de type 2 (environnementales) sont par la suite ajoutées au problème de conception. Ce chapitre sera illustré par les cas de conception et d'éco-conception d'une batterie.

Chapitre 4 Dans ce chapitre final qui servira d'ouverture, nous présenterons l'approche globale intégrée à base de modèles orientés-objets alliant synthèse et analyse, illustrée par l'exemple de la batterie. Nous aborderons en particulier l'intégration de l'analyse pour la validation d'architectures à partir des résultats obtenus lors de la synthèse de systèmes. Les limites de l'approche et des perspectives seront explorées notamment pour aider les concepteurs en cas de difficultés de validation d'architectures (pas de solutions, modèles non valides, perte d'admissibilité des architectures).

Chapitre 1

Introduction à la conception de systèmes en GE

1.1 La conception de systèmes complexes

Ce travail de thèse prend pour cadre le Génie Electrique et plus particulièremant l'électrification des véhicules. L'électrification des véhicules est étudiée depuis de nombreuses décennies [4–8] que ce soit pour le développement de véhicules à énergie électrique embarquée ou encore à piles à combustible [5, 9–11]. Cependant, l'autonomie des systèmes de stockages énergétiques rechargeables (les réservoirs d'hydrogènes des piles à combustibles ou encore les batteries électrochimiques) reste difficile à assurer car une hausse de l'autonomie est synonyme de masse et de coûts de fabrication plus importants. Pour pouvoir rester compétitifs, les industriels doivent pouvoir fournir des systèmes innovants et performants à leurs clientèles. À ces exigences difficiles à satisfaire s'ajoutent d'autre exigences plus récentes durant la conception d'un système électrique comme la sûreté de fonctionnement ou bien encore les exigences environnementales. L'ensemble de ces contraintes fait que l'électrification des véhicules conduit bien souvent à des problématiques de conception de systèmes complexes. Avant d'entrer plus en détail dans ce domaine, il est d'abord nécessaire de définir quelques notions-clés liées au système.

1.1.1 Notions liées au système

Le point de vue adopté sur un dispositif peut varier selon le niveau de la conception considéré, typiquement au niveau intégration système (en conception préliminaire) ou conception composant (en conception détaillée). Par exemple, une batterie d'accumulateurs pour VE

peut être vu comme un système (composé de cellules) si on cherche à la concevoir de manière détaillée ou comme un composant si on s'intéresse au VE globalement. Dans le cadre de ces travaux de thèse, nous nous intéressons aux systèmes décomposables et au cadre de la conception préliminaire. Nous considérons qu'un système se décompose en composants et qu'un composant peut se décomposer en éléments. L'élément sera l'unité atomique insécable (non décomposable). Certaines des définitions présentées sont empruntées d'ouvrages ou adaptées de ceux-ci (référencés) ou encore formulées pour les travaux présentés.

Système Nous parlerons ici de *système* pour un ensemble de composants intéragissant dynamiquement, évoluant dans un environnement et ce dans le but de réaliser une finalité [12]. Par exemple, nous pouvons considérer qu'une batterie de VE est un système car décomposable en cellules.

Constituants Un constituant fera ici référence de manière indifférente à un élément, un composant ou un sous-système.

Elément Un élément fera ici référence à un objet ou une pièce ayant son unité et qu'on peut ajouter à un ensemble pour le compléter, pour former un tout plus vaste. Cette notion d'élément est différente de celle propre au formalisme DEPS, introduite dans le chapitre 3. Par exemple un fil de cuivre ou une tôle de fer seront considérés comme des éléments.

Composant Un composant fera ici référence à un objet physique réalisant une fonctionalité, mais constitué d'éléments sans fonctionalité définie en dehors du composant. Par exemple, nous considérons qu'une inductance est un composant.

La frontière entre système et composant n'est cependant pas formellement établie et peut apparaître comme "floue" [12]. Par exemple, selon la définition précédente, un transformateur de sécurité sera considéré comme un composant, bien qu'il s'agisse d'un système à part entière. Tandis qu'un assemblage pignon-crémaillère pourrait être considéré comme un système. Nous proposons la distinction suivante : si la conception d'un objet nécessite de considérer la conception d'au moins un de ses éléments a posteriori (à travers une sous-traitance et un projet de conception différent), il s'agira d'un système, sinon d'un composant.

Systèmes de systèmes Nous évoquerons les termes *système de systèmes* pour se référer à un système dont au moins un des constituants est un système. Un système de systèmes est

un ensemble de fonctions réparties sur plusieurs systèmes qui intéragissent entre eux. Ses constituants sont interconnectés pour obtenir de nouveaux effets et fonctionnalités qu'aucun système ne peut réaliser séparément. Cette définition est légèrement modifiée de celle présentée dans [13]. Par exemple, nous pouvons considerer qu'un VE est un système de système avec pour systèmes les constituants de la chaîne de conversion comme un convertisseur statique ou une batterie.

Sous-système Un sous-système sera considéré comme un système faisant partie d'un système de systèmes, de manière similaire à la définition proposée dans [13].

Environnement d'un système L'environnement d'un système fera ici référence à l'ensemble des autres systèmes et entités qui interagissent avec celui-ci (définition adaptée de [12]). Par exemple, si on cherche à concevoir une batterie de VE, nous pouvons considerer qu'un convertisseur statique fait partie de l'environnement du système batterie.

1.1.2 Notions de système sous-défini et défini

Système sous-défini

Concevoir un système consiste à le définir, en respectant un cahier des charges. On cherche alors à définir son architecture complète. La notion d'*architecture* comme nous l'utilisons dans ce manuscrit implique une structure dimensionnée et une topologie particulière définie par le nombre de ses composants, leurs types ainsi que leurs caractéristiques (constituants, matériaux, etc.). L'objectif de la conception est de passer d'un système sous-défini (où plusieurs architectures sont possibles) en pré-conception à un système défini (une architecture qui respecte toutes les contraintes du cahier des charges) en conception détaillée.

Les notions de *sous-définition* ou *sous-défini* sont rarement employées en GE et sont issues de [14–16] dans le cadre de l'Ingénierie Informatique. En fait, [14–16] sont les seuls travaux en faisant mention (entre 1980 et jusqu'au plus récent en 1998) et repris que jusqu'à très récemment, notamment en aéronautique [17]. Ces différents travaux définissent formellement les notions de *sous-définition* et de *modèles sous-définis* et évoquent celles de *valeurs admissibles* dans le contexte de la Programmation Par Contraintes (PPC) [18] pour la formalisation et la résolution de problèmes logiques. À ces données, on peut associer des ensembles de valeurs qui constituent des domaines. Le terme *sous-défini* y est initialement employé dans le cadre de la théorie des ensembles [19] pour faire référence à la représen-

tation d'ensembles partiellement connus [14]. Nous pensons que nous pouvons reprendre la notion de *sous-définition* pour distinguer un système partiellement ou totalement connu dans le cadre de la résolution des problèmes du monde réel et pas uniquement des problèmes purement mathématiques. C'est notamment le cas en début de pré-conception où l'architecture du système physique à concevoir n'est pas connue. Un système à concevoir est ainsi par nature sous-défini et un système conçu défini.

Variable

Plusieurs dénominations sont utilisées plus ou moins formellement en conception pluridisciplinaire pour se référer aux paramètres variables décrivant un problème lors de la conception. On peut généralement voir que les termes employés et leurs définitions dépendent fortement des domaines d'ingénierie concernés, et qu'il semble difficile d'adopter un vocabulaire commun. Quelques exemples sont mentionnés ci-après.

Par exemple, [20] (Génie Mécanique) propose 2 types de variables : les *variables de conception* qui font référence à la structure d'un système et les *variables de comportement* qui font référence à son comportement. Autre exemple, [21] (Génie Informatique (aide à la décision)) définit 4 types de variables : les *variables de conception*, les *variables de décision*, les *variables auxiliaires* et les *variables alias*. Les *variables de conception* sont définies comme les principales caractéristiques d'un système et sont généralement discrètes (choix de composants, de dimensions principales, de choix de matériaux, de formes ou de topologies). Les *variables de décision* sont employées comme synonyme de *variables de performance* et désignent les performances du système au regard d'un cahier des charges. Les *variables auxiliaires* désignent l'ensemble des autres variables. Les *variables alias* désignent des variables auxiliaires qui sont définies par d'autres variables à travers une expression dans le but de rendre un problème plus compréhensible. [22] (Génie Electrique) utilise par exemple les termes *paramètres* ou *variables objets* et de *variables de conception* pour décrire des facteurs influant sur un critère à minimiser. Autre exemple, [23] (Génie Informatique (aide à la décision), Génie Mécanique) propose notamment pour les variables : les *variables de conception* qui font référence à la structure d'un système et les *paramètres de conception* qui font référence à son comportement.

Dans ce manuscrit, nous considérons qu'une donnée pouvant prendre plusieurs valeurs (car on ne connaît pas sa valeur) est traduite par une Variable. Une Variable peut être soit structurelle et continue : volume, longueur, masse, ou structurelle et discrète : un type de

composants, leurs nombres, une couleur, etc., ou encore comportementale (qui est toujours continue : couple, vitesse, potentiel, tension, température, etc.). Nous divisons les Variables entre variables de conception (ou degré de liberté (ddl) [1]) et variables déduites. Les ddl seront les variables indépendantes. Une fois leurs valeurs déterminées, elles induiront celles des variables déduites. Pour les ddl, il s'agira généralement de variables structurelles discrètes pour un choix de composants, mais aussi structurelles continues pour du dimensionnement. En fonction du problème de conception, une variable continue pourra également être un ddl (ex : le couple électromagnétique pour une machine électrique [24–26]).

Variabilité

Le terme *variabilité* comme employé en Ingénierie des Lignes de Produits [27–29, 27, 30–33] exprime la caractéristique d'un système logiciel ou embarqué à présenter plusieurs topologies possibles (variabilité liée à la topologie, à travers l'utilisation de Variables structurelles discrètes) [2] C'est dans ce sens que nous reprenons le terme *variabilité* dans ce manuscrit, mais en prenant également en compte l'utilisation de Variables continues (notamment pour les dimensions et les comportements multiples que peuvent avoir un système). Lorsque nous mentionnons la variabilité structurelle, il s'agira de variabilité topologique (typ. un type et/ou nombre de composants variable ou différents agencements possibles de composants) et/ou dimensionnelle (dimensions variables d'une architecture à une autre). La variabilité comportementale fera référence au fait qu'un système présente plusieurs comportements (ex : charge / décharge d'une batterie). Nous présentons ci-après un exemple pour la variabilité structurelle et la sous-définition d'un système.

Exemple

Considérons un circuit électrique simple composé d'une source de tension continue d'une valeur e de $6V$ débitant un courant i, en série avec deux résistances électriques r_1 et r_2. Le cahier des charges (ou spécification des exigences) impose que la valeur de la résistance de r_1 soit connue : 10 Ω ainsi que la tension à ses bornes : $v_1 = 2V$ et

1. Une liste de Variables indépendantes choisies et qui a pour nombre le ddl du système. Le nombre de ddl renvoie à la cardinalité des choix possibles dans la conception.

2. Cette notion de *variabilité* exclut celle d'*incertitude de mesure* comme on la retrouve en métrologie et qui fait référence au doute que l'on a sur la validité ou l'exactitude d'une valeur mesurée (tolérance de fabrication, ex : entrefer $e = 1 \pm 0.1mm$ [34]). Elle exclut aussi celles d'*intervalle de confiance* ou *de probabilité* comme on les retrouve en modélisation, en commande et en conception *robuste*.

le courant la traversant : $i_1 = 0.2A$. La valeur de la résistance de r_2 est donc la seule inconnue : $r_2 = ?\ \Omega$. Le "système" sous-défini à concevoir est donc le composant r_2 qui présente une variabilité. Le problème de conception consiste à déterminer la valeur de la résistance de r_2, qui est le seul ddl et de type structurel continu. Le "système" r_1 est défini car sa résistance est connue et serait représenté par un modèle de système défini. Les contraintes de conception issues d'un modèle mathématique de la résistance (lois constitutives des résistances, interconnexions des systèmes) permettent d'en déduire le courant i_2 traversant r_2 et la tension à ses bornes v_2 tel que :

- — loi des mailles : $e = v_1 + v_2$;
- — lois des noeuds : $i = i_1 = i_2$;
- — lois d'Ohm : $v_1 = r_1 * i_1$ et $v_2 = r_2 * i_2$.

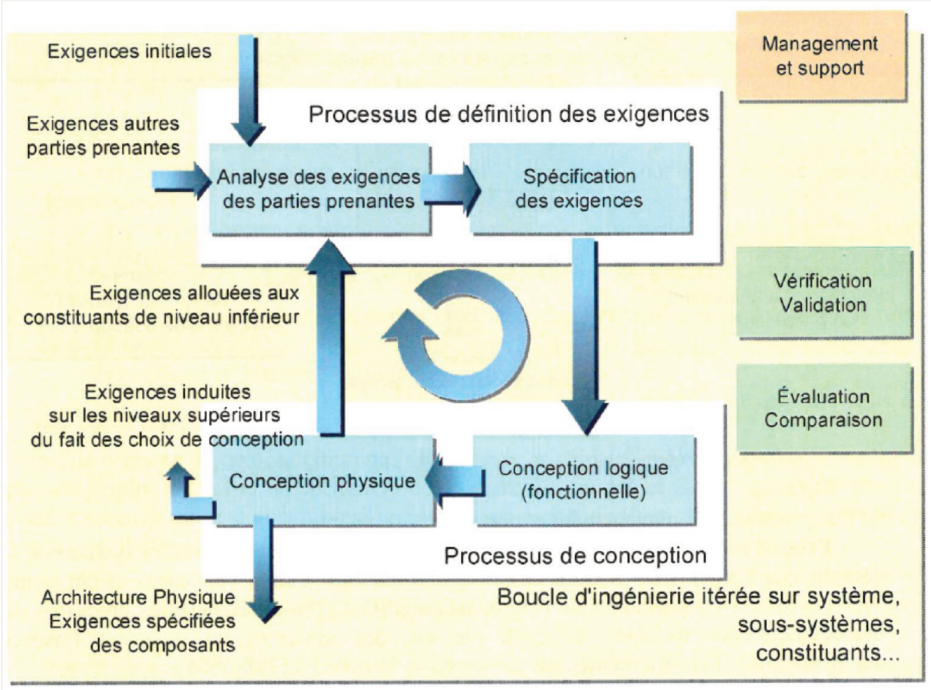
On en déduit ainsi les variables comportementales $i = i_2 = 0.2\text{A}$ et $v_2 = 4\text{V}$. Après résolution du problème, on trouve que $r_2 = 20\ \Omega$. r_2 est alors définie et est conçue.

Modèle de système sous-défini et défini

Dans ce manuscrit, nous parlerons de *modèle de système sous-défini* pour faire référence à une représentation du système sous-défini à concevoir et présentant ainsi de la variabilité structurelle. Par opposition, nous parlerons de *système défini* et de *modèle de système défini* pour faire référence respectivement à un système dont la structure est totalement connue, et à un modèle d'un tel système ne présentant aucune variabilité structurelle. Les seules Variables sont comportementales. Chaque ddl est instancié à une valeur. Alors que le modèle de système défini représente une unique architecture, celui de système sous-défini pourra en représenter plusieurs à la fois. Une architecture du système défini vérifiant l'ensemble des contraintes de conception du problème donné sera alors dite admissible [22, 2]. Les exigences du cahier des charges constituent une partie des contraintes de conception. L'autre partie serait spécifiée par les concepteurs qui peuvent avoir à réaliser des choix a priori durant la spécification du problème de conception.

Les notions d'*exigences* et de *spécification* apparaissent souvent à travers la littérature. Cependant, on peut voir qu'elles peuvent avoir des significations différentes en fonction du domaine d'ingénierie (ex : logiciel [35–39] ou physique [40, 41]) ou encore d'une traduction d'une langue à une autre. Nous avons ainsi besoin de les clarifier. Le dictionnaire Collins

Figure 1.1 Définir les exigences et concevoir itérativement le système et ses constituants en IS [13].

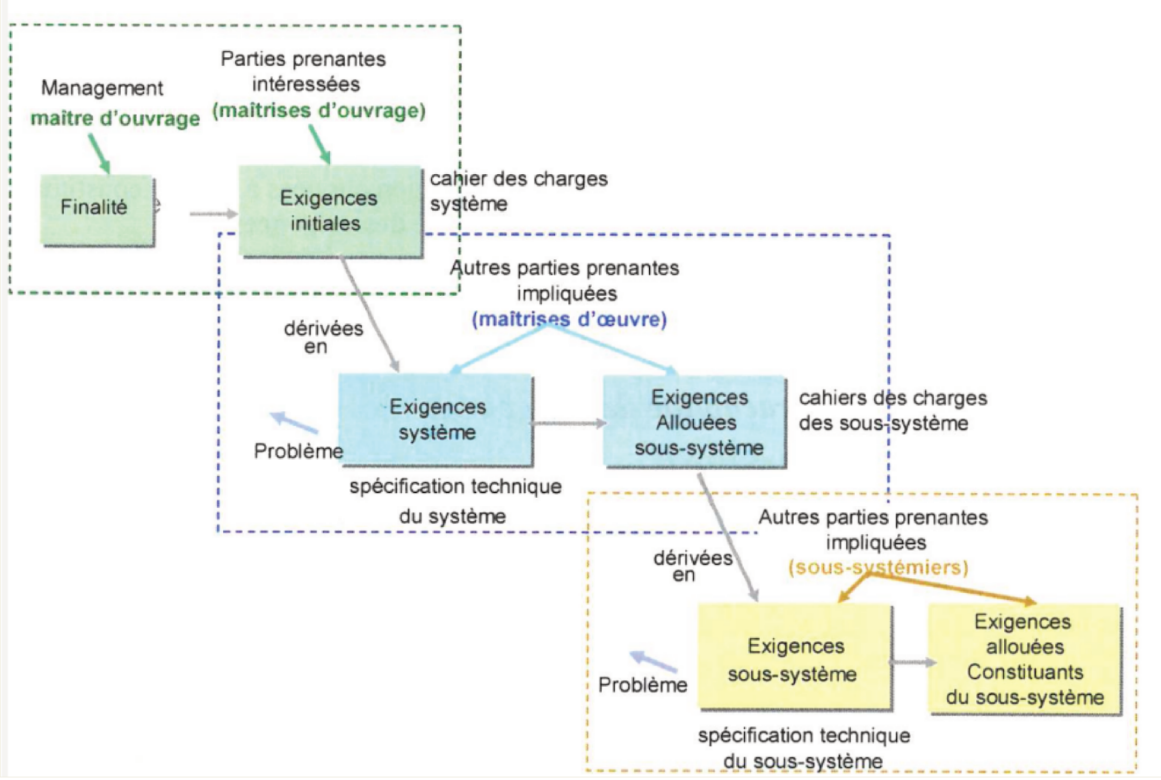


English propose des définitions générales des termes liés à l'*exigence* et la *spécification* qui permettent de faire facilement le lien avec l'ingénierie. Celui-ci définit une exigence comme "une qualité ou une qualification que l'on doit avoir afin de pouvoir faire quelque chose ou être adéquat à quelque chose." Pour revenir au cadre de la conception de systèmes, l'ingénierie des exigences est la branche des domaines d'ingénierie qui regroupe toutes les activités liées à l'établissement de la spécification des exigences à partir des exigences initiales (cf. Fig. 1.1).

1.1.3 Exigences système

Les termes *exigences* ou *exigences système* seront utilisés ici pour faire référence à l'ensemble des exigences fonctionnelles et non-fonctionnelles d'un système. Ces exigences sont des déclarations informelles et non précises mais pouvant être formalisées. Ces déclarations expriment ce que le système doit être, faire ou avoir. Elles sont formulées par les parties prenantes, composées de la clientèle, de l'ingénierie bureau d'étude et de la maîtrise d'ouvrage (système) ou de la maîtrise d'œuvre (constituants) [13].

Figure 1.2 Des exigences systèmes aux exigences des constituants [13].



La clientèle exprimera ses besoins à travers des exigences fonctionnelles mais aussi non-fonctionnelles. L'ingénierie bureau d'étude pourra événentuellement rajouter des exigences fonctionnelles pour pouvoir obtenir un système avant tout fonctionnel. Les exigences non-fonctionnelles peuvent impacter le fonctionnement du système. La maîtrise d'ouvrage ou d'œuvre négociera si nécessaire un compromis entre clientèle et ingénierie pour que le système soit théoriquement réalisable. Les exigences sont en général modifiées ou complétées au cours de la conception et allouées au différents constituants et domaines d'ingénieries (cf. Fig. 1.2). La classification d'une exigence en fonctionnelle ou non-fonctionnelle n'est pas toujours unique et peut dépendre du contexte du problème de conception à résoudre (il s'agit alors d'une interprétation du problème qui relève de l'ingénierie des exigences). Par exemple, prenons l'exigence "le couple doit être supérieur ou égal à 10 $N \cdot m$" pour une machine électrique à concevoir. Cette exigence peut être vue comme fonctionnelle (fournir la puissance nécessaire à la charge associée pour que le système remplisse sa fonction) ou non-fonctionnelle (pour améliorer les performances de la machine). Nous proposons la classification suivante d'une exigence, au regard de ces travaux de thèse : fonctionnelle, non-fonctionnelle de type 1 et non-fonctionnelle de type 2.

Exigences fonctionnelles

Les exigences fonctionnelles décrivent ce qu'un système doit réaliser pour fonctionner correctement, c'est-à-dire pour réaliser sa fonctionnalité. Par exemple, une bouilloire aura pour fonctionnalité de bouillir de l'eau. L'exigence associée est fonctionnelle et serait "Faire bouillir de l'eau". Si l'on souhaite contraindre le volume de la bouilloire pour que celle-ci puisse être vendue dans un emballage aux dimensions imposées, l'exigence d'intégration associée ne définirait pas une fonctionnalité du système.

Exigences non-fonctionnelles

Les exigences de performances et d'intégration d'un système sont généralement vues de manière dissociée des exigences fonctionnelles et non-fonctionnelles [42]. Nous considérerons dans ce manuscrit que les exigences de performances et d'intégration font partie intégrante des exigences non-fonctionnelles. Nous faisons explicitement la distinction entre les exigences non-fonctionnelles de type 1 et celles de type 2.

Exigences non-fonctionnelles de type 1 Celles-ci feront référence aux autres exigences de performances et d'intégration (ex : une limitation sur le volume). Elles sont généralement prise en compte dans le cadre de la conception ou de la conception optimale. Elles ne portent que sur le système considéré et sont le plus souvent exprimées sous formes d'inéquations. Elles sont également faciles à modéliser.

Exigences non-fonctionnelles de type 2 Celles-ci feront référence aux exigences traitant par exemple de sûreté de fonctionnement, de coût ou encore d'éco-conception. Elles peuvent ne porter que sur le système à concevoir mais aussi sur ceux interagissant avec lui, et sont exprimées sous diverses formes de modèles mathématiques à part entière via des équations, inéquations mais aussi logique via des tables ou des catalogues de composants. Elles sont plus difficiles à modéliser que les exigences de type 1. Elles sont également peu traitées en conception des systèmes physiques, à part les exigences de coût pouvant être déduites d'exigences non-fonctionnelles de type 1 (ex : le volume et la masse de matériaux permettent de contraindre le coût d'un système). Elles requièrent de pouvoir disposer de modèles mathématiques et autres relations informelles (comme des tables ou des abaques) pour les exprimer. Ces exigences sont également sujettes à des données fluctuantes (comme le coût de matières premières) ou sur lesquelles les concepteurs disposent de peu de connaissances. Il faut également pouvoir relier les modèles relatifs à ces exigences à celles du système (fonctionnelles).

Figure 1.3 Catégorisation d'exigences système pour un système spatial [13].

DOMAINE	EXPLICITATION	EXEMPLES
Fonctionnel	Ce que le système doit faire	Mission, fonctions du système, modes et états du système...
Configuration	Sa composition	Composants majeurs, modularité, accessibilité, éléments fournis...
Interfaces	Ses interfaces externes et internes	Externe : lanceur, GPS, équipage, interne : segment sol-segment spatial...
Physique	Ses caractéristiques physiques	Taille, masse, volume, capacités...
Environnemental	Les conditions sous lesquelles les fonctions sont réalisées	Accélération, altitude, contamination, hygrométrie, radiations, vibrations...
Facteurs de qualité	Les niveaux auxquels les fonctions doivent être réalisées	Sûreté, efficacité, fiabilité, maintenabilité, disponibilité, ergonomie, flexibilité
Exploitation	Les niveaux d'opérabilité	Autonomie, contrôle, management des dysfonctionnements...
Soutien	Le soutien nécessaire à la réalisation des fonctions	Maintenance, approvisionnement, formation, logistique...
Vérification	Méthodes de vérification des exigences	Inspections, revues, essais...

Leur couplage avec les exigences fonctionnelles rajoute ainsi une difficulté à la conception.

1.1.4 Spécification des exigences et spécification du système

Concernant le terme *spécification*, nous pensons que celui-ci ne peut pas être utilisé seul. Il s'agira toujours de faire référence à *la spécification de* .... Dans ce manuscrit, nous nous intéresserons à trois catégories de spécification : la spécification des exigences, la spécification du système et la spécification du problème de conception.

Le verbe *spécifier* est défini par le dictionnaire Collins de deux façons : "si on spécifie une chose, on donne des informations sur ce qui est requis ou devrait arriver dans une certaine situation" ou encore "si on spécifie ce qui devrait arriver ou être fait, on l'explique de manière explicite et détaillée." Le nom *spécification* est simplement référé par le dictionnaire Collins comme "une exigence clairement exprimée, par exemple à propos des caractéristiques nécessaires dans la conception d'une chose." Cette définition peut être reprise et adaptée pour définir les termes de spécification des exigences dans le cadre de la conception de systèmes.

Spécification des exigences

Dans ce manuscrit, nous ferons référence aux termes *spécification des exigences* comme étant le document formel et textuel relatant des exigences système d'une manière claire, précise, objective et explicite. Ce document doit permettre de définir clairement le système à

concevoir. Il doit rendre les exigences plus faciles à comprendre, pour permettre leurs vérifications et leurs validations. Il s'agit d'un contrat obtenu à l'issue de l'analyse des besoins avant la conception préliminaire et avec lequel chaque partie prenante est en accord. La spécification des exigences sera synonyme de cahier des charges dans ce manuscrit. Il exprimera le problème de conception du système considéré dans un langage *naturel* (ie. une langue parlée).

Plusieurs étapes peuvent généralement être suivies pour aider à la définition de la spécification des exigences. Tout d'abord, l'ensemble des exigences peut être retranscrit sur support (ex : papier, tableur). En second, cet ensemble pourra être structuré dans un document par catégorie d'exigences (fonctionnelles et non-fonctionnelles) et par niveau d'abstraction (cf. Fig 1.3). Dans un troisième temps, on pourra tenter de vérifier la consistence et la complétude de la structuration résultante. Par la suite, une priorité peut être assignée à chaque exigence. Finalement, ce document pourra être validé par les parties prenantes et résulter en la spécification des exigences.

Spécification du système

La spécification du système fera référence à la description des caractéristiques que le système doit avoir pour satisfaire la spécification des exigences. Cet ensemble de documents décrit la structure et le comportement du système.

1.2 Problème de conception de système

Spécification d'un problème de conception

La spécification d'un problème de conception fera ici référence à la définition d'un problème de conception. Celle-ci passera par la spécification des exigences et les choix de représentation du système à concevoir, ie. le choix *a priori* par l'ingénierie d'un modèle approprié de la structure et du comportement du système sous-défini et de celui de son environnement. Spécifier un problème de conception implique :

- — de définir le nombre total de Variables ;
- — de définir les ddl et leur nombre ;
- — d'en déduire les variables déduites et leur nombre ;

- de définir le nombre de contraintes de conception et leur type : égalité, inégalités, logiques, etc ;
- de représenter la structure du système sous-défini ;
- de représenter l'environnement du système sous-défini.

Difficulté du choix des ddl

Le plus souvent lors de la formalisation des problèmes de taille relativement grande, nous avons tendance à déclarer trop ou pas assez de ddl. En pratique, leur choix dépendra fortement du domaine d'expertise dans lequel on se place à savoir si la conception est réalisée d'un point de vue mécanique, magnétique, etc. Aussi, ces choix peuvent être difficiles à réaliser si le problème semble grand. Spécifier un problème de conception est une activité complexe pour les concepteurs qui se doivent de comprendre un système complexe. Celle-ci demande beaucoup d'expertise et d'expérience.

1.2.1 Classification des problèmes de conception en fonction de la structure du système à définir

Les problèmes de conception de systèmes physiques peuvent par exemple être classés selon ce que l'on cherche à définir au niveau de la topologie du système ou ses dimensions. En conception de systèmes, on peut identifier 4 catégories de problèmes qui expriment le besoin de trouver une ou plusieures architectures admissibles pour un système à concevoir à différents degrés de variabilité structurelle (dimensionnelle et/ou topologique) [17].

Le problème de dimensionnement [43, 44] a pour but de définir les dimensions du système (longueur, largeur, hauteur). La structure discrète du système est déjà fixée, ie. le type et le nombre de composants sont imposés. Il s'agit en général de problèmes à dominante continue (ce qui sous-entend que la plupart des Variables sont continues).

Le problème de configuration [45, 46] consiste à définir les types et les nombres de constituants du système mais pas à les dimensionner (les Variables sont généralement discrètes). En général, ceux-ci existent déjà sur étagères ou dans des catalogues.

Le problème de génération d'architectures [17] consiste à générer des architectures entièrement, à savoir définir configuration et dimensions. On résout alors à la fois un problème de dimensionnement et un de configuration du système.

La plupart des cas de conception réels impliquent des Variables mixtes, ie. mélangeant des Variables continues et discrètes (qualitatives et entières). Celles-ci sont définies dans des domaines de dimensions finies (intervalles d'énumérés, qui peuvent être à valeurs réelles ou discrètes) ou infinies (intervalles de réels).

Les problèmes de dimensionnement, de configuration et de génération d'architectures sont ceux qui nous intéressent pour la conception des systèmes électriques (et plus généralement physiques). En particulier, le problème de génération d'architectures qui est généralement mixte laisse la plus grande liberté de conception et donc le plus de possibilités pour obtenir des solutions techniques innovantes.

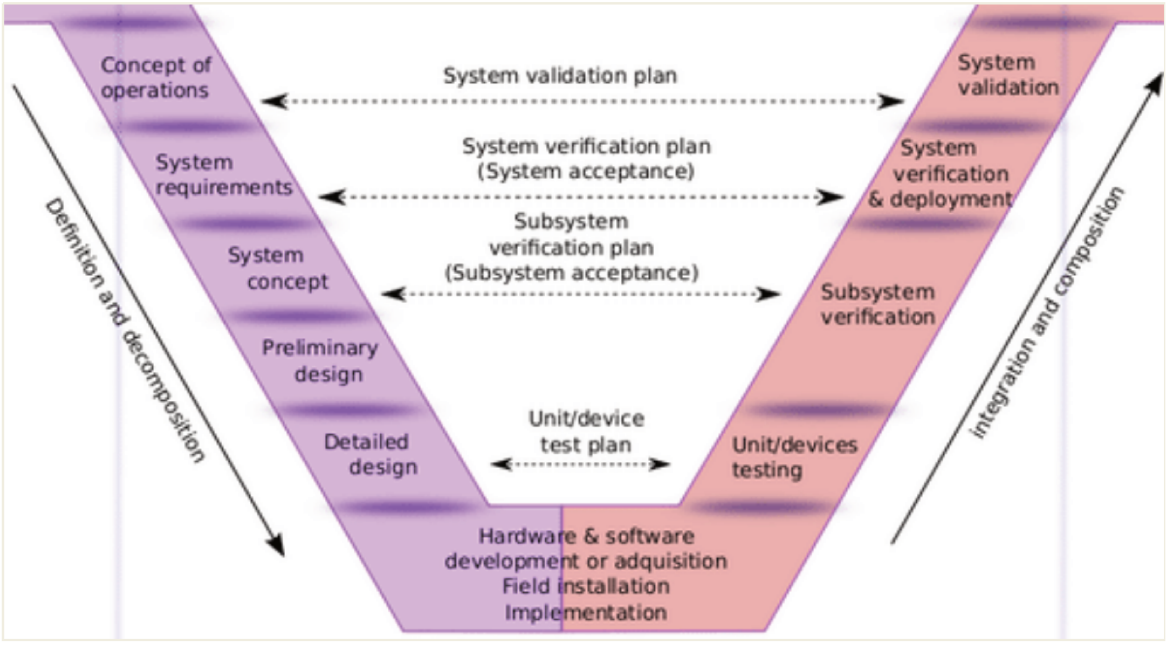
Les problèmes de génération d'architectures et de configuration sont moins étudiés et souvent plus difficiles que ceux de dimensionnement où l'architecture du système est déjà fixée. Les problèmes de dimensionnement peuvent dans certains cas n'être que des problèmes continus, contrairement aux problèmes de génération d'architectures et de configuration qui sont forcément discrets et le plus souvent mixtes, ce qui pose une difficulté supplémentaire [46, 47]. Contrairement aux problèmes de génération d'architectures, les problèmes de dimensionnement n'incluent pas également de règles de configuration entre composants d'un système [48] ce qui les rend plus simples. Ces règles sont spécifiées dans la spécification des exigences. Par exemple, on ne pourra pas spécifier explicitement que deux composants ne puissent pas être choisis en même temps pour une architecture admissible (exclusion mutuelle entre deux composants).

Le problème d'allocation de ressources [49, 50] vise à allouer des constituants physiques (ressources) à des constituants logiciels (tâches). Ce problème concernera plutôt les systèmes logiciels, informatiques ou embarqués. On peut retrouver cette catégorie de problèmes en conception de systèmes cyber-physiques.

1.2.2 Cycles de développement d'un système physique

L'Ingénierie des Systèmes (IS) est un ensemble d'approches et processus visant à maîtriser la conception, l'évolution et la vérification des systèmes complexes. Le développement d'un système repose communément sur l'utilisation d'un cycle. Il existe une multitude de cycles de développement, employés à travers les nombreux domaines d'ingénierie, comme par exemple en cascade [52], en V [51], itératif [53], semi-itératif [54], en spiral [55] ou

Figure 1.4 Le cycle de conception en V [51].



encore en Y [56]. Ces cycles reposent sur de l'évaluation, de la vérification et de la validation d'architectures. Ces travaux de thèse s'intéressant à la phase de conception préliminaire, nous prenons l'exemple du cycle en V qui permet de positionner clairement cette phase dans le cycle de développement. Par exemple, le cycle en V (cf. Fig. 1.4) est originellement issu de l'Ingénierie des Logiciels (IL) mais est aussi utilisé en pratique pour les systèmes physiques. On distingue en pratique (sur la branche descendante) : la phase d'analyse des besoins, de définition de la spécification des exigences et les phases de conception :

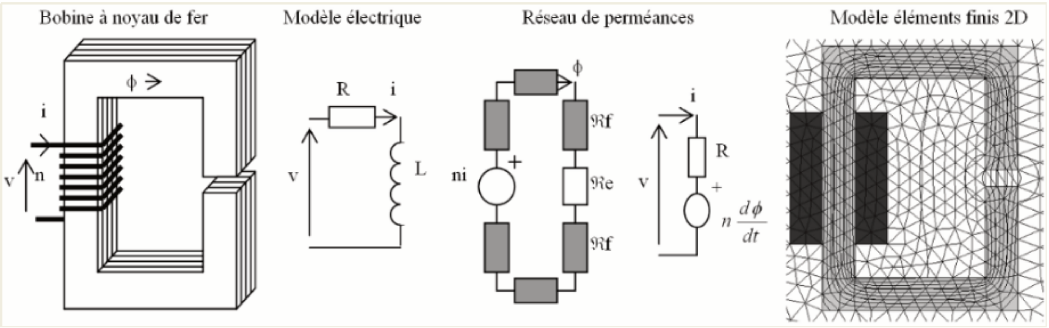
1. conception préliminaire (ou pré-conception) ;
2. conception détaillée ;

et les phases d'Intégration, Vérification & Validation (IVV) :

3. intégration ;
4. vérification ;
5. validation.

Pour accélérer la conception, la définition de la spécification des exigences et les deux phases de conception peuvent être traitées de manière parallèle et en interaction plutôt que séquentielle (cadre de l'ingénierie concurrente [57]). Les boucles de retour entre ces différentes phases sont ainsi limitées.

Figure 1.5 Une bobine à noyau ferromagnétique et ses différents modèles [58].



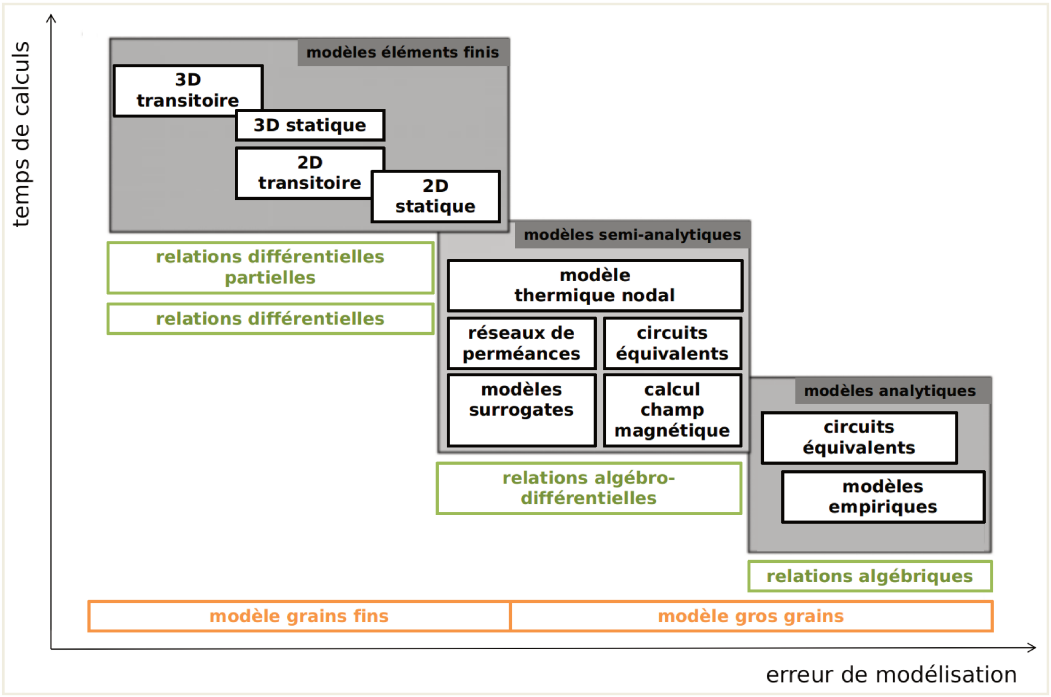
1.2.3 Modélisation des systèmes physiques

Durant la conception les concepteurs peuvent s'appuyer sur différents modèles plus ou moins détaillés afin de représenter les structures et comportements multiples du système complexe.

Les modèles mathématiques que l'on peut considérer en pré-conception des systèmes physiques prennent en compte différemment les aspects spatiaux (liés à la géométrie du système) et temporels / fréquentiels (liés à la dynamique du système). Plusieurs modèles complémentaires à granularités différentes sont généralement nécessaires lors de la conception (ex : modèles empiriques, réseaux de perméances, éléments finis 3D transitoire). Ces modèles, utilisés en pré-conception et/ou en conception détaillée, peuvent être de type boîtes noires (codes de calculs) ou boîtes blanches. Les modèles de type boîtes blanches sont généralement séparés en analytiques, semi-analytiques, algébriques ou encore numériques. La Fig. 1.5 montre différents niveaux de granularité possibles pour une bobine à noyau ferromagnétique. Il n'existe cependant pas de classification officielle de ces modèles, par exemple selon leurs utilisations en conception (quel modèle pour quelle phase de conception) ou encore en fonction de leur niveau de granularité.

Nous proposons dans ce manuscrit une classification pour les modèles de type boîtes blanches comme ci-après et résumée Fig. 1.6. D'une part, cette classification est inspirée de celle proposée dans [58] pour organiser les modèles selon leurs erreurs de modélisation (qui augmentent lorsque le niveau de granularité diminue). D'autre part, nous nous inspirons des notions de modèles *gros grains* et *grains fins* traduites de l'anglais (resp. *coarse grained model* et *fine grained model*), très utilisées en synthèse dans le domaine de la biochimie (et dont elles sont issues) [59–61]. La limite entre modèles gros grains et grains fins n'est pas stricte, notamment en GE et qu'il peut y avoir des modèles qui rentreraient dans l'une ou l'autre de ces catégories selon les situations. Cette limite dépend en pratique de l'outil uti-

Figure 1.6 Granularité des modèles mathématiques pour la conception de systèmes physiques électriques (adapté de [58]).



lisé (des techniques implémentées pour la résolution). La limite entre modèles gros grains et grains fins que nous proposons d'utiliser est liée à la capacité des outils de synthèse et d'analyse pris pour exemple dans l'approche présentée Chapitre 4. Dans cette approche, nous nous limitons à l'utilisation de modèles gros grains, dans lesquels nous regroupons les modèles algébriques et les tables de valeurs compatibles (DEPS et algébro-différentielles (Modelica). Les modèles grains fins pourront regrouper les modèles aux équations différentielles (ordinaires et partielles) et intégrales.

Le modèle gros grains algébrique comprend exclusivement des relations algébriques. La dimension spatiale se traduit par des relations équivalentes (ex : circuits équivalents, modèles empiriques). La dimension temporelle est considérée de manière implicite : le comportement du système est considéré en régime permanent ou en régime maximal (ex : à couple maximal pour une machine électrique) par une représentation fréquentielle (ie. statique ou quasi-statique).

Le modèle gros grains algébro-différentiel est constitué de relations algébro-différentielles et dépend du temps. La modélisation est temporelle. Le système est défini et on cherchera à évaluer ses architectures.

Le modèle grains fins Les modèles mathématiques différentiels et différentiels partiels en revanche seront vus comme des modèles grains fins. Un modèle grains fins est un modèle numérique (par exemple un modèle éléments finis 2D/3D). Les modèles grains fins sont plus précis que les gros grains et autorisent une meilleure représentation des phénomènes physiques (électrique, magnétique, mécanique ou thermique) et ce même si les phénomènes sont fortement couplés [58]. Mais ils nécessitent que la géométrie soit déjà entièrement spécifiée et sont en général peu intuitifs, lourds en temps de calcul.

1.2.4 Notion de point de vue acausal

Acausalité Le terme *acausalité* [62] signifie qu'on ne spécifie pas de causalité. Un problème peut être modélisé de manière acausale en employant uniquement des inéquations et équations au lieu d'affectations ou d'algorithmes. Les équations peuvent être manipulées de manière symbolique pour la résolution. Par exemple, la loi d'Ohm réecrite de manière acausale sous forme d'équation $e - r * i = 0$ n'impliquera pas que cette résistance soit branchée à une source de tension ou de courant.

Point de vue acausal Le point de vue acausal comme nous l'entendons dans ce manuscrit ne spécifie aucune causalité entre les exigences non-fonctionnelles (de type 1 et 2) et la structure du système. Le modèle n'est pas orienté et n'a ni entrées ni sorties. Les concepteurs doivent choisir les ddl. Même si un problème de conception pourrait n'être représenté que par des (in)équations et des ddl, il sera nécessaire en pratique de spécifier des variables déduites pour conserver le sens physique du problème et donc sa compréhension.

Orientation d'un modèle Lorsque le point de vue est causal, un modèle est orienté selon un sens : les performances sont déduites de la structure ou le contraire. Un modèle sera dit direct si les performances sont déduites de la structure d'une architecture du système. Un modèle sera dit inverse si la structure est déduite des performances ou du comportement du système [58].

1.2.5 Notions d'espace du problème, des connaissances et des solutions

Dans ces travaux de thèse, nous avons considéré que l'espace de conception est divisé en trois sous-espaces : l'espace du problème (*problem space*), des connaissances (*knowledge space*) et des solutions (*solutions space*).

Les notions d'*espace du problème* et d'*espace des solutions* ont été introduites en Ingénierie des Logiciels (IL) [63, 35, 31, 64, 29]. Ces notions ont été reprises et adaptées par l'Ingénierie des Systèmes (IS) [40, 13]. Ces deux notions ont été introduites dans le cadre de la formalisation des problèmes de conception pour séparer la spécification du problème (dans l'espace du problème) des architectures du système défini (dans l'espace des solutions). Une meilleure compréhension du problème était également recherchée.

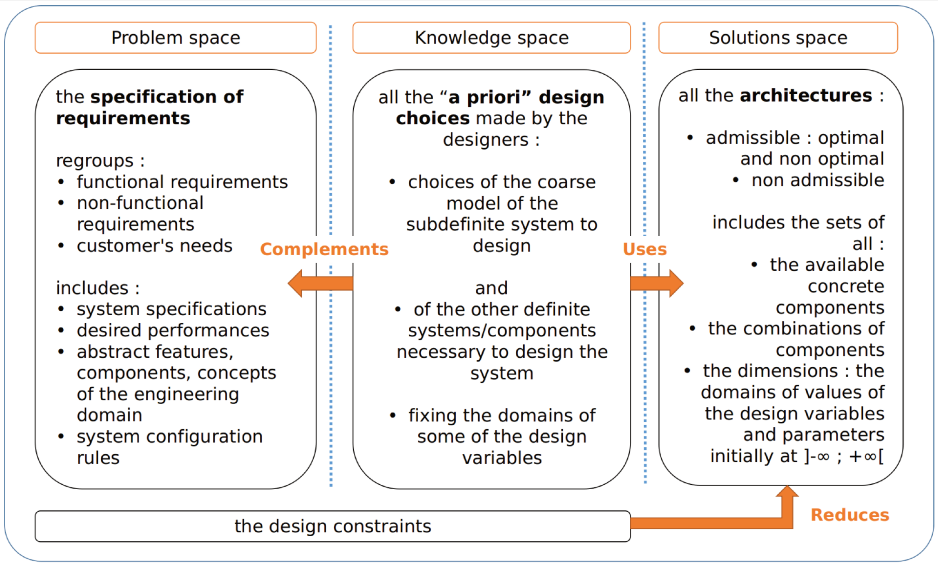
Parmis les premiers travaux étayant ces idées, [39, 37, 65, 66, 36] ont développé l'approche des *Problem Frames* dans laquelle l'*environment domain* et le *system domain* sont respectivement associés aux éléments du problème et à ceux des solutions. Les *Problem Frames* cherchent à gérer les problèmes complexes de conception de systèmes logiciels en identifiant des sous-problèmes plus simples et récurrents à de nombreux systèmes. Le problème complexe est alors représenté par une composition de plusieurs de ces sous-problèmes. Le cadre des *Problem Frames* est basé sur le *Two ellipse model* développé dans [65] qui divise l'espace de conception en 2 sous-espaces : celui du système à concevoir et celui de *l'environnement* dans lequel il sera installé. Ce dernier est composé des besoins de la clientèle (appelés *requirements*) et *monde* décrivant le-dit environnement. Le système sous-défini (le programme) n'est cependant pas séparé de la machine dans laquelle il sera implémenté, et constituent ensemble l'espace du système.

Dans [31], l'espace du problème contient les exigences et l'espace des solutions contient les produits, qui sont les architectures admissibles. Les règles de configuration de systèmes se retrouvent entre les deux espaces et permettent de faire le lien lors de la conception, qui se traduit par le passage de l'espace du problème à celui des solutions.

Les notions d'espace du problème et des solutions présentées dans les travaux d'IL [31] ont été modifiées et étendues pour ces travaux de thèse. Dans ce manuscrit, l'espace du problème contient la spécification des exigences et en particulier les règles de configuration de système. Les concepts génériques définissant des catégories de constituants (ex : un moteur) et leur variabilité structurelle (ex : mono- ou tri-phasé) en font également partie. Un problème de conception s'exprime naturellement à l'aide de l'espace du problème.

L'espace des solutions correspond à l'espace à explorer lors de la résolution : il contient l'ensemble des architectures admissibles, optimales ou non, et celles non admissibles. Cet

Figure 1.7 L'espace du problème, celui des connaissances et celui des solutions [67].



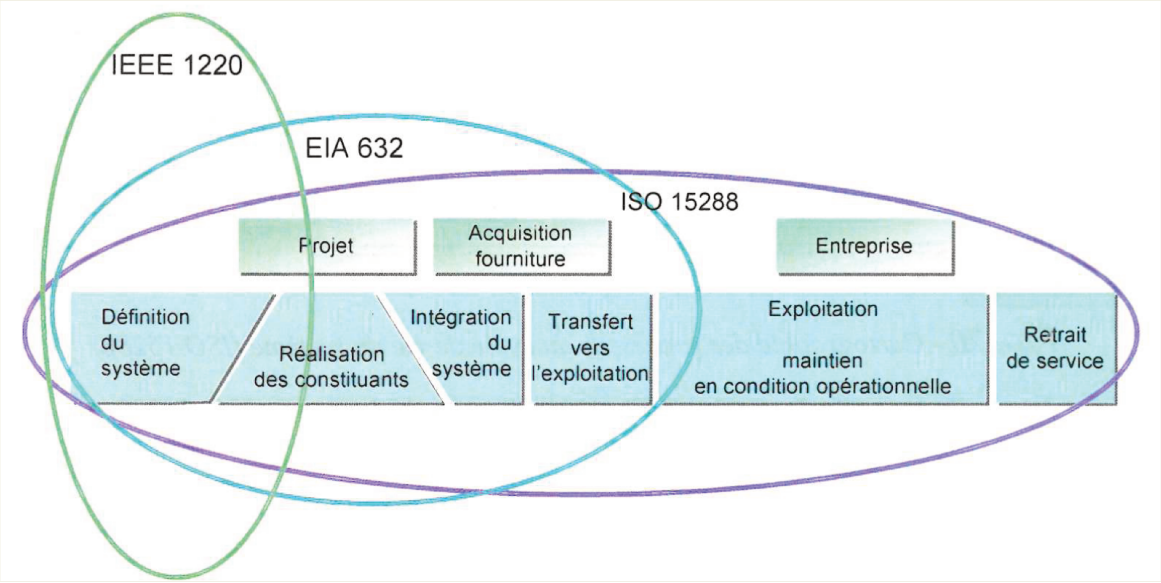
espace contient ainsi l'ensemble des composants concrets (ex : le moteur pas-à-pas 3EOX) et celui des dimensions.

Dans le cadre de ces travaux de thèse, nous avons enrichi ces concepts en y ajoutant celui *d'espace des connaissances* (cf. Fig. 1.7). L'espace des connaissances regroupe toutes les décisions basées sur l'expertise et l'expérience pour le choix du type de modèles mathématiques ou du domaine d'un ddl. Ces domaines sont sélectionnés dans l'espace des solutions pour compléter la spécification des exigences, ie. celui du problème. L'espace des connaissances aide ainsi à obtenir des architectures réalistes. Par exemple, un entrefer mécanique de moteur électrique sera technologiquement contraint à une valeur minimale d'1 *mm*, sinon le système ne sera pas réalisable. C'est une connaissance que détiennent les concepteurs et experts du domaine technologique considéré. Les contraintes de conception sont constituées à partir de l'espace du problème et des connaissances. Elles permettent de restreindre l'espace des solutions.

1.3 Conception à travers l'analyse d'architectures

L'analyse d'architectures est un processus de conception qui adopte un point de vue causal tel que les performances soient déduites de la structure d'un système défini. L'analyse d'architectures repose sur l'utilisation d'un modèle d'analyse. Ce modèle d'analyse est construit à partir d'un modèle mathématique direct du système (les performances sont évaluées à par-

Figure 1.8 Norme IEEE 1220 : définir le système (horizontal) au plus haut niveau de détail (vertical) [13].



tir d'une architecture) et d'un formalisme pour la spécification de systèmes. Le problème de conception est implicitement exprimé depuis l'espace des solutions. L'exploration de l'espace des solutions repose sur la connaissance a priori d'une architecture existante [68, 69].

1.3.1 Les normes de conception en Ingénierie des Systèmes

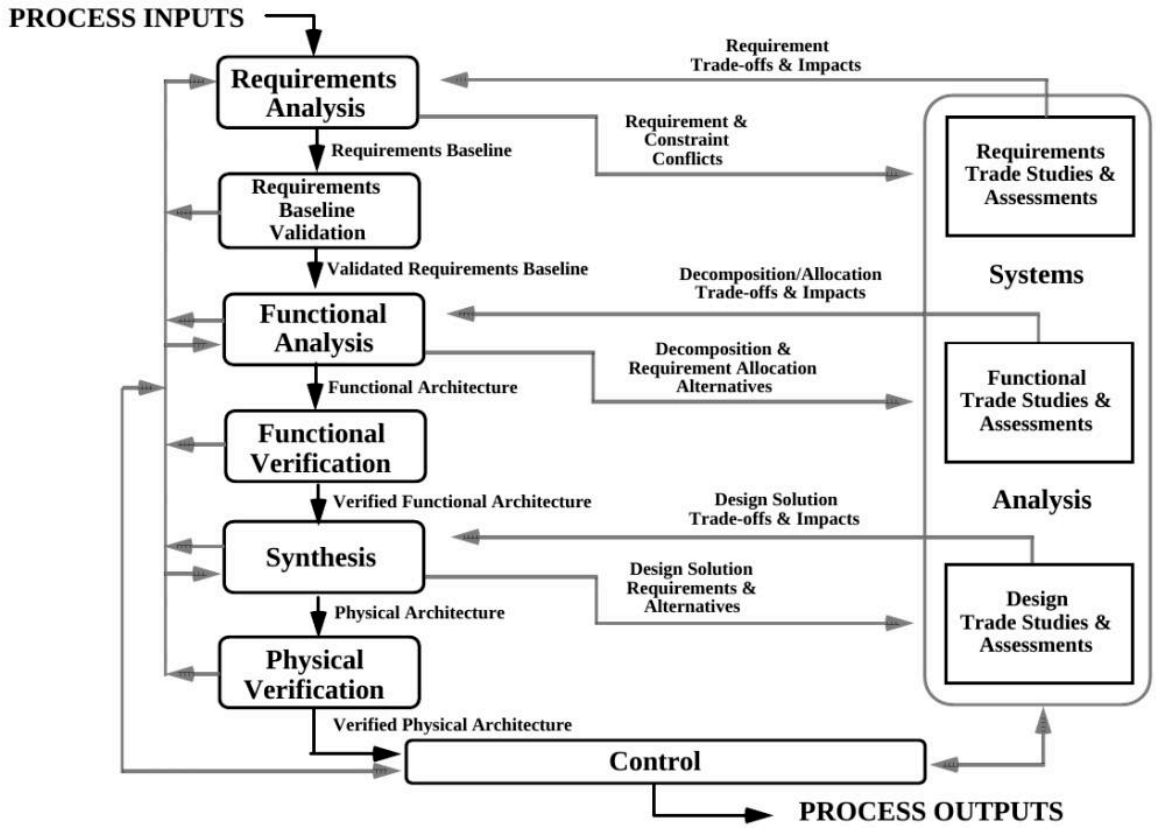
Les premiers standards de processus d'Ingénierie des Systèmes (IS) (1969) ont donnés par la suite des normes générales pour l'IS. Ainsi, trois normes ont initialement été formulées pour décrire les processus de l'IS : IEEE 1220 [70], EIA/ANSI 632 [71] et ISO 15288 [72]. Ces normes complémentaires établissent quels types d'activités sont à réaliser à différents niveaux de détails de la conception et quels résultats doivent être obtenus (cf. Fig. 1.8). Généralement, la conception de systèmes physiques est réalisée selon la norme IEEE 1220 (cf. Fig. 1.9). Bien qu'initialement dédiée à l'IL, celle-ci est également utilisée pour les systèmes physiques.

1.3.2 Conception selon le Model-Based System Engineering (MBSE)

Représentation des systèmes physiques réels par des modèles

Le système est représenté selon les différents niveaux de détails et points de vue que l'on peut avoir dessus par des modèles : structurellement (architecture fonctionnelle et physique

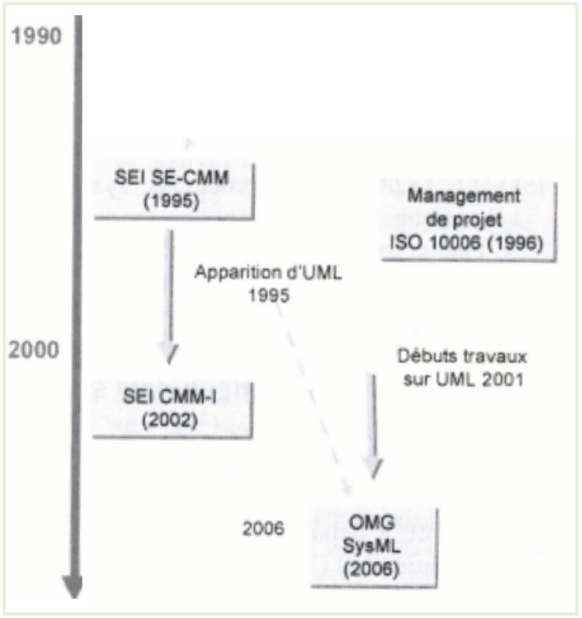
Figure 1.9 Détail des activités de la norme IEEE 1220 [70].



du système, ie. ses composants reliés les uns aux autres et avec l'environnement) et comportementalement (comment intéragissent les composants et quelles informations sont échangées). Ces modèles peuvent aussi bien être des plans que des modèles réduits du système, des systèmes d'équations, etc. Ils peuvent servir à des fins de communications entre parties prenantes ou concepteurs ou directement pour la conception. L'idée est de représenter le système réel de manière simplifiée, en éliminant les composants non nécessaires pour faciliter la compréhension, aider à la décision, examiner différents scénarios ou encore contrôler ou prédire des événements [13].

La conception dite *basée modèles* repose ainsi sur : un langage, une approche et des outils logiciels de modélisation de systèmes. Nous parlerons de *langage* pour désigner de manière générale un langage informel, semi-formel ou formel et de *formalisme* pour un langage formel. Le langage cherche à supporter les phases de conception (et il peut être structuré, orienté-objet, etc.).

Figure 1.10 Chronologie de l'apparition de SysML [13].

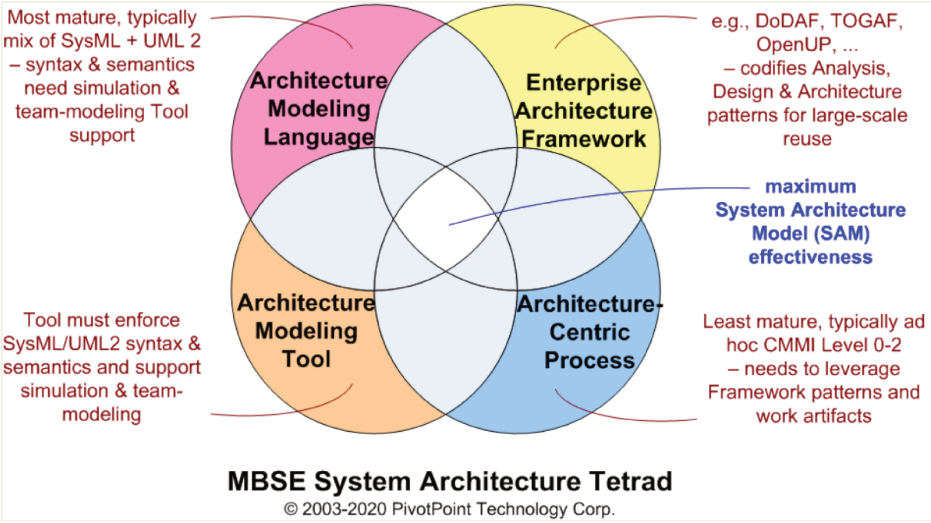


Approches et langages MBSE

Les approches basées modèles pour l'IS ou Model-Based System Engineering (MBSE) [73, 74] sont issues de l'IL. Les années 90 ont vu émerger les approches par objets et la création de l'Unified Modeling Language (UML) en 2001 par l'Object Management Group (OMG). L'IS n'ayant pas pu unifier la modélisation des systèmes, UML a dans un premier temps été adopté en 2003 comme langage commun pour l'IS. La nécessité de décrire des caractéristiques non présentes dans UML (exigences et leurs traçabilités vers l'architecture, élements non logiciels et leurs quantités physiques, représentation de flux continus comme l'énergie) a favorisé l'adaptation d'UML2 (2004) pour les systèmes physiques avec la création de Systems Modeling Language (SysML) adopté en 2006 par l'OMG et l'INCOSE (cf. Fig. 1.10).

L'adoption de SysML a favorisé le développement des approches MBSE au sein de l'IS. Les approches MBSE permettent également de développer des modèles cohérents du système et facilitent l'établissement des liens entre composants. Elles donnent un cadre favorable à la pré-conception des systèmes complexes et autorisent une réutilisation des modèles créés.

Figure 1.11 Les 4 piliers de la modélisation MBSE pour l'analyse d'architectures (*sysml.org*).



1.3.3 Modélisation des systèmes pour l'analyse

Les 3 normes d'IS définissent les piliers de la modélisation et de la simulation des systèmes dans le cadre de l'analyse d'architectures [13]. Les 5 piliers sur lesquels reposent l'analyse d'architectures peuvent être identifiés comme étant :

1. un formalisme permettant de décrire un système défini selon des modèles gros grains ou grains fins ;
2. une approche orientée solutions (depuis l'espace des solutions vers celui du problème) ;
3. un outil logiciel de modélisation de systèmes définis ;
4. une technique de résolution d'équations pour la simulation ;
5. un outil logiciel de simulation implémentant la technique précédente.

MBSE et analyse

Les approches MBSE issus de l'IS reposent ainsi sur l'utilisation de modèles pour supporter les phases du cycle en V notamment à l'aide d'un langage dit MBSE comme SysML, d'un outil d'analyse d'architectures, de processus et d'*Architecture Framework* [1] (cf. Fig. 1.11). Les langages MBSE que l'on retrouve couramment dans la littérature sont UML, MARTE, AADL, EAST-ADL(UML2), SysML, LML [2] ou encore Modelica.

1. Ensemble de conventions, principes et pratiques à utiliser pour les activités liées à l'architecture d'un système, établies au sein d'un domaine d'ingénierie ou de parties prenantes.
2. http ://www.lifecyclemodeling.org/spec/LML_Specification_1_0.pdf.

Une première approche MBSE basée sur les langages de type UML (UML, MARTE, AADL, EASTADL(2), SysML et LML) peut être extraite. Ces langages de modélisation dédiés ou Domain-Specific Modeling Language (DSML) facilitent l'expression des concepts d'un domaine particulier. UML, MARTE, AADL et EAST-ADL(UML2) sont cependant moins utilisés en IS car ils ont été développés pour les systèmes à dominante logicielle. Ces 4 langages de spécification de systèmes ne sont donc pas adaptés pour modéliser un problème de conception en GE. Le Lifecycle Modeling Language (LML) est un récent langage de spécification de systèmes pour l'IS. Il a été créé pour remplacer UML et SysML, qui étaient jugés trop compliqués pour l'IS. Cependant, il n'y a que très peu d'informations sur LML : seule la spécification semble disponible et il ne semble pas y avoir d'utilisation avec des cas concrets.

Une seconde approche MBSE de modélisation de systèmes définis courante en pré-conception des systèmes physiques est basée sur le formalisme Modelica. En GE, Modelica et ses outils sont aussi très employés pour la conception par analyse, mais on retrouve aussi très souvent des outils comme Matlab ou Scilab pour la dynamique ou plus rarement GREET [75] pour la LCA [76] (qui ne sont généralement pas catégorisés comme outils MBSE). Modelica est un formalisme de spécification de systèmes libre développé depuis 1996 par l'Association Modelica et utilisé dès les années 2000 en industrie. Le but de Modelica est de modéliser et de simuler le comportement dynamique des systèmes physiques de manière pratique pour les ingénieries orientées systèmes physiques (électrique, mécanique, électronique, etc.). Les modèles sont gros grains algébriques ou algébro-différentiels mais on peut aussi se servir de résultats issus des modèles éléments finis. Modelica est donc utilisé pour la conception par analyse d'architectures. De nombreux environnements de simulation commerciaux et académiques ont vu le jour de part et d'autre des industriels et des académiques. Les grands éditeurs de logiciels ont presque tous sorti leur version de logiciel basée sur Modelica. On peut citer notamment les logiciels industriels AMESIM (de Siemens), Dymola (de Dassault Systèmes), MapleSim (de Maplesoft), Wolfram SystemModeler (anciennement MathModelica - de Wolfram Research) ou encore le logiciel libre OpenModelica côté académique (de l'Université de Linköping).

Les deux approches de modélisation de systèmes définis en IS les plus courantes et pertinentes (utilisés à la fois par les industriels et les académiques) semblent a priori basée sur les langages SysML et Modelica.

Ambuiguité autour du langage SysML

SysML *se présente* comme un formalisme de modélisation pour "comprendre et décrire les besoins, spécifier et documenter les systèmes, esquisser des architectures logicielles, communiquer des points de vue et concevoir des solutions". Cependant, la question se pose en pratique de savoir si SysML est bien un langage de spécification de problème. En effet, la Fig. 1.11 et le site *sysml.org* [1] présente les modèles crées formellement avec SysML et le langage de contraintes Object Constraint Language (OCL) comme étant simulables, voire exécutables (avec génération de code).

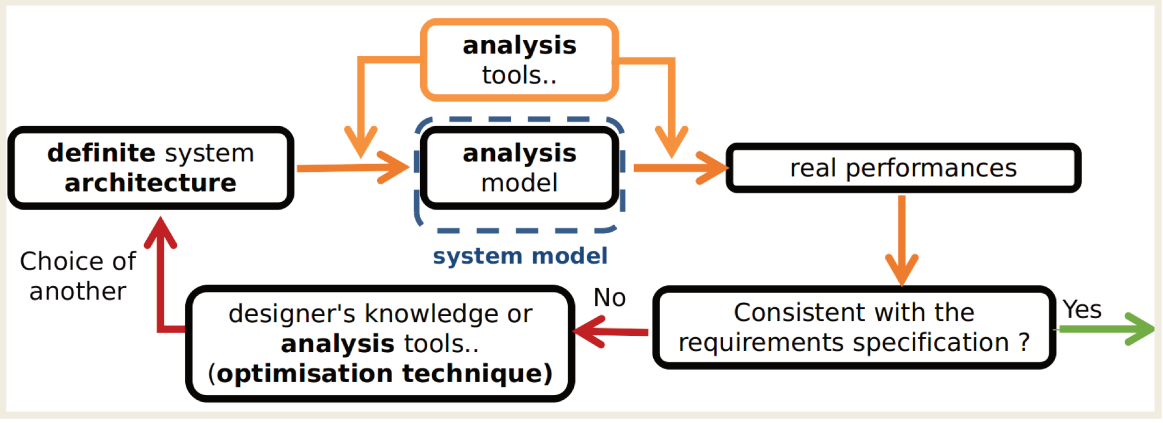
SysML est un langage semi-formel. Aucun des diagrammes SysML n'est formel. L'utilisation d'OCL est fortement recommandée mais n'est pas obligatoire. Si les contraintes ne sont pas formellement spécifiées avec OCL, les modèles ne sont pas du tout simulables. Les diagrammes informels (d'exigences *req*, de cas d'utilisation *uc*, et de package *pkg*) et les tables d'allocations SysML ne peuvent pas être utilisées pour la simulation. En particulier le diagramme *uc* qui est classé comme un diagramme de comportement, possède une sémantique ambigue et incomplète, contrairement aux diagrammes d'état machine *stm*, d'activités *act* et de séquence *seq*.

Un autre argument qui confirmerait que SysML soit un langage de spécification de systèmes plutôt que de problèmes de conception est son utilisation actuelle. Les 3 utilisations de SysML pour l'analyse sont ainsi spécifiées : *SysML-as-Model-Simulation*, *SysML-as-System-Architecture-Blueprint* et *SysML-as-Executable-System-Architecture*. Les inconvénients d'un langage semi-formel ont été capturés à travers la 4ème utilisation de SysML dite *SysML-as-Pretty-Pictures* :

1. *SysML-as-Pretty-Pictures* : usage le moins rigoureux et le plus fréquent. Les modèles sont rarement utilisables pour la simulation dynamique (analyse).
2. *SysML-as-Model-Simulation* : simulation dynamique partielle du comportement du système.
3. *SysML-as-System-Architecture-Blueprint* : peu utilisé. Il s'agit d'un modèle simulable suffisamment précis pour être communiqué entre les ingénieries impliquées (systèmes, électriques, mécaniques, logiciels, etc.).
4. *SysML-as-Executable-System-Architecture* : usage rare. Il s'agit d'une extension entièrement simulable du précédent voire exécutable (ie génération automatisée d'interfaces systèmes et de cas d'utilisations).

1. consulté le 16/10/2020.

Figure 1.12 La conception par analyse d'architectures.



A l'heure actuelle, les outils [1] utilisant les modèles SysML pour la conception sont des outils d'analyse-simulation. Quelques travaux font exception, en essayant de coupler SysML avec différents outils d'optimisation [77].

Du point de vue de l'obtention des architectures admissibles, SysML n'est utilisable que comme un langage de specification de systèmes. Chaque architecture à évaluer doit être modélisée et définie. Du point de vue de la modélisation, SysML serait théoriquement utilisable comme un langage de specification de problème de conception (mais pas en pratique car aucun outil de résolution n'est associé). Les capacités de SysML sont discutés pour la formalisation des problèmes avec le cas d'étude présenté au Chapitre 3 (cf. Annexe E).

1.3.4 Résolution en analyse

Pour résoudre un problème de conception par une approche d'analyse, des architectures sont choisies par l'espace de connaissances puis évaluées et modifiées par essai-erreur si elles sont non admissibles (cf. Fig. 1.12). On conçoit indirectement un système avec une approche d'analyse. Pour chaque point de vue du système à évaluer (ex : dynamique, LCA, sûreté de fonctionnement), un modèle d'analyse différent doit être construit (les processus d'analyse s'enchaînent ainsi, par exemple on évaluera le comportement dynamique puis on rélisera une LCA). Le modèle d'analyse décrivant l'architecture est un modèle de comportement du système uniquement et est construit à partir d'un modèle type boîte blanche ou boîte noire ainsi que de formalismes et outils d'analyse (ex : Matlab, Modelio, Altarica (sûreté de fonctionnement)). Après simulation, les performances réelles sont ensuite évaluées par rap-

1. industriels : MagicDraw (No Magic), Cameo Systems Modeler (No Magic), EA Sparx, Windchill Modeler (anciennement Integrity Modeler) et sous forme de plug-in : ParaMagic, Melody et Solvea.

port à la spécification des exigences. Si l'architecture est admissible, on peut poursuivre vers la conception détaillée, sinon il faut améliorer l'architecture. On évalue ainsi itérativement si une architecture est admissible depuis l'espace des solutions vers celui du problème. Nous pourrons parler dans ce cas d'une approche orientée solutions.

Amélioration d'architectures et optimisation

Les modifications d'architectures existantes visent à obtenir une amélioration de ses performances réelles vis-à-vis de la spécification des exigences. L'amélioration des performances réelles d'une architecture consiste à faire évoluer sa structure (ex : on choisira un type de composants différent) et peut être manuelle (espace des connaissances) ou automatisée via des outils et techniques de simulation numérique et d'optimisation (ex : cas en optimisation multi-disciplinaire, dite MDO, lorsque les modèles du système définis sont multi-disciplinaires [78]). Le problème de conception est alors généralement formulé comme un problème de conception optimal mono- ou multi-critères (conception par optimisation). Au sein du GE, de nombreux travaux s'intéressent ainsi au développement de techniques de simulation / optimisation pour améliorer une architecture [79–83]. Pour la pré-conception par analyse, deux catégories de techniques sont employées pour la résolution des problèmes : déterministes et évolutionnaires que nous présenterons succinctement ci-après à titre d'information. Dans le cadre de la pré-conception, nous ne mentionnerons pas les techniques de simulation / optimisation avec des modèles de type boîte noire (*black box optimization*).

Les techniques déterministes

Les techniques déterministes de simulation / optimisation sont généralement dédiées aux problèmes continus [84–86] et ne peuvent donc être employées que si le critère est au moins continu voire dérivable. Mais les problèmes de conception en ingénierie des systèmes physiques sont souvent mixtes. De plus, la plupart ne réalisent qu'une exploration locale (bien que celles-ci peuvent être étendues pour être globales en Derivative Free Optimization (DFO) [87]). Elles sont cependant rapides pour obtenir des résultats.

Bien que très efficaces pour les problèmes continus, les techniques Sequential Quadratic Programming (SQP) ou Mix Integer Linear Programming (MILP) ne sont adaptées resp. que pour les problèmes quadratiques et linéaires. Mais les problèmes de conception de systèmes physiques sont souvent non-linéaires. En conséquence, les problèmes qui ne sont pas respectivement quadratiques ou linéaires doivent être resp. transformés en quadratiques ou linéaires

pour être résolus avec ces techniques. La technique SQP requière également de transformer le critère ou les contraintes en fonctions de pénalités. La structure initiale du problème (non-linéarité voire inéquations) est altérée (si le modèle est de type boîte blanche), voire perdue (si le modèle est de type boîte noire), ce qui rend les modèles mathématiques et les problèmes de conception plus difficiles à comprendre (le sens physique du problème peut être altéré).

La simulation et l'optimisation des problèmes à variables discrètes sont moins bien gérée dans le cadre du Génie Electrique (GE). Les techniques comme SQP reposant sur la différentiabilité des contraintes (ou du critère) ne peuvent pas être employées sur les variables discrètes. Certains outils (ex : Matlab Optimization Toolbox [1]) ou travaux [88] considèrent que le problème discret ou mixte peut être transformé en un problème continu (relaxation continue). Le problème de conception optimal est ensuite résolu avec les techniques pour problèmes continus. Les solutions obtenues sont alors projetées sur le domaine discret. Une telle approche de conception présente plusieurs inconvénients. D'une part, même si les valeurs continues et discrètes du critère sont proches, les valeurs des Variables correspondantes peuvent être très différentes. D'autre part, la transformation du domaine d'une variable qualitative (ex : le type de matériau) n'aurait pas de sens physique. Mais surtout, cela présente le risque d'obtenir une solution qui aurait été en réalité non admissible (et non optimale) si le problème discret ou mixte avait été résolu sans relaxation continue puisque la vraie solution n'est pas l'arrondi au plus proche.

Des techniques de de simulation/optimisation spécifiques sont dédiées aux problèmes discrets. En conception par analyse, on retrouvera généralement les techniques évolutionnaires.

Les techniques évolutionnaires

Les techniques évolutionnaires (aussi dites évolutionnistes) telles que les algorithmes génétiques (Genetic Algorithm (GA) [89]) sont des métaheuristiques s'inspirant de la nature [2] pour approcher l'optimum global. Cependant, il ne s'agit que d'une approximation de l'optimum. De plus, les contraintes sont aussi prise en compte comme des fonctions de pénalités au sein de la fonction objectif et la structure initiale du problème (non-linéarités voire inéquations) est souvent perdue. Les GA sont parfois employés pour la résolution par optimisation des problèmes de conception en GE [22, 90, 91] notamment le NSGA-II [92].

1. utilisée notamment dans http ://l2ep.univ-lille1.fr/come/benchmark-transformer.html pour résoudre un problème mixte avec une technique SQP.

2. théorie de l'évolution des êtres vivants : on fait "évoluer" une variable dont la valeur est choisie aléatoirement.

Ces métaheuristiques sont en effet capables de traiter les problèmes à variables discrètes et mixtes. En revanche, lorsque de nombreuses variables discrètes sont présentes et qu'un nombre d'évaluations important est nécessaire pour la fonction objectif, on observe généralement un faible taux de convergence ce qui peut représenter un frein considérable au choix de ces techniques [22].

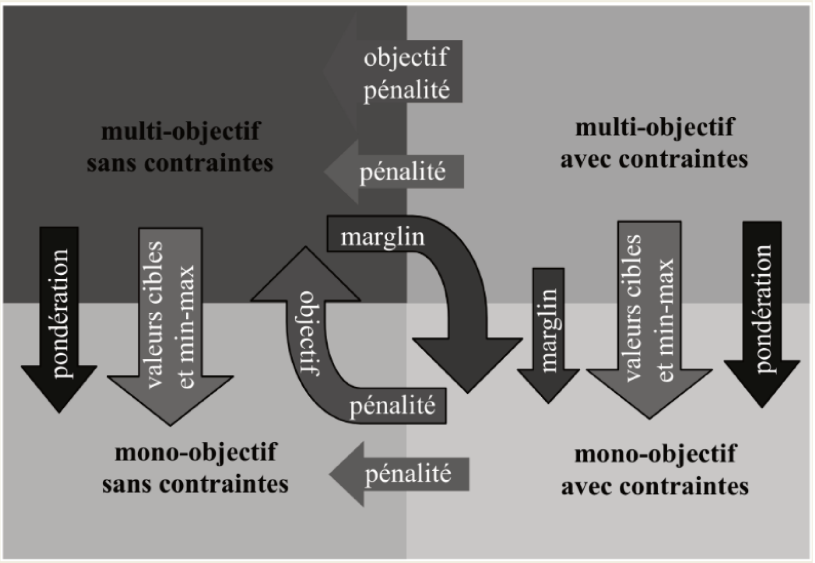
La conception demeure d'autant plus complexe dans la mesure où il n'existe pas de techniques générales permettant de résoudre tout type de problème. La résolution d'un problème de dimensionnement mixte non-linéaire et non-convexe sera ainsi traité différemment et plus difficilement que si le problème était linéaire et/ou convexe. Il est donc évident qu'il ne peut pas y avoir qu'une seule technique générale et efficace pour tout type de problèmes. Un problème de conception dans le cadre de l'analyse et de l'amélioration par techniques d'optimisation sera ainsi traité différemment selon [58] :

- le nombre de ddl (mono- ou multi-variables) ;
- le type des variables (continues ou discrètes) ;
- la linéarité du modèle mathématique (linéaire, quadratique, non-linéaire) ;
- la convexité : le nombre de minimums dans le domaine considéré ;
- le type de contraintes (égalité, inégalité, catalogue, conditionnelle, etc.) ;
- le nombre de critère à optimiser (mono- ou multi-objectifs).

De plus, les différentes techniques de transformations (pour passer par exemple d'un problème avec contraintes à un problème sans contraintes ou multi-objectifs à mono-objectif par exemple) rendent ces catégorisations plus floues (cf. Fig. 1.13). La complémentarité des techniques d'optimisation conduit de plus en plus la communauté optimisation du GE à procéder à une hybridation des techniques, par exemple en alliant une technique de recherche globale lente et peu précise lors de l'exploration de l'espace des solutions à une locale plus rapide et plus précise lorsque l'on se rapproche de l'optimum [58].

On peut noter cependant que concevoir selon une approche d'analyse présente les avantages suivants. Il est possible d'utiliser des modèles aussi bien de type boîte blanche que de type de boîte noire. La granularité des modèles type boîte blanche utlisés n'a priori pas de limites pour les outils d'analyse.

Figure 1.13 Les transformations de problèmes de conception optimales [58].



1.3.5 Limites de la conception par analyse et optimisation

Connaissance d'architectures initiales

À l'exception des techniques évolutionnaires, les techniques de simulation / optimisation reposent toutes sur la connaissance à priori d'architectures existantes du système à concevoir (il faut initialement une architecture pour commencer la simulation). C'est un des inconvénients principaux de la conception par analyse d'architectures.

Dégradation du problème

Pour l'ensemble des techniques de simulation/optimisation, la priorité est ainsi donnée à la résolution (obtention et rapidité d'obtention des résultats) plutôt qu'à la formalisation des problèmes de conception qui elle est dégradée (transformation des problèmes pour se ramener à des cas bien maîtrisés comme avec la technique SQP). Une approche d'analyse est ainsi limitée au niveau de la formalisation.

Le problème de conception n'est pas explicitement représenté, de manière externe au modèle (gros grains ou grains fins ou d'analyse). En conséquence, les contraintes structurelles ne sont pas exprimées (ex : pas de choix de composants depuis un catalogue). La modélisation de problème en GE est peu lisible et peu réutilisable. Aussi, comme seul le comportement du système est décrit, le dimensionnement se fait ainsi souvent selon un seul point de fonctionnement, ce qui présente un risque de sur- ou de sous-dimensionnement [42]. Finalement, la conception selon une approche d'analyse nécessitant de séparer les exigences

à évaluer (dynamique, environnementales, etc.), il est difficile d'avoir une vision globale du système.

La plupart des travaux en GE s'intéressent ainsi d'avantage à la résolution d'un problème plutôt qu'à sa formalisation [93, 26]. La pré-conception doit considérer de manière simultanée la formalisation ainsi que la résolution d'un problème de conception [88, 2].

Gestion des exigences contradictoires

La conception par analyse ne permet généralement pas de détecter les exigences contradictoires lorsque le nombre d'exigences à respecter est grand. Le risque de ne pas trouver de solution ou de travailler avec une solution non-admissible ou irréalisable est important.

Gestion des exigences non fonctionnelles de type 2

Si des exigences non fonctionnelles de type 2 (environnementales, sûreté, etc.) sont présentes, il n'y a aucune garantie qu'elles soient bien respectées dans les architectures proposées. Ces exigences ne sont pas prises en compte conjointement avec les autres exigences fonctionnelles et non-fonctionnelles de type 1 dans une approche d'analyse d'architectures. La conception doit ainsi être rallongée par d'autres processus ultérieurs. Par exemple, pour traiter un problème d'éco-conception d'un système physique, on réalisera une Life-Cycle Assessment (LCA) a posteriori comme dans [76].

Architectures peu innovantes

L'analyse d'architectures est parfaitement adaptée pour la conception détaillée et l'amélioration d'une architecture existante mais est moins pertinente pour générer des architectures nouvelles. Les seules aides à la disposition des concepteurs, aussi bien en industrie qu'en académie, sont alors leurs expertises et les règles métiers [2]. Mais les concepteurs débutants ne disposent pas nécessairement du recul qu'ont les concepteurs expérimentés pour le choix de ces architectures initiales.

Difficultés d'inversion des modèles directs

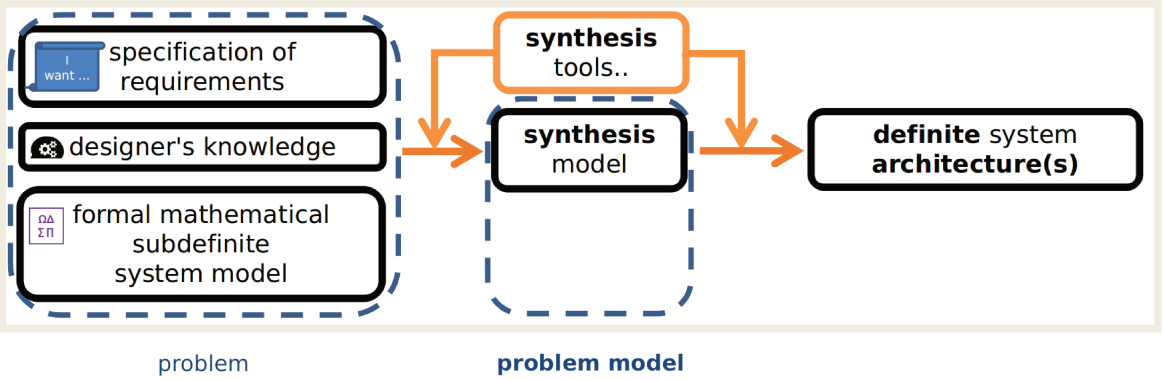
L'utilisation courante de l'analyse d'architectures pour la pré-conception peut s'expliquer par la difficulté voire l'impossibilité d'obtenir des modèles inverses par inversion des modèles directs.

Hormis les modèles de susbstitution (dits *surrogates* ou de surfaces de réponse) [94] établis à partir de résultats d'expériences (voire par la simulation), les modèles gros grains sont construits à partir de modélisations des phénomènes physiques (lois de conservation, lois constitutives, etc.) car le système réel n'existe pas encore en pré-conception. Ces modélisations physiques sont naturellement construites sous forme directe. Il est généralement très difficile voire impossible d'inverser un modèle direct comportant de nombreuses équations de manière symbolique [93]. Le modèle inverse résultant sera alors : soit insatisfaisant, soit "trop" gros grains (et donc relativement peu pertinent au regard du sens physique), ou soit que partiellement inversé. Ces modèles directs doivent alors être implicitement inversés par des techniques d'optimisation, ce qui peut expliquer leurs utilisations.

Le peu de modèles inverses que l'on trouve dans la littérature s'appuie également sur des modèles construits pour une application spécifique, sont de type boite noire ou sont empiriques (pour tenter d'augmenter la généricité des modèles [95]). Par exemple pour une machine électrique, la modélisation d'un phénomène thermique sera ramenée à une équation à coefficients variables à travers un échauffement approximatif [24, 25]. Ces modèles sont alors soit non réutilisables dans le premier cas car l'accent est mis sur la résolution, soit trop approximatifs. Les hypothèses sur les phénomènes négligés et pris en compte ne sont également pas souvent explicitées. En conséquence, le processus de conception de systèmes physiques repose sur les modèles principalement disponibles qui sont ceux à causalité directe et donc pré-disposés pour l'analyse d'architectures. Nous verrons au chapitre suivant que la Programmation Par Contraintes (PPC) peut fournir une réponse à l'inversion de modèles de type boîte blanche gros grains algébriques mixtes et non linéaires.

Nous pensons qu'il existe deux raisons principales menant à l'utilisation très majoritaire d'une approche d'analyse en pré-conception (en particulier en GE). D'une part, la causalité des modèles de type boîte blanche ou boîte noire souvent directe. D'autre part, comme mentionné par [88], "la formalisation des problèmes n'a pour le moment pas suffisamment été identifiée et étudiée, [..] car la tendance dominante est de supposer que celle-ci est évidente pour les concepteurs [..] et que la réelle difficulté est seulement la résolution du problème" (traduit de l'anglais). Comme souligné par [88], nous considérons que cette supposition de la part des concepteurs est fausse puisqu'il faut également prendre en compte les temps et efforts de modélisation en plus de ceux de résolution.

Figure 1.14 Le processus de synthèse de systèmes.



Face aux limites énoncées précedemment concernant la conception par analyse d'architectures et optimisation, on peut se pencher sur les possibilités actuelles données par la conception par synthèse de systèmes.

1.4 Conception à travers la synthèse de systèmes

Synthèse de systèmes Lors de la conception d'un nouveau système, il semble naturel de penser que les valeurs des Variables structurelles (définissant sa géométrie et sa configuration) soient les sorties d'un processus utilisant la spécification des exigences en entrée [2, 95]. C'est le processus de synthèse de systèmes (cf. Fig. 1.14). L'objectif n'est pas de trouver dès le départ une unique architecture définitive autour de laquelle se poursuivra la conception du système. On cherchera plutôt à avoir un aperçu du panel de solutions dont on dispose en concevant le système. On conçoit directement un système avec une approche de synthèse. L'existence (ou la non-existence) d'architectures admissibles est assurée au plus tôt durant la conception. La synthèse repose sur la formalisation du problème plutôt que des architectures comme par exemple dans [96]. Une approche reposant sur la synthèse de systèmes peut être vue comme une approche orientée problème.

Un modèle de synthèse est une représentation formelle d'un problème de conception. Il sera dit *formel* car construit avec un formalisme. Un tel modèle doit représenter la spécification des exigences, le système sous-défini ainsi que son environnement. Ces modèles de synthèse formels sont construits à partir de formalismes dédiés au sein d'environnements de modélisation et sont utilisés par un solveur qui dans l'idéal permettrait de résoudre les problèmes de conception à variables mixtes et de type génération d'architectures. On pourrait également envisager des problèmes avec prise en compte d'exigences non-fonctionnelles de

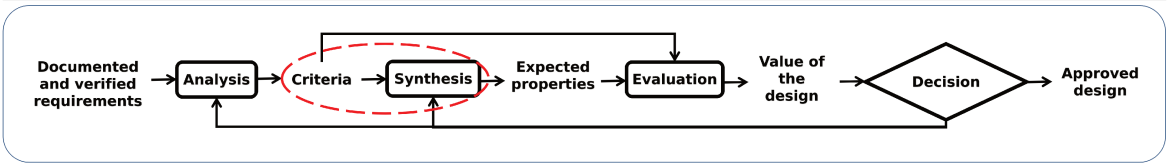
type 2 pour traiter des problèmes comme l'éco-conception, au cours d'un unique processus de conception.

Une approche de synthèse permet de formaliser directement le problème, ce qui permet au concepteur d'avoir une meilleure vision de celui-ci qu'avec une approche d'analyse. L'inconvénient principal se trouve au niveau du type de modèle que l'on peut prendre en compte au départ, qui sont quasi-uniquement de type boîte blanche gros grains algébriques comme nous le verrons au chapitre 3. En revanche, ces modèles sont relativement rapides et il est possible d'exprimer des contraintes globales. L'autre inconvénient d'une approche de synthèse est qu'il faut disposer et maîtriser de techniques assurant l'admissibilté d'une architectures. En GE, un problème n'est quasi jamais résolu selon une approche de synthèse, ie. reposant sur des formalismes, outils et techniques de synthèse. Les raisons semblent être la difficulté voire l'impossibilité d'inversion des modèles directs type boîtes blanches ainsi que le manque de formalismes et outils pour formaliser et résoudre un problème de conception explicitement, en particulier si celui-ci est mixte et non-linéaire. Quelques travaux font office d'exceptions [97, 88, 98, 25, 24].

1.5 Conclusion

Nous avons relevé 3 verrous scientifiques auquels ces travaux de thèse ont voulu répondre. Tout d'abord, l'approche de synthèse n'ayant quasi jamais été utilisée en GE et au regard des freins liés à son utilisation en pratique, peut-on l'appliquer pour la pré-conception en GE ? Ensuite, la modélisation de problème en GE étant peu lisible et peu réutilisable, comment en améliorer la généricité et le niveau d'abstraction ? Enfin, les avantages d'une approche d'analyse étant les inconvénients de celle de synthèse, peut-on allier les deux pour concevoir en GE ? Nous proposons dans ces travaux de thèse une approche de pré-conception de systèmes physiques complexes comme basée sur la Fig. 1.15. Cette approche traite en particulier des problèmes non-linéaires, mixtes de dimensionnement, configuration et de génération d'architectures, à exigences fonctionnelles et non-fonctionnelles (notamment de type 2 environnementales) avec catalogue, de systèmes structurés et décomposables. Les différents points de cette approche générale basée sur la synthèse de systèmes vont être présentés pas à pas dans les prochains chapitres. Le Chapitre 2 cherche à répondre au premier verrou en s'intéresserant à la construction de l'approche du point de vue de la résolution dans le cadre de la synthèse. Le Chapitre 3 cherche à répondre au second verrou en s'intéresserant à la construction de l'approche du point de vue de la formalisation dans le cadre de la synthèse.

Figure 1.15 Vision globale de l'approche proposée (schéma adapté de [99, 100]).



Le Chapitre 4 cherche à répondre au troisième et dernier verrou en s'intéresserant principalement à l'évaluation et à la validation des architectures admissibles par raffinement dans le cadre de l'analyse.

Chapitre 2

Programmation par contraintes pour la synthèse de systèmes complexes

2.1 Introduction : approche proposée pour la synthèse

Les 5 piliers de la formalisation des problèmes de conception

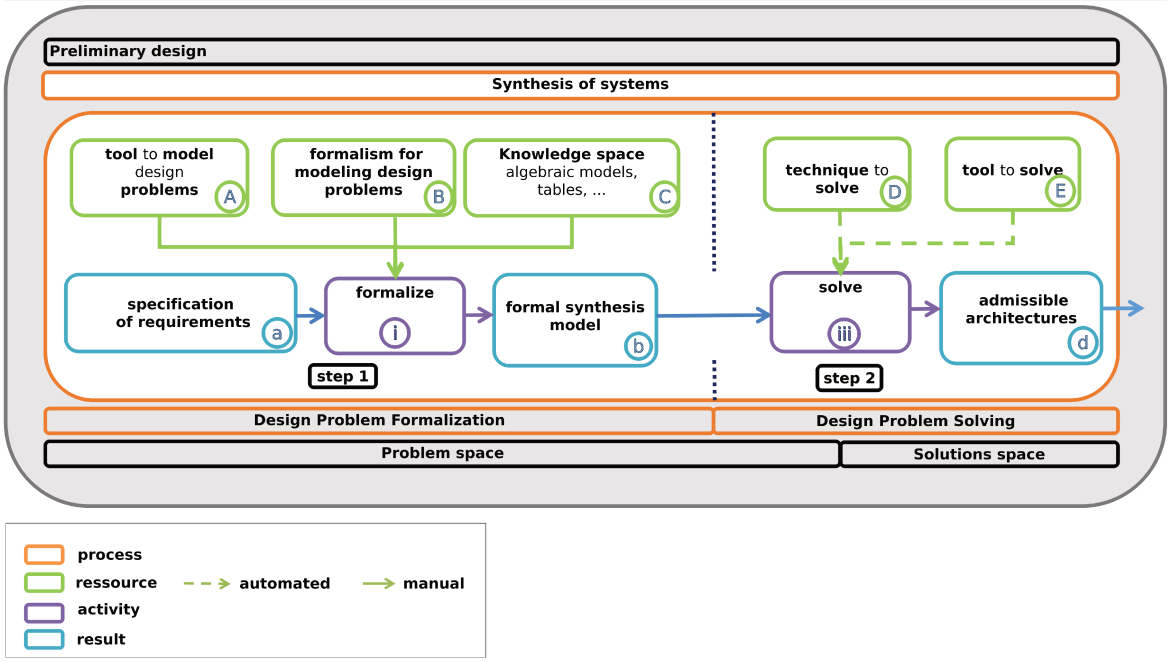
Nous proposons de définir 5 piliers sur lesquels devrait reposer la synthèse de systèmes et qui doivent être intégrés au sein d'un même environnement de travail :

1. un formalisme supportant le processus de synthèse de systèmes et ses activités de conception. Il servira à construire les modèles de synthèse formels des problèmes de conception ;
2. une approche orientée problème ;
3. un outil logiciel de formalisation de problèmes ;
4. une technique de résolution ;
5. un outil logiciel de résolution implémentant la technique précédente.

La formalisation d'un problème de conception

Le cadre à considérer pour la formalisation d'un problème de conception d'un système physique est décrit ci-après. La formalisation consistera à traduire la spécification des exigences, le modèle gros grains algébrique (donc de type boîte blanche, représentant le système sous-défini et son environnement) et toute autre relation (comme les catalogues de composants) en un modèle de synthèse formel du problème.

Figure 2.1 Une approche orientée problème selon le processus de synthèse de systèmes.



Les concepteurs peuvent avoir à restreindre le domaine de variation de certaines ddl pour des raisons de contraintes de réalisation technologique par exemple, réduisant ainsi l'espace des solutions. La formalisation devrait être réalisée à l'aide d'un formalisme dédié à la spécification de problème de conception. Il doit être suffisamment expressif pour pouvoir exprimer une variabilité totale au niveau structurel et comportemental (suffisamment pour pouvoir spécifier un problème de génération d'architectures). La pré-conception de systèmes physiques complexes selon le processus de synthèse de systèmes peut être envisagée dans un premier temps comme décrit et décomposé sur la Fig. 2.1. Nous recherchons des formalismes, ouils et techniques permettant de supporter la synthèse. Le processus de synthèse de systèmes serait alors découpé en 2 sous-processus : la formalisation du problème de conception (step 1) et sa résolution (step 2). Les résultats de ces deux sous-processus seraient resp. un modèle de synthèse formel (b) et l'ensemble des architectures admissibles du problème (c). Les rebouclages entre les résultats de chaque sous-processus et les activités ne sont pour l'instant pas considérés mais le seront au chapitre 4.

Choix de modèles pour la pré-conception En pré-conception, le bon niveau de détails doit être choisi pour représenter le système sous-défini et son environnement. La synthèse devrait reposer sur les modèles gros grains [88, 2]. Cependant cette représentation doit être

suffisamment précise pour représenter correctement le fonctionnement du système, de ses constituants et leurs structures. Le sens physique doit être conservé. Du point de vue de la formalisation, il faut que les concepteurs puissent comprendre le fonctionnement global du système et les interactions entre constituants, ce qui est plus envisageable en utilisant ce type de modèle. Du point de vue de la résolution, le modèle gros grains doit permettre d'évaluer l'existence d'architectures admissibles relativement rapidement, ou du moins en des temps de résolution plus courts qu'avec un modèle grains fins. En pré-conception, un modèle gros grains algébrique sera également préférable car celui-ci sera le plus inversible possible [42]. Les modèles gros grains sont rapides mais leurs limites apparaissent lorsque les systèmes présentent des géométries complexes ou des comportements fortement non-linéaires [2]. Le modèle grains fins serait plus adapté à la conception détaillée pour décrire le comportement des constituants du système.

Etapes proposées pour la synthèse de systèmes

Step 1 Le sous-processus de formalisation du problème de conception consisterait en une activité de conception physique : formaliser le problème de conception (i). Celle-ci serait réalisée à l'aide de 3 ressources : l'espace des connaissances (C), un formalisme de spécification de problème de conception (B) et un environnement de formalisation de problème (A) pour transformer le modèle naturel du problème qui est la spécification des exigences (a) en un modèle de synthèse formel (b). Le modèle naturel du problème serait lui-même le résultat du processus de définition des exigences précédant la pré-conception.

Step 2 Le sous-processus de résolution du problème de conception consisterait en une activité de conception physique : résoudre le problème de conception (ii). Celle-ci serait réalisée à l'aide de 2 ressources : une technique de résolution et un outil (logiciel) implémentant le précédent, pour transformer le modèle de synthèse formel en un ensemble d'architectures admissibles.

Dans ce second chapitre, nous cherchons à répondre au premier verrou : l'approche de synthèse n'ayant quasi jamais été utilisée en GE et au regard des freins liés à son utilisation en pratique, peut-on l'appliquer pour la pré-conception en GE ? Pour concevoir par approche de synthèse, la Programmation Par Contraintes (PPC) semblait adaptée. Ce second chapitre propose ainsi d'explorer les formalismes, outils et techniques de la PPC par une approche de synthèse de systèmes pour le GE.

2.2 La programmation par contraintes en conception

La première section de ce second chapitre propose une introduction à la PPC [101–103] à destination de l'Ingénierie des Systèmes (IS) et en particulier du Génie Electrique (GE). La PPC s'intéresse à la formalisation et la résolution de nombreux problèmes de dimensions importantes de manière efficace. La PPC permet une expression acausale des contraintes via la formulation en Constraint Satisfaction Problem (CSP) [104, 101, 103]. C'est donc un cadre qui s'adapte naturellement au point de vue acausal et qui peut être interessant à explorer dans la construction de notre approche de conception puisque les modèles gros grains n'ont pas à être inversés.

La PPC est dédiée au départ à la résolution de problèmes mathématiques discrets comme rencontrés en Intelligence Artificielle [104–107] et en Programmation Logique [108]. Ces problèmes combinatoires sont par exemple ceux de planification de tâches, d'ordonnancement ou encore de coloration de graphes [103].

La PPC en conception est historiquement dédiée à la résolution des problèmes combinatoires discrets en Recherche Opérationnelle (RO) [109]. La PPC est assez récente en conception. La raison principale étant que les retours sur investissement des problèmes de conception (qui sont généralement mixtes ou continus) sont plus difficiles à évaluer que pour les problèmes de la RO. Son utilisation pour la conception est plus répandue en Ingénierie des Logiciels (IL) notamment pour les systèmes embarqués [46]. En conception des systèmes physiques, elle n'a émergé que depuis une vingtaine d'années [110–112] tout d'abord pour traiter des problèmes discrets [48] puis continus [113, 114, 96].

Les travaux de recherches liés à la résolution des problèmes de conception mixtes en PPC sont assez récents et peu nombreux [2, 46, 47, 115]. On peut noter que l'utilisation de la PPC en conception reste assez rare en GE. Il ne s'agit également que d'une formulation CSP des problèmes et de travaux axés principalement sur la résolution [24–26]. La PPC permet également d'adresser des exigences fonctionnelles et non-fonctionnelles. La prise en compte d'exigences environnementales a notamment été tentée à l'aide de la formulation CSP [76, 116, 117].

2.3 Formalisation d'un problème de conception à l'aide d'un CSP

La formalisation d'un problème consistera à le spécifier sous forme de CSP. C'est en effet à l'origine la seule manière de formaliser un problème en PPC. Un CSP est défini par trois composantes X, D et C tel que [101, 104] :

- $X = \{x_1, x_2, ..., x_n\}$ est un ensemble fini de n Variables, avec n un entier ;
- $D = \{d_1, d_2, ..., d_n\}$ est un ensemble fini de n domaines d_i, chacun étant associé à une variable x_i tel que $\forall\, i \in \{1, \cdots, n\},\ x_i \in d_i$. Chaque domaine peut être continu ou discret. Un domaine continu est un intervalle. Il est composé d'une infinité de valeurs réelles et peut être borné ou non. Un domaine discret est une énumération d'un ensemble fini de valeurs réelles ou discrètes (qualitatives (ex : une couleur) ou quantitatives (ex : un entier)).
- $C = \{c_1, c_2, .., c_p\}$ est un ensemble fini de p contraintes, avec p un entier. Chaque c_i est une relation appelée contrainte portant sur un sous-ensemble X_i de X tel que $\forall\, i \in \{1, \cdots, p\},\ \exists!\, X_i \subseteq X, c_i(X_i)$.

Contraintes en PPC

Un CSP peut être vu comme un graphe bipartite entre l'ensemble des contraintes et l'ensemble des domaines [118]. Dans le cadre des CSP, une contrainte fait référence à une relation d'égalité ou d'inégalité impliquant une ou plusieurs variables. Les contraintes peuventt correspondre à tout type de relations mathématiques : linéaires, quadratiques, booléennes, etc. Elles permettent de spécifier les combinaisons de valeurs admissibles entre les variables y prenant part et ainsi de réduire l'espace des solutions lors de la résolution. On peut les classifier en fonction du nombre de variables qui y sont impliquées : unaire, binaire ou n-aires, faisant intervenir respectivement une, deux ou n variables. Les contraintes peuvent également être classées en fonction de leur type au sens mathématique [21, 114] :

- logique : $z \&\& y = 1$;
- arithmétique : $7x - 3y = 2z$;
- non-linéaire : $e^y + ln(x) > 4$;
- explicite : sous la forme de n-tuplets de valeurs possibles : $(x, y) = (0, 5)$;
- par morceaux : le domaine d'une variable se verra associé, sur différentes parties, une contrainte différente ;

— globales comme la contrainte catalogue qui est spécifiée sous forme de catalogue de composants ou la contrainte *alldiff(x,y,z)* qui contraint les variables x, y et z à avoir des valeurs différentes [119].

Les contraintes définissent alors le type de CSP. Cette distinction est primordiale car les techniques de résolution destinées à la résolution d'un CSP discret sont différentes de celles utilisées pour un CSP continu. Les CSP mixtes sont résolus en employant conjointement les techniques associées resp. aux CSP discrets et aux CSP continus. Ainsi, les :

— CSP discrets correspondent aux CSP dont toutes les contraintes opèrent sur des variables discrètes.
— CSP continus ou numériques correspondent aux CSP dont toutes les contraintes opèrent sur des variables réelles.
— CSP mixtes : correspondent aux CSP dont au moins une contrainte opère sur des variables réelles et discrètes.

2.4 Résolution d'un CSP

La résolution d'un CSP cherche à s'assurer de la solvabilité ou non de celui-ci. Un CSP est résolu lorsque chaque variable de X est instanciée, ie. une valeur admissible lui a été affectée. Les couples *variables-valeurs* forment une solution admissible s_i. Résoudre un CSP consiste à trouver, si il existe, l'ensemble des solutions admissibles. Pour cela, on cherchera à réduire l'espace des solutions en identifiant et rejetant les couples *variables-valeurs* non-admissibles. La résolution d'un CSP peut être décomposée en deux grandes étapes : la propagation des contraintes et l'exploration de l'espace des solutions. L'algorithme de résolution a ainsi la possibilité de les réaliser de manière consécutive ou alternée.

La propagation des contraintes [120, 103] est un schéma d'inférence qui permet de réduire la taille du domaine d'une variable sur laquelle porte une contrainte. Cette première inférence va permettre ensuite de restreindre le domaine d'une autre variable et ainsi de suite. L'objectif de cette étape est de diminuer les temps de résolution associés à la phase d'exploration qui est plus coûteuse, en se servant plus efficacement des contraintes en amont [107]. La propagation peut prendre un intérêt tout particulier pour vérifier relativement rapidement qu'un CSP n'admet pas de solutions admissible. C'est elle qui permet de réduire les domaines efficacement et peut suffire à résoudre un CSP. L'exploration complète la propagation et permet d'obtenir les solutions admissibles.

L'exploration est généralement employée avec des heuristiques [107, 121, 111, 103]. Une heuristique est une technique informelle, issue de l'expérience, qui sera utilisée pour choisir la variable qui sera instanciée en premier ou encore comment sera parcouru son domaine. C'est donc une technique issue de l'espace des connaissances. Le choix de l'heuristique dépendra du problème et il faudra alors adopter la bonne stratégie de résolution. Elle peut être appliquée avant ou pendant la propagation des contraintes ou l'exploration de l'espace des solutions.

Nous présenterons la résolution des CSP de manière chronologique : discret, continu et mixte. Chaque partie sera organisée ainsi : techniques de propagation, heuristiques et technique d'exploration. Pour les heuristiques, il existe celles relatives aux choix de variables (nommées ci-après *d'ordonnancement des variables*) et celles de choix de valeurs (nommées ci-après *d'ordonnancement des valeurs*). Nous nous intéressons aux travaux de la communauté PPC dans ce chapitre.

2.4.1 CSP discret

Propagation sur domaines dicrets

La propagation des contraintes sur les domaines discrets existe sous plusieurs formes (comme la node-consistance ou la path-consistance [107]). La plus utilisée est l'arc-consistance [122] qui est la plus rentable en termes d'efficacité / temps de calculs (l'algorithme AC3 [107] étant le plus connu).

Arc-consistance : un domaine d_i sera arc-consistant si toutes ses valeurs v_i satisfont l'ensemble des contraintes binaires dans lesquelles est impliquée la variable x_i. Un CSP sera arc-consistant si chaque domaine est arc-consistant avec toutes les autres variables. Il est possible que l'arc-consistance n'ait aucun effet de réduction sur le domaine d'une variable, ce qui est le cas si celui-ci est déjà arc-consistant. L'arc-consistance peut suffire au filtrage des domaines de définition voire à trouver une solution admissible pour un CSP discret [103]. Mais pour d'autres, elle ne parviendra pas à réduire suffisamment les domaines. Cela est dû au fait qu'elle ne vérifie la satisfaction de contraintes qu'entre deux Variables.

Recherche de solutions pour CSP discrets

Les heuristiques d'ordonnancement de variables Il existe de très nombreuses heuristiques d'ordonnancement de variables et quelque unes sont présentées ci-après (les plus

connues ou utilisées dans la suite du manuscrit). Par exemple, l'heuristique *fail-first* [123] aussi appelée *minimum-remaining-value* consiste à choisir la variable de plus petit domaine d_i. On estime intuitivement que c'est celle qui aurait le plus de chances de causer un échec au plus tôt lors de la recherche. Cette heuristique est généralement plus efficace qu'une sélection "statique" de variables à tour de rôle (selon leur position dans X) ou qu'une sélection aléatoire comme cela est le cas pour de nombreux solveurs [103]. En fonction du problème, la *fail-first* peut s'avérer très efficace. Elle est en générale mieux adaptée aux domaines discrets. Un exemple d'adaptation du *fail-first* est l'heuristique *Brelaz* [124].

L'heuristique *most-constrained-variable* [102] repose aussi sur un ordonnancement "intuitif" de Variables. Ce sera la variable la plus contrainte qui sera choisie en premier. L'heuristique *degree* sélectionne les Variables présentes dans le plus grand nombre de contraintes. Elle est notamment utilisée lorsque les domaines sont de même largeur, cas où la *fail-first* est inefficace. Cette heuristique est en général moins opérante que la *first-fail* mais peut être utile en association avec cette dernière [103].

Les heuristiques d'ordonnancement de valeurs Pour un CSP discret, les heuristiques d'ordonnancement de valeurs ne sont en général pas utilisées. Pour chaque valeur d'un domaine, une nouvelle branche sera généralement créée puis une valeur affectée à la variable de manière aléatoire ou selon sa position dans le domaine. Il est aussi possible de ne créer des branchements que pour une partie des valeurs et d'en créer une autre pour l'ensemble des valeurs restantes. On peut imaginer que pour les problèmes de conception de taille importante, cela peut s'avérer moins efficace qu'une heuristique d'ordonnancement de valeurs [103].

L'exploration pour un CSP discret repose sur l'Exhaustive Enumeration (EE) [125]. Lors de l'EE, on teste successivement les valeurs d'une variable (on génère un couple *variable-valeur* puis on le teste). Si une valeur est consistante (elle respecte toutes les contraintes), celle-ci la lui est affectée. On ré-effectue le teste sur une autre variable, jusqu'à résolution ou échec. Toutes les combinaisons possibles sont ainsi testées. Généralement, on essaiera de limiter le plus possible l'emploi de l'EE en se reposant sur la propagation. Le principe de résolution combinant propagation et exploration est le suivant :

1. utilisation d'une heuristique d'ordonnancement de variables, qui sera en pratique la *fail-first* ou la *most-constrained-variable* ;
2. utilisation d'une heuristique d'ordonnancement de valeurs ;
3. application d'une technique de propagation ;

En fonction du résultat de la propagation, on procédera ou non à l'exploration :

4. (a) si toutes les variables ont été instanciées, alors le CSP est résolu ;
   (b) s'il reste des variables non instanciées, alors on retourne à l'étape 1 ;
   (c) si une variable présente un domaine vide, alors c'est un échec. Dans ce cas on effectue un retour en arrière (backtrack) sur les points de choix précédents.
5. fin.

2.4.2 CSP continu

Problème de représentation des nombres réels

Comme un nombre réel est représenté sur un nombre fini d'octets (souvent 4 ou 8), tous les nombres réels ne peuvent pas être représentés. Un nombre réel sera arrondi au nombre flottant le plus proche. L'ensemble des nombres flottants est inclut dans celui des réels. La norme IEEE 754 [126] définit un nombre flottant f_p par un bit de signe, une mantisse m et un exposant E : $f_p = (-1)^s \cdot m \cdot 2^E$. De part cette représentation appoximative, les calculs sur les nombres réels donnent des erreurs d'arrondis. Les techniques de propagation sur domaines discrets ne peuvent donc pas être employées. La propagation sur domaines continus repose ainsi généralement sur la Hull- ou la Box-consistance [18, 122, 127, 102] et l'exploration sur une technique de Branch and Prunes (BP) [128].

Propagation sur domaines continus

L'arithmétique des intervalles L'arithmétique des intervalles [129] est une technique qui permet de contourner le problème des erreurs de calcul sur nombres réels. Elle repose sur l'encadrement de la valeur exacte d'un réel x par deux nombres flottants qui constituent la borne maximum $\overline{x}$ et la borne minimum $\underline{x}$. Les calculs sont réalisés sur l'intervalle $[\underline{x}, \overline{x}]$ ainsi formé plutôt que directement sur les nombres réels. $[\underline{x}, \overline{x}]$ est un sous-ensemble connecté et fermé de $\mathbb{R}$. L'arithmétique des intervalles définit ainsi quatre opérateurs arithmétiques de base $\{+, -, *, \div\}$. Si $[\underline{x}, \overline{x}]$ et $[\underline{y}, \overline{y}]$ sont deux intervalles, les quatres opérations arithmétiques de base sont :

- $[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$;
- $[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$;
- $[\underline{x}, \overline{x}] * [\underline{y}, \overline{y}] = [min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})]$;
- $[\underline{x}, \overline{x}] \div [\underline{y}, \overline{y}] = [min(\underline{x} \div \underline{y}, \underline{x} \div \overline{y}, \overline{x} \div \underline{y}, \overline{x} \div \overline{y}), max(\underline{x} \div \underline{y}, \underline{x} \div \overline{y}, \overline{x} \div \underline{y}, \overline{x} \div \overline{y})], 0 \notin [\underline{y}, \overline{y}]]$;

Les notations suivantes sont également définies [129, 130] :

— $mid([\underline{x}, \overline{x}]) = (\underline{x} + \overline{x})/2$, le centre de l'intervalle ;

— $w([\underline{x}, \overline{x}]) = \overline{x} - \underline{x}$, la largeur de l'intervalle ;

Le calcul par intervalles Le calcul par intervalles est une extension de l'arithmétique des intervalles [131]. Il inclut notamment l'ensemble des opérateurs algébriques usuels (quadratiques, trigonométrique, logarithmiques, etc). De plus, le calcul par intervalles autorise la division par un intervalle contenant un zéro [132, 131].

Exemple : le calcul sur l'intervalle $[1.2, 3]$ par la fonction q définie par $q(x) = x + \log(x)$, avec $x \in [1.2, 3]$, donnera :
$q([1.2, 3]) = [1.2, 3] + \log([1.2, 3])$
$q([1.2, 3]) = [1.2, 3] + [0.079181246, 0.477121255]$
soit $q([1.2, 3]) = [1.279181246, 3.477121255]$.

Il existe de nombreuses techniques de propagation sur domaines continus. Toutes reposent notamment sur les propriétés de contraction (ie. de réduction) des domaines. Les deux principales techniques sont la Hull-consistance (HC) et la Box-consistance (BC) [127].

La HC [120] est une relaxation de l'arc-consistance. Elle sera présentée ci-après car c'est la technique qui a été principalement utilisée pour résoudre les problèmes de conception présentés dans ce chapitre. Un intervalle sera réduit par HC en deux étapes : par *forward evaluation* puis par *backward propagation*. Ces deux mécanismes opèrent sur une représentation sous forme d'arbre d'une contrainte continue. Chaque feuille correspond à une variable. La racine correspond à un opérateur arithmétique, ie. à l'intersection entre le membre de gauche et celui de droite de la contrainte. Un algorithme très employé qui implémente la HC est le HC4Rev [133].

Exemple (HC) On considère le CSP suivant défini par deux Variables, deux domaines et une contrainte : $\text{CSP}(X, D, C) = (\{x_1, x_2\}, \{[-4, 8], [0, 35]\}, \{x_2 = 3.5 * x_1\})$.

Etape 1 : forward evaluation 1 L'expression $3.5 * x_1$ est tout d'abord évaluée par calcul par intervalle (cf. Fig. 2.2a) : $(3.5 * x_1) \in ([-4, 8] * [3.5, 3.5] = [-14, 28])$.

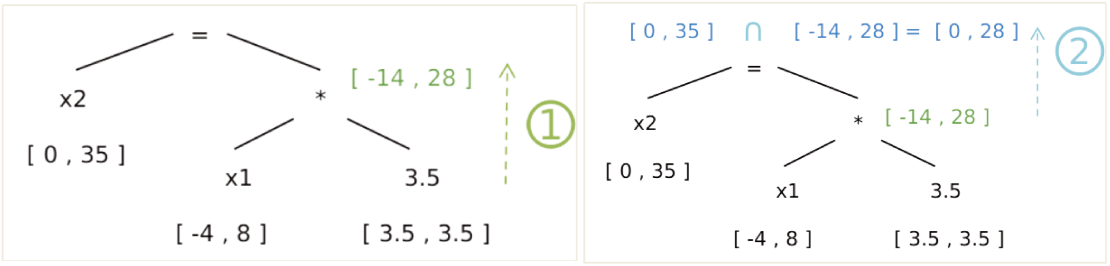
Etape 2 : forward evaluation 2 La contrainte est ensuite évaluée en utilisant le nouvel intervalle réduit (cf. Fig. 2.2b) : $x_2 \in ([-14, 28] \cap [0, 35] = [0, 28])$.

Etape 3 : backward propagation Le nouvel intervalle de x_2 permet de réduire celui de x_1 (cf. Fig. 2.3) :

$x_1 \in ([-4, 8] \cap ([0, 28] \div [3.5, 3.5]))$

$x_1 \in [0, 8]$.

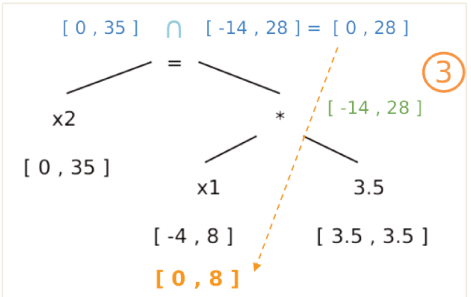
Figure 2.2 Hull-consistance (HC) - Forward evaluation.



(a) Forward evaluation 1.

(b) Forward evaluation 2.

Figure 2.3 Hull-consistance (HC) - Backward evaluation.



Limites de la HC : L'efficacité de la HC est limitée lorsqu'une variable apparaît plusieurs fois dans une contrainte arithmétique. Le calcul des intervalles est d'autant moins précis que le nombre de fois qu'apparaît la variable [102].

La Box-consistance (BC) permet d'obtenir une meilleure évaluation des domaines continus si une variable est présente plus d'une fois dans une même contrainte [120]. Si la variable n'apparaît qu'une fois, la BC devient équivalente à la HC [134] et il vaudra mieux privilégier cette dernière. La BC est en effet plus coûteuse en temps de calcul et présente des taux de convergence relativement bas. Un algorithme très employé qui implémente la BC est le BC4Rev [133].

Recherche de solutions pour CSP continus

L'efficacité des techniques HC ou BC peut être grandement améliorée quand elles sont employées pendant l'exploration de l'espace des solutions plutôt qu'avant celle-ci. On peut en effet estimer que l'utilisation seule des contraintes n'aurait pas été suffisamment efficace pour réduire un domaine. Une bissection de ce domaine par une technique de Branch and Prunes (BP) permettra alors de réduire le domaine et d'en explorer les deux sous-intervalles ainsi formés. Les algorithmes de BP sont des techniques de résolution déterministes. Une solution admissible sera toujours incluse dans un intervalle selon une précision donnée, malgré la précision de calcul finie du calculateur. Associé au calcul par intervalles, il est possible de trouver toutes les solutions admissibles du problème, en utilisant les bornes des domaines de chaque variable. L'implémentation de la technique BP n'est pas unique [129, 110].

Il existe de nombreuses heuristiques d'ordonnancement de variables ainsi que de valeurs pour les CSP continus, dont certaines sont listées ci-après (dont celles qui seront mentionnées dans la suite de ce chapitre).

Les heuristiques d'ordonnancement de variables L'heuristique *round-robin* (*en tourniquet*) applique un choix "statique" de variables, en les essayant les unes après les autres [135]. Elle est largement employée. L'heuristique *reduce-first* selectionne la variable de plus large domaine.

Les heuristiques d'ordonnancement de valeurs Parmis les heuristiques d'ordonnancement de valeurs employés pour résoudre les CSP continus, la heuristique *least-constraining-value* favorise la valeur de la variable qui laissera le domaine le plus grand possible aux autres variables, ie. celle qui sera la moins contraignante envers les autres variables. Elle peut être très utile dans le choix d'un sous-intervalle à parcourir après une bissection [103].

Le principe de résolution d'un CSP continu est le suivant : on alterne propagation et découpe des intervalles jusqu'à ce qu'une précision ϵ soit atteinte. ϵ correspond au critère d'ar-

rêt de l'algorithme. Un technique de résolution par BP est présentée Alg. 1 [114]. Le procédé de résolution d'un CSP continu est décrit ci-après :

1. création d'une liste de solutions vide L ;
2. définition d'une largeur minimale d'intervalle ϵ ;
3. utilisation d'une heuristique d'ordonnancement de variables, qui sera en pratique la *round-robin* ou la *reduce-first*. La variable x_i de domaine $d_i = [\underline{x}, \overline{x}]$ est choisie ;
4. si la largeur de d_i, $w(d_i)$, est strictement supérieure à ϵ, alors d_i est bissectionné en deux sous-intervalles $d_{i1} = [\underline{x}, mid(d_i)]$ et $d_{i2} = [mid(d_i), \overline{x}]$;
5. utilisation d'une heuristique d'ordonnancement de valeurs. Par exemple, on choisit d_{i1} ;
6. propagation des contraintes sur d_{i1} par HC ou BC ;
7. propagation à d'autres domaines.

En fonction du résultat de la propagation, trois alternatives sont possibles :

8. (a) si $\forall d_k \in D$, $w(d_k) \leq \epsilon$, alors le CSP est résolu. L'ensemble des intervalles d_k est ajouté à la liste des solutions L ;
   (b) si d_{i1} est vide, alors c'est un échec. On retourne à l'étape 5 et d_{i2} est choisi ;
   (c) si aucune solutions n'a été trouvée, alors l'algortihme s'arrête ;
9. fin.

Algorithme 1 Résolution d'un CSP continu par algorithme de BP [114].

```
1 : procedure BP(CSP(X, D, C),L)
2 :     require X, D, C
3 :     L = { }
4 :     D ← Prune(C, D)
5 :     if notEmpty(D) then
6 :         if OkPrecise(D) then
7 :             Insert(D, L)
8 :         else
9 :             (D1, D2) ← Split(D, ChooseVariable(X))
10 :            BP(CSP(X, D1, C),L)
11 :            BP(CSP(X, D2, C),L)
12 :        end if
13 :    end if
14 :    return L
15 : end procedure
```

2.4.3 Extension aux CSP mixtes

De manière générale, la résolution des CSP mixtes repose sur une énumération pour les variables discrètes et sur une bissection d'intervalles pour les variables continues. Les techniques de recherches pour CSP mixtes résultant d'une telle combinaison ne sont pas très répandues dans les travaux de la communauté PPC [136, 121] à laquelle nous nous intéressons dans ce chapitre [1].

Propagation des contraintes mixtes

Comme énoncé précédemment, nous considérerons qu'un CSP est mixte si au moins une contrainte n-aire est mixte. Cela n'exclut donc pas la présence de contraintes purement discrètes ou continues. Certains travaux étendent les algorithmes de HC4Rev et BC4Rev [133] pour la prise en compte d'expressions algébriques opérant à la fois sur des domaines continus et discrets [137].

Recherche de solutions des CSP mixtes

Il n'y a pas de stratégies de résolution propres aux CSP mixtes. Généralement, on empruntera les techniques de resolution des CSP discrets pour la gestion des variables discrètes et celles des CSP continus pour les variables continues (BP). Comme énoncé en introduction de ce chapitre, la formalisation d'un CSP adoptant un point de vue acausal, il n'y a donc pas de notion de variables d'entrées ni de sorties. La résolution n'a donc pas lieu de manière séquentielle, comme on le ferait en résolvant un problème par calcul symbolique. Dans ce cadre, l'utilisation des heuristiques sera déterminante sur la capacité à résoudre et les temps de calcul. C'est la limite principale de la PPC.

2.5 PPC et problèmes de conception optimale

2.5.1 Formalisation d'un problème de conception optimale

La PPC est initialement destinée à la formalisation et à la résolution de problèmes de recherche de solution(s) admissible(s). Elle peut cependant être adaptée pour traiter un pro-

1. Nous avons chosis de ne pas traiter des autres techniques de résolution des CSP dans les autres communautés comme celle des MINLP en optimisation mathématique numérique, car les intervalles y semblent quasi-toujours perdus avec les techniques de résolution associées, et pour réduire le champ d'étude (axée optimisation, mais nous nous intéressons principalement à la recherche de solutions admissibles dans ces travaux de thèse).

blème d'optimisation mono-critère [132]. Celui-ci est alors formulé sous forme de Constraint Satisfaction Optimization Problem (CSOP) par un quadruplet (X, D, C, f). f est le critère à optimiser et est exprimé comme une Variable définie sur un sous-ensemble de X. L'expression définissant f est utilisée comme une contrainte, afin de réduire les domaines lors de la propagation.

2.5.2 Résolution d'un problème de conception optimale

La résolution d'un CSOP (X, D, C, f) est réalisée par boucles d'amélioration à l'aide d'un algorithme de BP (appelé aussi *Branch and Bound* dans le cadre de l'optimisation [**??**]). Elle consiste en la résolution séquentielle de plusieurs CSP pour lesquels les bornes du domaine de f sont à chaque fois améliorées et actualisées. Par ailleurs, contrairement aux techniques d'optimisation de type gradient, il n'est pas nécessaire que f soit (continûment) différentiable. Un algorithme d'optimisation par BP [114] est présenté Fig. 2. Le principe de résolution est décrit ci-après, pour la minimisation de f :

1. définition d'une précision ϵ, d'un triplet (X, D, C) et d'un critère f de domaine $(d_f \in D) = [\underline{f}, \overline{f}]$;
2. définition d'un CSP par le triplet (X, D, C).

   Tant que la largeur de d_f, $w(d_f)$, est strictement supérieure à ϵ :

3. (a) ajout d'une nouvelle contrainte $c \in C$ au CSP : la valeur de f doit être strictement inférieure au point milieu de d_f tel que $f < mid(d_f)$. On explorera la partie gauche de d_f ;

   (b) résolution du CSP par BP ;

   (c) si le CSP est résolu, alors on substitue la nouvelle valeur de f inférée f_{val} à $\underline{f}$ tel que $d_f = [\underline{f}, f_{val}]$;

   (d) sinon c'est un échec, alors on retire la contrainte c de C.

   On actualise la borne inférieure de f à la valeur $mid(d_f)$.

   Le processus est réitéré mais cette fois-ci, on recherchera le minimum de f dans l'autre sous-intervalle de d_f, celui de droite (et réciproquement de gauche si l'intervalle droite échoue, jusqu'à ce qu'il n'y ait plus d'intervalle à explorer ou qu'une solution ait été trouvée).
4. fin.

Algorithme 2 Minimisation d'un critère avec la PPC [114].

```
1 : procedure OptimCSP((X, D, C))
2 :     require (X, D, C), f ∈ [f_min, f_max], ε
3 :     CSP ← (X, D, C)
4 :     while (f_max − f_min) > ε do
5 :         C ← C ∪ {f < (f_max + f_min)/2}
6 :         if a solution f_val is found for CSP then
7 :             f_max ← f_val
8 :         else
9 :             C ← C − {f < (f_max + f_min)/2}
10 :        end if
11 :    end while
12 :    return [f_min, f_max]
13 : end procedure
```

Suite à cette introduction sur la formalisation et la résolution des CSP, nous présentons maintenant les outils et langages permettant leurs implémentations en pratique. Nous présenterons trois cas de pré-conception de machines électriques que nous avons mené durant ces travaux de thèse. Ils seront illustrés par des outils et langages sélectionnés auparavant. Cela permettra d'appréhender au mieux les enjeux liés au couplage entre la formalisation et la résolution (optimale) de ces problèmes en PPC.

2.6 Outils et langages pour CSP mixtes

Contrairement à l'Ingénierie des Systèmes (IS), la PPC n'est pas une spécialité standardisée. Il n'y a donc pas de langages ni d'outils s'inscrivant dans une approche encadrée et normée de résolution. Une classification générale que l'on retrouve par exemple dans [47, 46] se positionne plutôt du côté de la résolution. On distingue trois formes d'implémentations d'outils pour CSP :

1. les solveurs de contraintes ;
2. les bibliothèques d'objets dédiées aux CSP qui sont écrites en C++, Python ou Java ;
3. les interpréteurs Constraint Logic Programming (CLP), qui utilisent des modèles écrits dans un langage de programmation logique de type Prolog (Prolog, ECLiPSe, B-Prolog ou encore EssencePrime) [138] ;

Dans le cadre de la conception de systèmes, on pourra rajouter une quatrième catégorie, plus axée sur la formalisation :

4. les formalismes dédiés aux CSP.

Le choix des formalismes et outils dépendra :

1. de l'expression du problème et du coût de la modélisation ;
2. du type de domaine : discret, continu ou mixte ;
3. du type de contraintes (spécialisées, catalogue, non-linéaires, etc.) ;
4. de la technique de résolution ;

Difficultés du choix de formalisme/outil

La séparation entre ces 4 catégories n'est pas stricte. Les bibliothèques peuvent être vues comme des formalismes car elles nécessitent généralement l'emploi d'un solveur tiers. On trouve également des formalismes *hybrides* (mi-formalisme, mi-solveur) qui nécessitent soit d'utiliser un solveur existant (interfaçage) soit d'en écrire un. C'est le cas par exemple du formalisme Constraint Handling Rules (CHR) [139]. D'un autre côté, les solveurs de contraintes sont soit génériques (non dédiés à la résolution des CSP), soit élaborés par des équipes de recherches pour des besoins spécifiques [47]. De plus, chaque solveur implémente différentes techniques pour la résolution. Pour exploiter toutes les capacités de ces solveurs, il faudra souvent en choisir un et s'y tenir afin de le comprendre et le maîtriser pleinement [47]. Pour les novices, ceci complique en pratique la pleine maîtrise de la PPC.

Aspects pris en compte pour le choix

Il existe un très grand nombre de formalismes et outils pour les CSP qui sont principalement discrets.

Pour une utilisation en conception, les formalismes doivent permettre une expression simple des CSP. On évite ainsi au plus les détails d'implémentations propres aux langages, ce qui permet de mieux évaluer les avantages et inconvénients de la formulation en CSP. Nous excluons ainsi les extensions "trop poussées" de formalismes comme les bibliothèques d'objets dédiées aux CSP et les interpréteurs CLP (qui seront abordées au Chapitre 3). Nous recherchons également des formalismes capables d'exprimer des contraintes mixtes, non-linéaires et globales. Finalement, un formalisme adéquat pour la conception doit permettre en plus la déclaration de variables déduites. Il n'y a en effet pas de notion de variables déduites avec la formulation CSP (on considère que seuls des ddl sont employés pour décrire le problème). On ne peut donc pas directement construire de modèles de synthèse. On ne peut

ainsi pas directement envisager l'utilisation des CSP pour la formalisation des problèmes de conception dans un processus de synthèse de système.

Pour une utilisation en conception, les outils doivent être intégrés avec un formalisme. La prise en main est alors simplifiée. Egalement, cela permet de réduire les problèmes de compatibilités entre formalismes et solveurs (notamment de version et disponibilités en fonction du stade de développement). Nous écartons donc les solveurs génériques tels que Gurobi [140], GAMS [141], Knitro [142], SCIP [143] ou encore CONOPT [144].

Nous avons ainsi retenu les 5 formalismes / solveurs suivants : a) OPL / CPLEX CP Optimizer, b) Minibex / IbexSolve, c) Minion Input Language / Minion, d) RealPaver / RealPaver, et e) CE / CE.

OPL / CPLEX CP Optimizer [145]

Le formalisme OPL est associé au solveur de contraintes CPLEX CP Optimizer commercialisé par IBM ILOG®. Ils sont dédiés aux problèmes combinatoires linéaires et quadratiques multi-objectifs de type planification de tâches. OPL ne permet pas la définition de contraintes non-linéaires autres que quadratiques. Il faudrait donc l'étendre pour pouvoir définir des contraintes mixtes, ce qui n'est pas notre objectif. OPL / CPLEX CP Optimizer ne seront donc pas utilisés pour illustrer notre démarche.

Minibex / IbexSolve [146]

Minibex est un formalisme utilisé pour spécifier des problèmes continus et non-linéaires (dont CSP). Les modèles écrits sont très intuitifs et se rapproche au plus de la formulation CSP avec une extension pour définir des constantes (d'autres extensions sont disponibles mais un modèle peut n'être décrit que par des constantes, variables, domaines et contraintes). Le solveur de contraintes associé IbexSolve alterne la HC pour la propagation et BP pour l'exploration. IbexSolve est basé sur un solveur générique écrit en C++. IbexOpt est l'optimiseur associé. Ils sont commercialisés par IBEX®. On peut considérer que IbexSolve (et IbexOpt) sont des solveurs mixtes à dominante continue, ce que nous justifions comme ci-après. Minibex / IbexSolve ne semblent pas permettre de traiter les CSP discrets et par extension mixtes officiellemnt de manière native (ce que nous recherchons à travers un formalisme et outil intégré). Des astuces sont utilisées pour la gestion des Variables et contraintes continues pour pouvoir résoudre des CSP mixtes. Une interface avec le solveur pour CSP

discret Choco a été créée à l'aide d'un plug-in Java [1], mais comme nous le mentionnerons Section 3.1.1, Choco ne semble pas très intuitif à utiliser (très proche d'un langage de programmation). C'est également une "hybridation" de solveurs, ce qui n'est pas ce que nous recherchons. Plus récemment, une discussion sur un forum [2] mentionne le développement de nouvelles contraintes "floor" et "integer" qui s'appuient sur un encadrement de l'entier. Il ne semble pas a priori que l'on puisse utiliser la contrainte catalogue (mais ces deux contraintes ne sont pas encore officiellement documentées). Minibex / IbexSolve ne seront pas choisis.

Minion Input Language / Minion [147]

Minion Input Language est le formalisme par défaut accepté par le solveur de contraintes Minion. Les CSP peuvent être modélisés en séparant Variables, domaines et contraintes. On peut aussi définir une ou plusieurs heuristiques d'ordonnancement de variables. Le langage a également été interfacé avec EssencePrime mais toujours pour problèmes combinatoires. Les modèles autorisent la définition de contraintes globales et arithmétiques mais uniquement sur des Variables discrètes. On ne peut pas y déclarer de Variables continues car le solveur Minion ne le permet pas. On ne peut donc pas traiter de CSP continus ou mixtes. Ainsi, Minion Input Language / Minion ne seront pas retenus.

RealPaver / RealPaver [148]

Le formalisme RealPaver permet de spécifier des problèmes continus. RealPaver a déja été utilisé pour la conception [21]. Un modèle RealPaver est séparé en constantes, variables et contraintes. Les contraintes non-linéaires sont également supportées. En revanche, RealPaver ne permet pas de spécifier de contraintes globales ni l'opérateur de différence. La propagation des contraintes est réalisée par HC ou BC et l'exploration de l'espace des solutions par BP. Comme RealPaver / RealPaver ne permettent pas de traiter les CSP discrets et par extension mixtes, ils ne seront pas non plus choisis.

CE / CE [149]

Le formalisme CE est dédié à la spécification des problèmes de conception mixtes et il est parmis les plus adaptés pour les concepteurs métiers. CE a notamment été utilisé pour la pré-conception de systèmes mécaniques [96] et également dans le cadre de l'éco-conception optimale d'un système de propulsion naval [116]. La formulation en CSP inclut en parti-

1. https ://choco-solver.org/docs/advanced-usages/ibex/.
2. https ://github.com/ibex-team/ibex-lib/issues/507 et https ://github.com/ibex-team/ibex-lib/issues/168.

culier les extensions suivantes : constantes et variables déduites (appelées *alias*). Les modèles peuvent être construits de manière structurée, en séparant également les ddl avec leurs domaines de définition, et contraintes de conception. Les contraintes comme la contrainte *catalogue*, logiques ou encore non-linéaires, peuvent aussi être définies. En revanche, les contraintes globales ne sont pas spécifiables. Le solveur CE implémente différentes techniques de propagation (*fail-first* ou *round-robin* couplée avec HC ou BC). L'exploration est réalisée par BP (les algorithmes exacts ne sont pas accessibles car le code source n'est pas disponible). Une optimisation par algorithme n'est pas possible, cependant une procédure de résolution par optimisation similaire à celle présentée Alg. 2 peut être implémentée à la main.

Très peu de solveurs de contraintes semblent résoudre des CSP mixtes nativement (sans nécessiter d'extension à l'aide de plug-in par exemple) hormis CE et DEPS Studio (sous réserve qu'il y en ait d'autres en cours de développement). DEPS Studio ne sera pas étudié dans ce chapitre car le formalisme associé DEPS est un formalisme étendu. DEPS / DEPS Studio feront l'objet du prochain chapitre. Des tentatives de couplage entre solveurs discrets et continus ont été réalisées mais n'ont pas été très concluantes [21]. Le formalisme et solveur CE étaient également liés à l'encadrement de ces travaux de thèse et disponibles. En conclusion, nous avons choisi le formalisme et solveur CE distribué par Dassault Aviation pour évaluer les capacités et limites de la PPC pour la conception de systèmes physiques en Ingénierie des Systèmes (IS) et en Génie Electrique (GE).

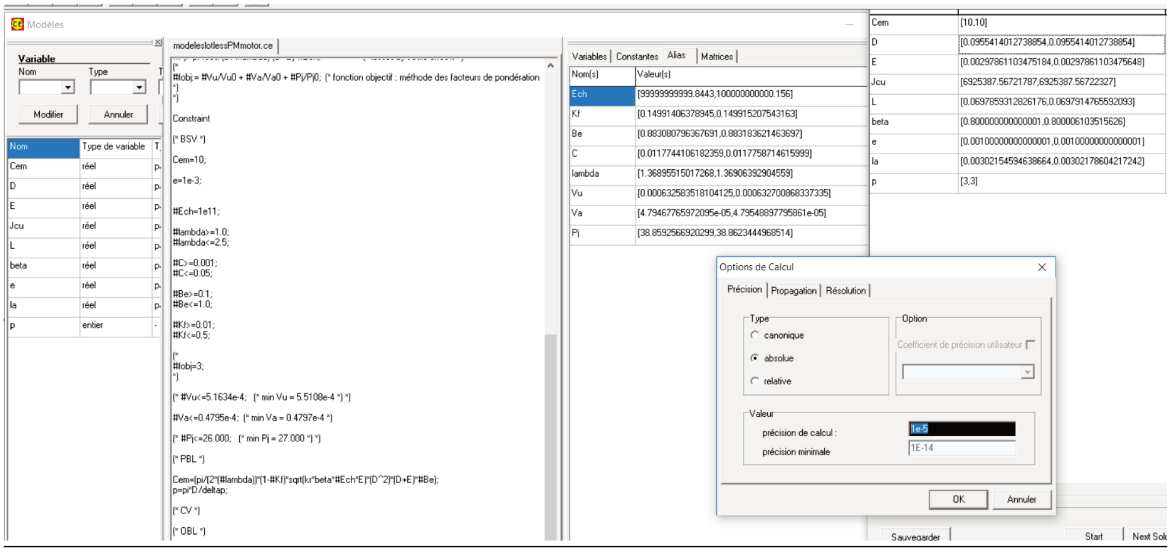
2.7 Exemples de pré-conception en GE à l'aide de CE

Dans cette section, nous allons illustrer la conception de systèmes du GE en utilisant la PPC à travers le formalisme et outil CE. Afin de montrer leur pertinence en conception des systèmes physiques, 3 exemples seront présentés.

Le langage CE étend la définition originelle des CSP via la possibilité de déclarer et définir de manière distincte :

— Les *constantes* : elles peuvent regrouper des constantes universelles, des paramètres constants (ex : caractéristiques de matériaux) et des contraintes technologiques ou de réalisation (ex : un entrefer e à 1 mm). Elles sont déclarées et définies dans la Zone *Constant* ;

Figure 2.4 L'environnement de formalisation et de résolution CE de CSP.



— Les *alias* : il s'agit d'une référence à une expression scalaire ou matricielle comme par exemple l'aire d'un cercle que l'on va nommer A au lieu de réécrire son expression πr^2. Les alias correspondent à des variables déduites. Ils sont déclarés et définis à l'aide du symbole d'affectation := dans la Zone *Alias* (ex : $A := \pi r^2$) ;

— Les *fonctions* : une fonction définit un prototype d'expression paramétré par une liste de domaines. Des fonctions prédéfinies sont disponibles et on peut en déclarer de nouvelles. Un modèle CE doit également comprendre une zone de déclaration de fonctions (même si celle-ci est vide).

Les ddl sont typés lors de leurs déclarations dans la zone *Variable*. Cependant, l'expressivité des types est limitée pour la conception, car seules des quantités de base (ie. réelles ou discrètes) peuvent être renseignées. Ainsi, un modèle CE est une suite de déclarations de constantes, fonctions, ddl, variables déduites et contraintes. L'environnement de formalisation et de résolution CE est présenté Fig. 2.4.

Analyse causale CE propose un outil intégré pour l'analyse causale des modèles construits qui permet d'analyser la causalité entre les ddl et et les variables déduites à partir des contraintes d'égalité et ainsi d'établir un arbre dit de causalité. A partir de cette analyse causale, on peut identifier plus facilement les "faux" ddl qui sont en réalité des variables déduites. Cet outil d'analyse causal peut ainsi aider à la détection de problème sur-contraints ne présentant pas de solutions admissibles, mais également aider à la définition de la taille du problème qui est souvent sur-estimée.

2.7.1 Objectifs de présentation des cas d'étude

Les exemples qui seront étudiés ci-après auront 4 objectifs :

1. illustrer la PPC sur des cas de conception concrets du GE : problèmes mixtes, non-linéaires, voire sur-contraints pour la résolution et l'optimisation (comme dans les exemples benchmark, les problèmes d'optimisation multi-critères sont gérés en les ramenant à des problèmes mono-critères par diverses techniques comme par exemple celle des facteurs de pondération) ;
2. évaluer l'adéquation de formalismes/outils dédiés pour gérer les différents types de problèmes de conception de systèmes complexes physiques, notamment le dimensionnement et la génération d'architectures ;
3. souligner la complémentarité entre formalisation et résolution et notamment l'importance de la première ;
4. souligner les limites de la PPC.

2.7.2 Pré-conception d'un moteur DC Brushless (dimensionnement optimal)

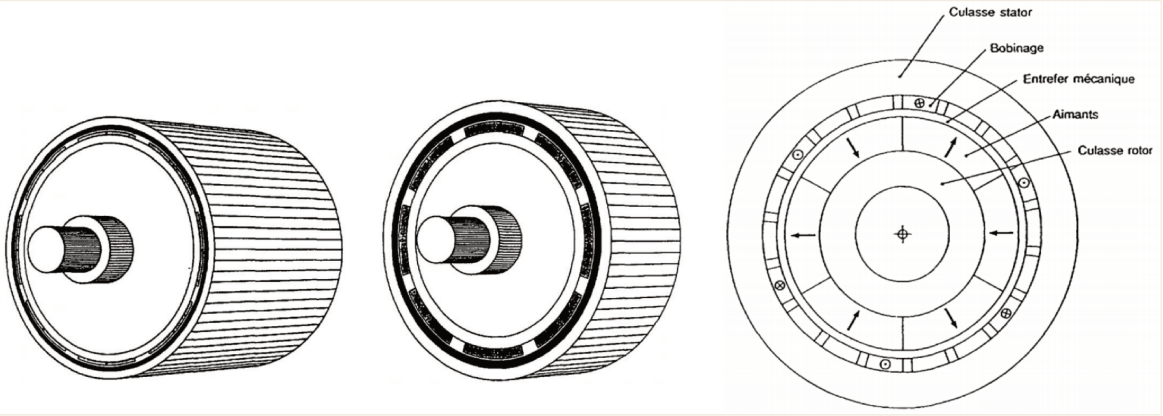
Les objectifs de ce cas d'étude sont de :

— traiter un problème de modèle gros grains algébrique simple pour introduire les problématiques de choix de modélisation ;
— traiter un problème de dimensionnement ;
— traiter un problème mixte ;
— s'intéresser à l'existence de solutions admissibles ;
— réaliser des optimisations mono- et tri-critères ;

Ce premier exemple introduit dans le benchmark [150, 24] traite de la pré-conception d'une machine DC brushless sans encoches à aimants permanents (AP) (cf. Fig. 2.5). Le benchmark traite 4 problèmes de pré-dimensionnement optimal : la minimisation du volume utile V_u, celle du volume des aimants V_a, celle des pertes joules P_j ainsi que leur minimisation simultanée.

Les modèles gros grains sont tirés du benchmark [150, 24]. Le modèle gros grains est non-linéaire et est reporté en Annexe A. Tel que présenté, il est composé de 17 Variables

Figure 2.5 Le moteur DC brushless à AP [150].



dont 11 de conception et de 8 contraintes dont 6 d'égalités [150]. Les ddl sont : le couple électromagnétique C_{em}, l'induction à vide dans l'entrefer Be, l'épaisseur des bobinages statoriques E, le nombre de paires de pôles p, la longueur de la machine L, l'épaisseur d'un aimant rotorique l_a, l'entrefer e, le coefficient d'arc polaire β, le coefficient de fuites interpolaires K_f, le diamètre d'alésage D, l'épaisseur de la culasse statorique C. Dans ce problème mixte, la seule variable discrète est le nombre de paires de pôles p (entier).

Les exigences sont spécifiées ainsi : pour chaque problème, on cherchera à minimiser le critère considéré, sous les conditions suivantes :

- — la machine doit être capable de fournir un couple égal à 10 $N \cdot m$ (cf. Equ. A.16) ;
- — le pas polaire est fixé à 0.100 m (cf. Equ. A.15) ;
- — l'échauffement approximatif de la machine est imposé à $10^{11} A^2/m^3$ (cf. Equ. A.14) ;
- — l'induction maximale dans les tôles ferromagnétiques ne doit pas dépasser 1.5 T (cf. Equ. A.13) ;
- — la magnétisation d'un aimant P est fixée à 0.90 T (cf. Equ. A.12) ;
- — le coefficient de remplissage des bobinages est imposé à 70% (cf. Equ. A.11) ;
- — le coefficient de fuites interpolaires ne doit pas dépasser 0.5 ni être inférieur à 0.01 (cf. Equ. A.7, A.8, A.9, A.10).

Extension du benchmark : problématique d'existence de solutions admissibles

Avant de considérer une problématique de conception optimale, il peut être intéressant de se demander si le problème admet des solutions. Ainsi, nous proposons dans un premier

temps de poser un cinquième problème, celui de l'existence de solutions admissibles, que nous nommerons ci-après problème de solvabilité.

Construction des exigences pour le problème de solvabilité En général, on cherchera à obtenir un couple minimum à fournir à une charge donné. On peut donc relaxer la contrainte de conception $C_{em} = 10N \cdot m$ (cf. Equ. A.16) en une contrainte d'inégalité, telle que $C_{em} \geq 10N \cdot m$ (cf. Equ. A.17). Aucun critère de minimimisation n'est considéré. Cependant, trois contraintes d'intégration seront ajoutées afin de limiter les valeurs de V_u, V_a et de P_j, resp. à 20 m^3, 20 m^3 et 150 W car en pratique, on souhaitera les restreindre (cf. Equ. A.21, Equ. A.22, Equ. A.23). Ces valeurs ont été choisies pour ne pas trop restreindre les domaines de définition spécifiés pour ces variables.

La formulation sous forme de CSP est donnée ci-après pour chacun des cinq problèmes : `P1sol` pour le problème de solvabilité et `P1optVu`, `P1optVa`, `P1optPj`, `P1optTri` resp. pour les problèmes de minimisation mono-critères de V_u, V_a, P_j et celui tri-critères. Afin d'alléger les notations et de ne pas surcharger le manuscrit, les contraintes seront référencées par leurs indexes (ex : A.1). Egalement, les ensembles des contraintes des quatre problèmes d'optimisation seront définis par rapport à l'ensemble `C1` du problème de dimensionnement. La notation $C = (C1 - (A.17)) \cup \{Equ.\ A.16, A.22\}$ sera ainsi équivalente à : "l'ensemble `C` est constitué de l'ensemble `C1`, duquel on aura retiré la contrainte A.17 et ajouté les contraintes A.16 et A.22." Comme il n'y a aucune notion de variables d'entrée/sortie avec les CSP, l'ensemble `X1` de `P1sol` est constitué de 25 ddl x_i.

$$\texttt{P1sol : CSP} = \begin{cases} X1 = \{\mathrm{C}_{em}, B_e, E, p, L, l_a, e, \beta, V_u, V_a, P_j, K_f, D, C, k_f, P, \\ B_{iron}, E_{ch}, \Delta p, V_{amax}, V_{umax}, P_{jmax}, K_{fmin}, K_{fmax}\}, \\ \\ D1 = \{\ [10..10], [0.1,1.0], [0.001,0.05], [|1..10|], [0,0.5], \\ [0.003,0.05], [0.1e\text{-}4,5e\text{-}4], [0.8,1.0], [0,30], [0,50], \\ [0,150], [0,+\infty[, [0,+\infty[, [0,+\infty[, [0,+\infty[, [0,+\infty[, [0,+\infty[, \\ [0,+\infty[, [0,+\infty[, [0,+\infty[, [0,+\infty[, [0,+\infty[, [0.01], [0.5]\}, \\ \\ C1 = \{Equ.\ A.1, A.2, A.6, A.3, A.4, A.7, A.8, A.11, A.12, A.13, \\ A.14, A.15, A.18, A.19, A.20, A.17\} \end{cases}$$

Figure 2.6 Modèle du problème de solvabilité P1sol (CE).

```
Constant
P=0.9 ;
rocu=0.018e-6 ;
pi=3.14159265358979323846 ;
kr=0.7 ;
Ech=10e11 ;
deltap=0.100 ;
Biron=1.50 ;
Vumax=20 ;
Vamax=20 ;
Pjmax=150 ;
Function
log10(_x):=ln(_x)/ln(10);
Variable
Cem:real ;
E:real=[0.001,0.05] ;
e:real=[0.1e-4,5e-4] ;
p:int=[1,10] ;
la:real=[0.003,0.05] ;
beta:real=[0.8,1.0] ;
L:real=[0,0.5] ;
Be:real=[0.1,1.0];
Vu:real=[0,30] ;
Va:real=[0,50] ;
Pj:real=[0,150] ;
Alias
#D:=(p*deltap)/pi ;
#Kf:=1.5*p*beta*(e+E)/#D ;
#C:=(pi*beta*#Be*#D)/(4*p*Biron) ;
Constraint
Cem=(pi*(1-#Kf)*sqrt(kr*beta*Ech*E)*(#D^2)*(#D+E)*Be)/(2*#D/L) ;
Cem>=10 ;
Be=(2*la*P)/(#D*log((#D+2*E)/(#D-2*(la+e)))) ;
Vu=pi*L*(#D+E-e-la)*(2*#C+E+e+la) ;
Va=pi*beta*la*L*(#D-2*e-la) ;
Pj=pi*rocu*L*(D+E)*Ech ;
#Kf<=0.5 ;
#Kf>=0.01 ;
Vu<=Vumax ;
Va<=Vamax ;
Pj<=Pjmax ;
```

$$\texttt{P1optVu : CSP =} \begin{cases} X = X1 \cup \{ \mathrm{J}_{cu} \} \\ D = D1 \cup \{ \text{[1e5,1e7], [0,+}\infty\text{[} \} \\ C = (C1 - (A.17)) \cup \{ Equ.\ A.16, A.22, A.5 \} \end{cases}$$

$$\texttt{P1optVa : CSP =} \begin{cases} X = X1 \cup \{ \mathrm{J}_{cu} \} \\ D = D1 \cup \{ \text{[1e5,1e7], [0,+}\infty\text{[} \} \\ C = (C1 - (A.17)) \cup \{ Equ.\ A.16, A.21, A.5 \} \end{cases}$$

$$\texttt{P1optPj : CSP =} \begin{cases} X = X1 \cup \{ \mathrm{J}_{cu} \} \\ D = D1 \cup \{ \text{[1e5,1e7], [0,+}\infty\text{[} \} \\ C = (C1 - (A.17)) \cup \{ Equ.\ A.16, A.23, A.5 \} \end{cases}$$

$$\texttt{P1optTri : CSP =} \begin{cases} X = X1 \cup \{ \mathrm{J}_{cu}, f_c \} \\ D = D1 \cup \{ \text{[1e5,1e7], [0,+}\infty\text{[} \} \\ C = (C1 - (A.17)) \cup \{ Equ.\ A.16, A.24, A.5 \} \end{cases}$$

Formalisation de P1sol avec CE

La Fig. 2.6 présente une formalisation du CSP `P1sol` avec CE. Ce modèle de synthèse a été suffisamment commenté pour qu'il soit compréhensible même par une personne ne connaissant pas le formalisme CE. On notera que le modèle contient essentiellement une ré-écriture de l'ensemble des contraintes du CSP et de ddl initiaux en alias.

Choix des ddl (P1sol) Dans ce modèle, le choix a été fait dans un premier temps de conserver le même nombre de ddl que dans le benchmark, soit 11. Les choix de ddl sont étudiés sous trois angles : la sémantique, la formalisation et la résolution.

Pour la sémantique Nous considérons les volumes et pertes joules comme ddl à la place de D, K_f et C. En effet, on peut considérer que les paramètres les plus importants pour une machine électrique sont avant tout les paramètres magnétiques. On cherche en général à déterminer les différentes inductions dans le circuit magnétique ainsi que le couple électromagnétique. De même, les caractéristiques géométriques telles que p ou L sont à prendre en compte en priorité. Les pertes joules sont aussi des composantes critiques pour les machines électriques car la température des bobinages, dont elles dépendent, est fortement contrainte.

Enfin, on peut également vouloir ne s'intéresser qu'à l'encombrement du moteur plutôt qu'à ses dimensions. En conséquence, le modèle gros grains a dû être manuellement inversé pour une relation, afin d'exprimer l'alias D en fonction de la géométrie des aimants, tel que : $D := (p * \Delta p)/\pi$. Si cela a été très simple et rapide, cela peut rapidement devenir fastidieux et coûteux si le modèle gros grains du problème comporte beaucoup de relations, qui plus est non-linéaires. C'est pourquoi avoir une bonne estimation de la taille du problème dès le départ est importante. Les exigences, à l'exception de celle portant sur K_f ont été formalisées comme un ensemble de constantes. En effet, pour chacune de ces variables, il faut considérer les aspects suivants.

Pour la formalisation : Tout d'abord, il faut se demander si ces ddl initiaux représentent de vrais ddl ou simplement des variables intermédiaires (qui peuvent être définies comme des alias). Autrement dit, quelle est leur sémantique dans le problème considéré. Si on considère que le problème est résolvable par 11 ddl, il faudra alors en substituer certaines déjà déclarées et définir les contraintes de conception associées. Ensuite, il faut se demander si les exigences représentent des contraintes de fonctionnement communes aux problèmes de conception des machines ou non. Si oui, on peut les envisager comme des variables pseudo-constantes, ie.

constantes pour un type de problème de conception donné. Enfin, on peut s'interroger sur l'impact de tels choix sur le coût de la modélisation. Si on peut les définir simplement comme des constantes sans que la lisibilité du modèle ne soit trop impacté, alors c'est le choix le plus simple et le moins coûteux.

Pour la résolution : La déclaration et définition d'alias améliore la compréhension des modèles construits pour les concepteurs. Cependant, on peut s'interroger sur leurs gestions par le solveur. En effet, elles peuvent considérablement raccourcir ou rallonger les temps de calculs selon leur "origine". Si un ddl est changé en alias, la résolution est plus rapide (sous réserve que le problème puisse encore être résolu). En revanche, si une contrainte est changée en alias, la résolution serait plus longue (la contrainte n'est plus utilisée pour la propagation). On peut donc voir que ce problème simple du point de vue du modèle mathématique, nécessite tout de même une réflexion au regard de la formalisation. Il est en effet important de concevoir des modèles de synthèse aussi bien performants au niveau de la résolution, que lisibles pour pouvoir être repris ou retravailler dans d'autres projets.

Résolution de P1sol avec CE

Pour la résolution de `P1sol`, les deux heuristiques d'ordonnancement de variables (*fail-first*, *round-robin*) ainsi que les deux techniques de propagation (Hull-consistance (HC), Box-consistance (BC)) disponibles ont été appliquées. Il y a donc 4 combinaisons qui ont été testées, avec une précision globale ϵ de 10^{-5}. Les différentes résolutions ont permis de confirmer la solvabilité du CSP. Ce problème mal-posé comprend plus d'un million de solutions admissibles. Une solution admissible est reportée Tableau 2.1. On peut voir que pour ce problème de solvabilité toutes les contraintes ont été respectées.

La BC testée avec la *fail-first* et la *round-robin* était moins performante en termes de temps de calcul que la HC. En effet, la HC associée à chacune de ces heuristiques permettait d'obtenir une première solution admissible en moins de 2 et 5 secondes respectivement et en une vingtaine de secondes pour la BC. Même s'il est difficile d'évaluer les performances des techniques sans d'autres résultats (obtenus avec d'autres techniques), on peut s'imaginer que ces temps sont satisfaisants. L'heuristique *fail-first* était dans les deux cas la plus adaptée. En observant les domaines des ddl, le plus petit est celui de p avec 10 valeurs (au regard de la précision de découpe ϵ). C'est donc le ddl le plus susceptible de causer un échec et a été instancié en premier avec la *fail-first*. Avec la *round-robin*, les ddl sont affectés à tour de rôle en commençant par C_{em}. Comme son domaine est plus grand, on peut envisager que

Table 2.1 Une solution admissible au problème de solvabilité du moteur DC brushless.

ddl	Domaines	Dimensionnement
C_{em}	$[10,+\infty]$	10.17
B_e	[0.1,1.0]	0.72
E	[0.001,0.05]	0.00507
p	[|1,10|]	6
L	[0,0.5]	0.0105
l_a	[0.003,0.05]	0.003
e	[0.1e-4,5e-4]	4e-5
β	[0.8,1.0]	0.8
V_u	[0,30]	9.98e-5
V_a	[0,50]	6.92e-6
P_j	[0,150]	57.84
Variables déduites		
K_f	$[-\infty,+\infty]$	0.396
D	$[-\infty,+\infty]$	0.191
C	$[-\infty,+\infty]$	0.0960

l'exploration soit plus longue si elle est menée dans une branche plus longue et dont la valeur de la feuille est inconsistante.

Formalisation de P1optVu avec CE

Choix des ddl (P1optVu) Pour les problèmes de conception optimale, on définira généralement les critères V_u, V_a et P_j comme des variables déduites. Ils sont donc redéclarés et définis comme alias. Le rapport D/L qui est utilisé pour les définir, correspond au facteur de forme de la machine, et est généralement remplacé par la variable alias λ. D et L sont alors déclarées comme ddl.

De plus, le choix a été fait de considérer E_{ch} comme une variable déduite plutôt que comme une constante. En effet, en pratique, l'échauffement de la machine est soumis à une contrainte qui variera d'une application à un autre (en fonction des conditions de pression et de température prévues pour son fonctionnement). De plus, le modèle analytique ne donne qu'une approximation très grossière de cette inconnue.

En général, on cherchera plutôt à connaître la densité de courant linéique dans les bobinages en cuivre J_{co} lors de la conception de la machine. Nous disposons alors de seulement 10 ddl. Si on se reporte au problème de dimensionnement `P1sol`, il est nécessaire d'en déclarer une de plus. Le choix s'est porté sur l'épaisseur de la culasse statorique C car il s'agit notamment d'une variable structurante. Nous obtenons ainsi le problème de minimisation

de V_u `P1optVu` à 11 ddl, qui sont : $Cem, E, e, p, la, beta, L, J_{co}, D, C$ et B_e. Pour finir, les constantes V_{umax}, V_{amax} et P_{jmax} correspondant aux valeurs limites de V_u, V_a et P_j sont directement substituées par leurs valeurs, afin d'obtenir un modèle toujours lisible mais plus compact.

Implémentation de la boucle d'optimisation La minimisation de V_u consistera à résoudre une suite de problèmes où sa valeur sera décrémentée manuellement, en fonction de sa valeur obtenue précédemment (l'automatisation de cette boucle d'optimisation n'est actuellement pas disponible dans CE). Dans un premier temps, il faut donc s'intéresser à la solvabilité de ce CSP, comme pour le problème précédent `P1sol`.

Analyse causale (P1optVu) Dans le problème précédent, nous nous intéressions simplement à la solvabilité du CSP `P1sol`. On peut s'interesser à présent à la qualité de la formalisation du problème, à savoir si sa taille a été bien estimée ou sur-estimée comme cela est souvent le cas. On s'aidera pour cela de l'outil d'analyse causale de CE. Les relations causales entre les variables sont analysées après filtrage des domaines, à partir de toutes les contraintes d'égalités. A partir de celles-ci, un graphe de causalité est construit.

Ce graphe établit ainsi les liens de dépendances (s'il y en a) entre les ddl déclarés dans le modèle de synthèse. On peut ainsi voir qu'en réalité seulement 9 ddl définissent ce problème et que deux sont en réalité des variables déduites. (qui peuvent être ré-écrites comme des alias). Du point de vue du coût de la formalisation, le moins coûteux consisterait à déclarer et définir C, C_{em} ou B_e comme alias, au regard du modèle gros grains disponible. Comme expliqué précedemment, il est primordial de conserver C_{em} comme ddl pour conserver la sémantique du problème. Suite à ces considérations, on obtient finalement le modèle présenté Fig. 2.7. C'est dans cette même démarche de réflexion qu'ont été construits les modèles de synthèse des problèmes de minimisation `P1optVu`, `P1optVa`, `P1optPj` et `P1optTri`.

Résolution de P1optVu, P1optVa, P1optPj et P1optTri Les Tab. 2.2 et 2.3 regroupe les solutions optimales obtenues pour chacun des quatre problèmes de minimisation. Les solutions ont été obtenues par *fail-first* et HC pour les mêmes raisons que lors de la résolution du problème `P1sol`. Les temps de résolution des problèmes mono-critères doivent cependant tenir compte du processus manuel d'optimisation et pour lesquels les valeurs optimales ont été obtenues au bout de 2 itérations chacune en moins de 3 secondes.

Figure 2.7 Formalisation de P1optVu de la machine sans encoches à AP (CE).

```
Constant
P=0.9;
rocu=0.018e-6;
pi=3.14159265358979323846;
kr=0.7;
deltap=0.100;
Biron=1.50;
Function
log10(_x):=ln(_x)/ln(10);
Variable
Cem:real;
E:real=[0.001,0.05];
e:real=[0.1e-4,5e-4];
p:int=[1,10];
la:real=[0.003,0.05];
beta:real=[0.8,1.0];
L:real=[0,0.5];
Jco:real=[1e5,100e5];
D:real=[0.01,0.5];
Alias
#Ech:=kr*E*(Jco^2);
#Kf:=1.5*p*beta*(e+E)/D;
#Be:=(2*la*P)/(D*log10((D
    +2*E)/(D-2*(la+e))));
#C:=(pi*beta*#Be*D)/(4*p*
    Biron);
#lambda:=D/L;
#Vu:=pi*(D/#lambda)*(D+E-e
    -la)*(2*#C+E+e+la);
#Va:=pi*beta*la*(D/#lambda
    )*(D-2*e-la);
#Pj:=pi*rocu*(D/#lambda)*(
    D+E)*#Ech;
Constraint
Cem=10;
#Ech=1e11;
#lambda>=1.0;
#lambda<=2.5;
#C>=0.001;
#C<=0.05;
#Be>=0.1;
#Be<=1.0;
#Kf>=0.01;
#Kf<=0.5;
#Vu<=5.5e-4;
#Va<=2e-4;
#Pj<=60;
 (*PBL *)
Cem=(pi/(2*(#lambda))*(1-#
    Kf)*sqrt(kr*beta*#Ech*
    E)*(D^2)*(D+E)*#Be);
p=pi*D/deltap;
```

Comparaisons entre résultats obtenus dans ces travaux de thèse et ceux des autres équipes (benchmark)

Les résultats de ce benchmark obtenus par d'autres équipes de recherche sont également présentés (qui utilisent la technique de Method of Multipliers (MOM) [151, 152] pour [150] et de Branch and Bound (BB) pour [24]). L'optimisation simultanée des trois critères V_u, V_a et P_j est obtenue via l'Equ. A.24. Les résultats que nous avons obtenus sont quasi-systématiquement plus optimaux que ceux trouvés dans [150] et [24]. Ils sont meilleurs pour les minimisations de V_u (5.501e-4 m^3) et V_a (4.79e-5 m^3) et celle tri-critères. Les résultats obtenus par BB sont légèrement meilleurs pour la minimisation de P_j : 37.695 W vs. 37.581 W (ils sont globalement assez proches de ceux que nous avons obtenus). Ces premiers résultats encourageants pour le dimensionnement de ce système nous a mené à envisager d'autres problèmes plus difficiles du point de vue de la résolution, notamment des problèmes de génération d'architectures et de contraintes catalogue comme nous le verrons dans le prochain exemple.

2.7.3 Génération d'architectures optimales d'un moteur brushless

Ce cas d'étude de conception permettra de :

- traiter un problème de modèle gros grains algébrique simple pour introduire les enjeux de modélisation des problèmes de génération d'architectures ;
- traiter un problème mixte avec plus d'une variable discrète ;
- réaliser des optimisations mono- et bi-critères ;

Table 2.2 Solutions optimales aux problèmes de minimisation de V_u, V_a.

ddl	Optimisation					
	V_u			V_a		
	BP/**CE**	MOM [150]	BB [24]	BP/**CE**	MOM [150]	BB [24]
C_{em}	10	10	10	10	10	10
E	0.005436	0.0038	0.0055	0.002978	0.0021	0.0054
p	5	4	5	3	4	5
L	0.0637	0.0692	/	0.6978	0.1273	/
l_a	0.0030	0.0053	0.0030	0.0030	0.0017	0.0030
e	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010
β	0.80	0.8	0.8	0.8	0.8	0.8
J_{co}	5.126e6	6.148e6	5.096e6	6.925e6	8.179e6	5.143e6
D	0.1592	0.1273	0.1592	0.0955	0.1273	0.1592
Variables déduites						
B_e	0.2882	0.462	0.287	0.883	0.315	0.289
K_f	0.243	0.180	0.245	0.150	0.118	0.241
C	0.0038	0.0062	0.0038	0.0117	0.0042	0.0039
E_{ch}	1e11	1e11	1e11	1e11	1e11	1e11
λ	2.4982	/	2.4935	1.3690	/	2.4996
Critères						
V_u	5.501e-4	6.073e-4	5.511e-4	6.326e-4	6.704e-4	5.527e-4
V_a	7.40e-5	11.02e-5	7.42e-5	4.79e-5	6.72e-5	7.40e-5
P_j	59.280	51.317	59.463	38.86	93.211	59.282

— prendre en compte des contraintes catalogue.

Pour ce deuxième exemple, nous avons choisi de traiter le problème de pré-conception optimale décrit dans le benchmark [25] qui fait suite aux travaux présentés dans [150, 24]. Un cas original de conception de moteur brushless à aimants permanents (AP) à flux radial y est présenté (cf. Fig. 2.8). C'est en effet un des rares cas de la littérature à traiter de problème de génération d'architectures de machines électriques (qui est notamment mixte). Cet exemple est également intéressant du point de vue de la formalisation puisqu'il intègre une contrainte catalogue. C'est en particulier sur ce dernier aspect que sera étudié ce cas d'étude. Les travaux présentés dans cette section ont fait l'objet d'une communication dans le congrès SGE 2018 [153].

Trois problèmes de pré-conception optimales ont été étudiés : deux mono- et un bi-critères : la minimisation de la masse des parties actives M_a, celle du volume global V_g ainsi que de la prise en compte simultanée de ces deux critères. Le modèle gros grains proposé dans [25]

Table 2.3 Solutions optimales aux problèmes de minimisation de P_j et tri-crit.

ddl	Optimisation					
	P_j			Tri-critère		
	BP/**CE**	MOM [150]	BB [24]	BP/**CE**	MOM [150]	BB [24]
T_{em}	10	10	10	10	10	10
E	0.0035	0.0054	0.0031	0.0046	0.0044	0.0040
p	4	4	4	3	4	4
L	0.0509	0.0510	/	0.0590	0.0604	/
l_a	0.0084	0.0117	0.0180	0.0063	0.0074	0.0076
e	0.0004	0.0010	0.0010	0.0010	0.0010	0.0010
β	0.8	1.0	1.0	0.8	0.8	0.8
J_{co}	6.394e6	5.160e6	6.788e6	5.931e6	5.687e6	5.976e6
D	0.1274	0.1273	0.1273	0.1273	0.1273	0.1273
Variables déduites						
B_e	0.577	0.545	0.633	0.519	0.502	0.521
K_f	0.132	0.300	0.193	0.135	0.204	0.188
C	0.0077	0.0091	0.0105	0.0071	0.0067	0.0069
E_{ch}	1e11	1e11	1e11	1e11	1e11	1e11
λ	2.471	/	2.497	1.921	/	2.116
Critères à optimiser						
V_u	5.333e-4	6.961e-4	7.600e-4	5.649e-4	6.137e-4	6.121e-4
V_a	1.281e-4	2.131e-4	3.000e-4	1.285e-5	1.328e-4	1.352e-4
P_j	37.695	38.215	37.581	44.003	45.000	44.664

Figure 2.8 Problème de génération d'architectures optimal d'un moteur brushless [25].

a été repris dans ces travaux de thèse. Il prend notamment en compte (cf. Table 2.4) :

- les différentes configurations possibles : présence ou non d'encoches, forme d'onde sinusoïdale ou rectangulaire (moteur AC ou DC brushless), rotor interne ou externe ;
- le type de matériaux constitutifs : des aimants NdFeB (aimants à liant synthétique ou moderne) ou encore du circuit magnétique (poudre magnétique douce ou matériaux emboutis.

Table 2.4 Propriétés des matériaux magnétiques.

Matériau	ddl	Type de matériau	J (T)	ρ_{PM} (kg/m^3)	B_M (T)	ρ_{CM} (kg/m^3)
Aimants permanents	σ_m	plastique	0.6	6000	/	/
		terres rares/moderne	0.9	7900	/	/
Conducteur magnétique	σ_{mt}	poudre	/	/	1.2	6000
		emboutie	/	/	1.5	7900

Ce modèle gros grains est non-linéaire et allie des modèles géométrique, électromagnétique et thermique en régime permanent. Il est reporté en Annexe B. Tel que présenté, il est composé de 56 variables dont 22 de conception. Parmis les contraintes, 2 sont d'inégalités [150]. Les ddl présentés sont : le diamètre d'alésage D, l'épaisseur des bobinages statoriques E, la longueur de la machine L, le coefficient de remplissage des bobinages k_r, la densité de courant surfacique j, la longueur d'une encoche a, la longueur d'une dent d, le coefficient d'arc polaire β, le nombre de paires de pôles p, l'entrefer g, l'épaisseur d'un aimant rotorique l_a, le nombre de phases q, le nombre d'encoches par pôles par phases m, l'épaisseur de la culasse statorique C, la température des bobinages en cuivre θ_J, la température dans l'entrefer θ_e, la température de l'air ambiant θ_c, le type d'encoches σ_e, le type de forme d'onde σ_f, le type de configuration rotorique σ_r, le type d'aimants permanents σ_m, le type de conducteur magnétique σ_{mt}.

Les 6 ddl discrets sont p (entier) ainsi que les 5 ddl de choix de structures (booléens). Les 9 variables déduites sont les fonctions d'inclusions, soit : K_f, C_{em}, Be, K_s, k_d, B_c, B_t, R_{int}, R_{ext}. Six constantes sont également définies. Les exigences sont spécifiées ainsi : pour chaque problème, on cherchera à minimiser le critère considéré, sous les conditions suivantes :

- la densité de flux dans les dents B_t (cf. Equ. B.16) ;
- pour une machine à encoches, la densité de flux dans la culasse statorique B_c ne doit pas dépasser l'induction maximale dans les aimants , qui dépend du type de conducteur magnétique (cf. Equ. B.17).

Chaque type de configuration et de matériau induit un comportement différent de la machine. Pour représenter ces différents comportements et structures alternatives, 5 variables booléennes sont introduites : $\sigma_e, \sigma_f, \sigma_r, \sigma_m$ et σ_{mt}. Une des difficultés calculatoires introduite dans ce problème de conception mixte est que certaines variables de comportement

sont directement induites à partir des ddl discrets (σ_r et σ_{mt}). Or ces deux ddl apparaissent au sein d'une contrainte catalogue (cf. Tab. 2.4). Pour tenter de contourner cette difficulté, [25] propose d'introduire directement dans le modèle gros grains des fonctions opérant sur des intervalles pour toutes les fonctions univariantes opérant sur des réels (telles que $S(\sigma_r)$) intervenant dans des contraintes mixtes. Cela dans le but de pouvoir appliquer des techniques de résolution utilisant le calcul par intervalles.

Ce modèle mathématique théorique est pensé pour la résolution avec la configuration du moteur modélisée en amont de la construction des modèles de synthèse. Cependant, celle-ci n'est pas complètement décrite par le modèle gros grains, car il faut en traduire une partie qui n'est formulée que textuellement. C'est le cas pour les contraintes de conception logiques introduites implicitement avec les 5 ddl de choix de configuration, telles que : "si la fonction $S(\sigma_r)$ vaut -1, alors $\sigma_r = 0$, sinon 1 si $\sigma_r = -1$". Ces connaissances implicites ne sont généralement pas comptabilisées dans le nombre d'équations du modèle gros grains. On peut tout de même se demander dans quelle mesure un tel modèle est valide en pratique pour la conception. Egalement, celui-ci peut se trouver alourdi si les possibilités de configurations et surtout de comportements sont nombreuses.

Pour chacun des quatre problèmes de minimisation, une formulation CSP est donnée de manière similaire à celles présentées dans le premier exemple. Les CSP `P2optMa`, `P2optVg`, `P2optBi-M` et `P2optBi-P` correspondent respectivement aux problèmes de minimisation mono-critère de M_a, de V_g et celui bi-critères.

Minimisation de M_a

$$\texttt{P2optMa : CSP =} \begin{cases} X2 = \{\ \text{D, E, L, j, a, d, p, la, C, } \sigma_e, \sigma_f, \sigma_r, \sigma_m, \sigma_{mt}, C_{em}, B_e, \\ \text{K}_s, k_d, B_c, \text{B}_t, R_{int}, R_{ext}, M_a, V_g, V_c, V_t, V_{co}, V_a, \rho_{cu}, \\ \text{h}_e, h_a, \tau_m, \rho_{co}, \rho_{Al}, g, \theta_J, \theta_e, \theta_c, B_M, q, m\} \\ \\ D2 = \{[1,300], [4,100], [50,150], [3,6], [4,+\infty], [4,+\infty], [1,16], \\ [4{,}100], [4{,}100], \{0{,}1\}, \{0{,}1\}, \{0{,}1\}, \{0{,}1\}, \{0{,}1\}, \\ [0.4{,}0.9], [0.5{,}100], [0.4{,}0.6], [0.9, B_m(\sigma_{mt})], [0.9, B_m(\sigma_{mt})], [50{,}+\infty], \\ [0{,}80], [0, +\infty[, [0, +\infty[, [0, +\infty[, [0, +\infty[, [0, +\infty[, [0, +\infty[, \\ [1{,}1], [3{,}3], [130{,}130], [55{,}55], [50{,}50], \{1.2{,}1.5\}\} \\ \\ C2 = \{Equ.\ B.1, B.2, B.3, B.4, B.5, B.6, B.7, B.8, B.9, B.10, B.11, \\ B.12, B.13, B.14, B.15, B.16, B.18, B.19, B.20, B.21, B.22, B.23, \\ B.24, B.25, B.26, B.27, B.28, B.29, B.17, B.31, B.32\} \end{cases}$$

Minimisation de V_g

$$\texttt{P2optVg : CSP =} \begin{cases} X = X2 \\ D = D2 \\ C = (C2 - (B.29)) \cup \{Equ.\ B.30\} \end{cases}$$

Pour trouver la solution pareto-optimale au problème bi-critères, deux techniques ont été utilisées : de Marglin [154, 25] et des facteurs de pondération [155]. Toutes deux ramènenent en réalité un problème d'optimisation multi-critères à un problème mono-critère. Pour la technique de Marglin, M_a a été choisi comme critère d'optimisation et V_g transformé en contrainte d'inégalité.

$$\texttt{P2optBi-P : CSP =} \begin{cases} X = X2 \\ D = D2 \\ C = (C2 - (B.29)) \cup \{Equ.\ B.34\} \end{cases}$$

Figure 2.9 Formalisation du problème (P2optMa) (CE).

```
Constant
pi=3.14159265359;
roCo=8900;
roAl=2700;
roCu=0.018e-6;
he=12;
ha=4;
tauM=10^7;
kr=0.7;
g=1e-3;
beta=0.85;
q=3;
m=1;
CRint=50e-3;
CRext=80e-3;
thetaC=323.15;
thetaJ=403.15;
thetaE=328.15;
Function
barre(_x):=1-_x;
Variable
Cem:real;
E:real=[4e-3,100e-3];
p:int=[1,16];
la:real=[4e-3,100e-3];
a:real=[4e-3,inf];
d:real=[4e-3,inf];
L:real=[50e-3,150e-3];
D:real=[1e-3,300e-3];
j:real=[3e6,6e6];
C:real=[4e-3,100e-3];
Be:real=[0.4,0.9];
Rext:real=[50e-3,80e-3];
sigmaM:enumint={0,1};
J:enumreal={0.6,0.9};
roPM:enumreal={6000,7900};

sigmaMt:enumint={0,1};
Bm:enumreal={1.2,1.5};
roCM:enumreal={6000,7900};

sigmaE:enumint={0,1};
sigmaF:enumint={0,1};
sigmaR:enumint={0,1};
Alias
#Ne:=2*p*q*m;
#Kf:=1.5*p*beta*((E+g)/D)*
    sigmaF*(barre (sigmaE)
    );
#kT:=(pi/2)*(sigmaF*(1-#Kf
    )*sqrt(beta)+(barre (
    sigmaF))*(sqrt(2)/2)*
    sin(beta*pi/2));
#kd:=sigmaE*(d/(d+a));
#kc:=1/(1-sigmaE*(#Ne*(a
    ^2)/(5*pi*D*g+pi*D*a))
    );
#Bt:=Be*sigmaE*(a+d)/a;
#Bc:=Be*D*(beta*(pi/2)*(
    barre(sigmaF))+sigmaF)
    /(2*p*C);
#Ks:=kr*E*j*((a/(a+d))*
    sigmaE+(barre(sigmaE))
    );
#Re:=1/(pi*D*L*he);
#Rc:=1/(pi*(D+2*SsigmaR*(E
    +C))*L*ha);
#phiTh:=(thetaJ-thetaE)/#
    Re+(thetaJ-thetaC)/#Rc
    ;
#Vm:=beta*pi*L*la*(D-
    SsigmaR*(2*g+la));
#Vd:=pi*L*E*(D+SsigmaR*E)
    *((1-beta)*sigmaE+(d/(
    d+a))*barre(sigmaE));
#Vc:=2*pi*L*C*(D+SsigmaR*(
    E-g-la));
#Vco:=kr*pi*L*E*(D+SsigmaR
    *E)*(beta*barre(sigmaE
    )+(a/(a+d))*sigmaE);
#Vg:=((pi*L)/4)*(sigmaR*((
    D+2*(E+C))^2)+barre(
    sigmaR)*((D+2*(g+la+C)
    )^2));
#Ma:=#Vm*roPM+#Vc*roCM+#Vd
    *(roCM*sigmaE+roAl*
    barre(sigmaE))+#Vco*
    roCo;
#Rint:=(D/2)-C-barre(
    sigmaR)*E-sigmaR*(g+la
    );
Constraint
Cem=10;
sigmaR=0 ~>{SsigmaR=-1;};
sigmaR=1 ~>{SsigmaR=1;};
sigmaE*#Bt<=Bm;
#Bc<=Bm;
#Rint>=CRint;
Rext=(D/2)+C+sigmaR*E+(
    barre(sigmaR))*(g+la);
C=(pi*beta*Be*D)/(4*p*Bm);
#Ks>=500;
#Ks<=1e5;
#Kf>=0.1;
#Kf<=0.3;
#kd>=0.4;
#kd<=0.6;
#Ma<=5;
Cem=#kT*D*(D+barre(sigmaE)
    *SsigmaR*E)*L*Be*#Ks;
Be=(2*J*la/(SsigmaR*D*ln((
    D+2*E*SsigmaR*(barre(
    sigmaE)))/(D-2*SsigmaR
    *(la+g)))))*(1/#kc);
table(3,sigmaM,J,roPM,
    tablePM) ;
table(3,sigmaMt,Bm,roCM,
    tableMC) ;
```

$$\texttt{P2optBi-M : CSP} = \begin{cases} X = X2 \\ D = D2 \\ C = (C2 - (B.29)) \cup \{Equ.\ B.33\} \end{cases}$$

Pour ce cas d'étude, nous nous interesserons en particulier à la formalisation des contraintes catalogues avec `P2optMa`. La démarche est similaire pour les trois autres CSP (`P2optVg`, `P2optBi-M` et `P2optBi-P`).

Choix des ddl Les choix ont été fait de considérer les ddl q et m comme des pseudo-constantes. En effet, les contraintes unaires portant sur ces Variables sont communes à la conception des machines (généralement triphasées). Pour les mêmes raisons que dans le pre-

mier cas d'étude, Cem, B_e et R_{ext} ont été choisies comme ddl. Les différentes Variables liées aux températures ont été déclarées et définies comme des alias, mais la densité de courant surfacique a été conservée comme ddl.

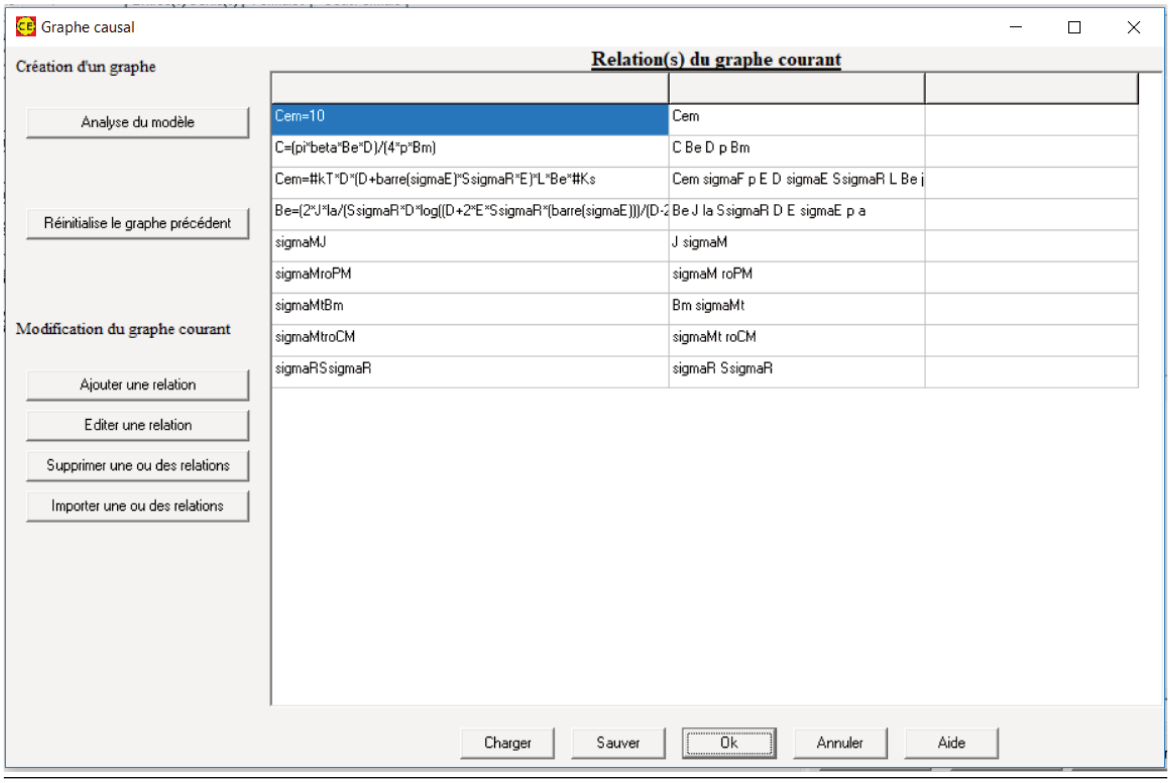
Formalisation de contraintes catalogue La contrainte catalogue induite par la Table 2.4 impose les valeurs de σ_m pour le choix des aimants permanents et de σ_{mt} pour celui du conducteur magnétique. Cette formalisation est bien évidemment propre à CE mais elle permet d'illustrer l'implémentation d'une telle contrainte. Elle est modélisée en quatre étapes. Tout d'abord, le catalogue de composants est modélisée dans un fichier séparé au modèle CE. Ensuite, les ddl de la table doivent être déclarées comme le reste de ddl. Troisièmement, il faut établir les relations causales liant les différentes variables à la table et qui seront ajoutées à l'arbre de causalité. Finalement, la contrainte table est définie par une fonction table dont les entrées correspondent au nombre de ses ddl, leurs identificateurs dans le modèle ainsi qu'à celui de la table, comme le montre le modèle Fig. 2.9 - Ligne 166 pour le choix des aimants permanents. On peut ainsi voir que l'application en pratique de la formalisation d'une contrainte catalogue est intuitive, bien que la troisième étape peut être très longue s'il y a de nombreuses variables intervenant dans la table.

Formalisation de la connaissance implicite conditionnelle Pour formaliser une connaissance implicite conditionnelle, le formalisme et le solveur associé doivent pouvoir gérer les déclarations conditionnelles. Si la condition ne porte que sur une variable, et qu'il y a peu d'alternatives, on peut donc la formaliser soit comme une table, soit comme une déclaration conditionnelle classique (cf. lignes 150-151).

Analyse causale (P2optMa)

Les contraintes ont d'abord été propagées par HC avec l'heuristique *fail-first*. L'analyse causale de ce modèle de synthèse nous montre que celui-ci est constitué de 9 relations d'égalités indépendantes (cf. Fig. 2.10). Egalement, tous les ddl (20 au final) sont bien contraintes (cf. Fig. 2.11 et la première ligne de l'arbre Fig 2.12). On peut ainsi voir que la machine électrique n'est en réalité définie que par 12 ddl qui correspondent ici à : C_{em}, C, D, p, σ_f, E, σ_e, L, l_a, σ_m, σ_{mt}, σ_r. On peut remarquer que la première ddl à instancier est C_{em}, dont la contrainte de fonctionnement associée impose un couple de 10 $N \cdot m$. En revanche, on pourrait s'attendre à ce que les ddl discrets soient ensuite sélectionnés, ce qui n'est pas le cas. Les 12 ddl peuvent être réorganisées de manière à redefinir un ordonnancement pour ceux-ci avant l'exploration.

Figure 2.10 Indépendance des équations pour P2optMa (CE).



Comparaisons entre résultats obtenus dans ces travaux de thèse et ceux des autres équipes (benchmark)

La solution optimale du CSP `P2OptMa` a été trouvée en moins de 5 secondes. Les solutions optimales de l'ensemble de ces 4 CSP ainsi que celles issues du benchmark [25] sont reportées Tab. 2.5 et 2.6. Une étude détaillée des résultats peut être trouvée dans [153] et est résumée ci-après. Les résultats par BP avec CE sont plus optimaux que ceux obtenus par BB dans [25] pour les critères à minimiser : 2.20 kg pour M_a, 0.80e-3 m^3 pour V_g et pour l'optimisation bi-critères : 2.21 kg pour M_a et 0.88e-3 m^3 pour V_g. On peut voir que les résultats que nous obtenons sont réalistes et cohérents avec ceux du benchmark. Les Variables respectives se trouvent dans les mêmes ordres de grandeurs et ont des valeurs parfois très proches voire identiques.

2.7.4 Pré-dimensionnement optimal d'un transformateur de sécurité

Les objectifs de ce cas d'étude de conception sont de :

— traiter un problème sur-contraint ;

Figure 2.11 Analyse causale permettant d'obtenir la vraie taille du problème P2optMa (CE).

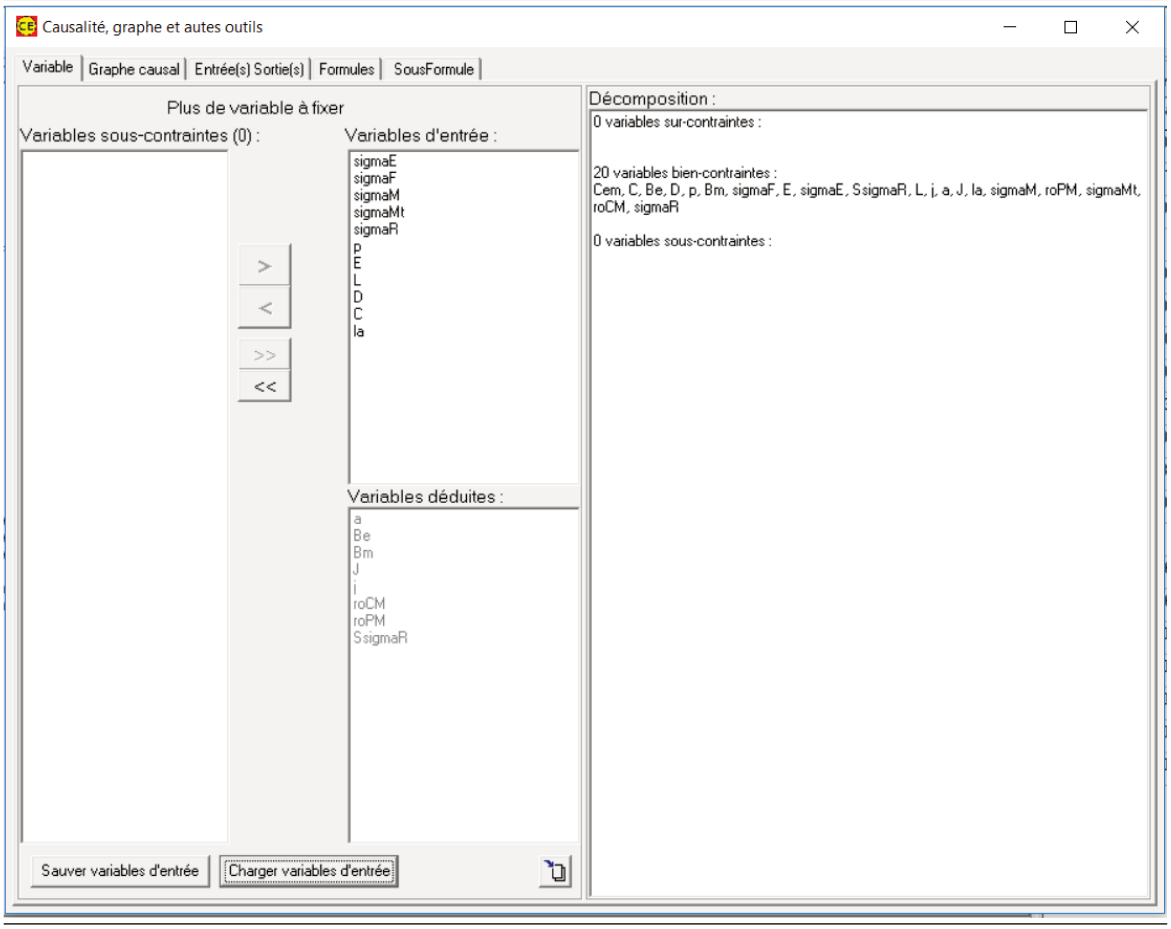
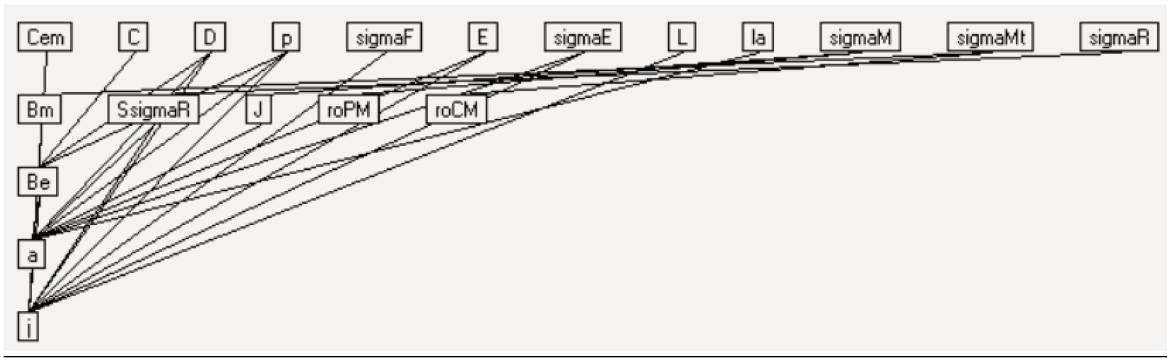


Figure 2.12 Arbre de causalité du problème de conception du moteur (CE).



— traiter un problème mixte ;

— étudier l'impact du choix de techniques pour la résolution selon les deux aspects précédents ;

— traiter un problème de dimensionnement ;

— traiter une optimisation mono-critère ;

Table 2.5 Résultats des optimisation mono-obj. de M_a et V_g (CE et benchmark).

ddl	Unités	Optimisation mono-obj.			
		M_a		V_g	
		[25]	**CE**	**[25]**	**CE**
D	mm	138.5	128.6	126.0	123.0
E	mm	6.0	10.0	8.1	7.0
L	mm	54.2	51.7	51.2	50.6
j	A/mm^2	5.7	5.9	4.8	6.0
a	mm	4.2	4.1	4.7	4.7
d	mm	4.2	4.7	4.2	4.0
p	/	9	7	7	7
la	mm	4.2	4.2	4.4	4.0
C	mm	4.6	4.2	4.6	4.4
σ_e	/	1	1	1	1
σ_f	/	1	1	1	1
σ_r	/	1	0	0	0
σ_m	/	1	0	1	1
σ_{mt}	/	0	0	1	1
Variables déduites					
B_e	T	0.53	0.41	0.58	0.56
K_s	kA/m	12.3	19.5	14.4	16.0
k_d	/	0.49	0.54	0.47	0.46
B_c	T	1.19	1.20	1.50	1.50
B_t	T	1.06	0.89	1.23	1.04
R_{int}	mm	59.5	50.0	50.2	50.0
R_{ext}	mm	79.8	73.7	73.0	70.9
V_c	$10^{-5}m^3$	21.6	17.0	18.4	16.9
V_t	$10^{-5}m^3$	7.4	2.90	7.2	1.95
V_{co}	$10^{-5}m^3$	5.3	6.3	5.6	4.9
V_a	$10^{-5}m^3$	7.9	7.8	7.8	7.0
Critères					
M_a	kg	2.84	2.20	3.16	2.49
V_g	$10^{-3}m^3$	1.08	0.88	0.86	0.80

— prendre en compte une contrainte catalogue.

Pour ce dernier cas d'étude, nous nous intéresserons à un problème mal-posé en termes de sur-contraintes, ainsi qu'aux conséquences sur la résolution en termes de techniques employées et de solutions obtenues. Cet exemple permet d'aborder deux difficultés de formalisation qui n'ont pas été rencontrées dans les deux cas d'étude précédents. Il permet d'appuyer la démarche de réflexion à réaliser autour de la formalisation. Les travaux présentés dans cette section ont fait l'objet d'une communication dans le congrès OIPE [157] et d'une

Table 2.6 Solutions optimales aux 2 problèmes bi-critères de la machine AC/DC brushless (CE et benchmark).

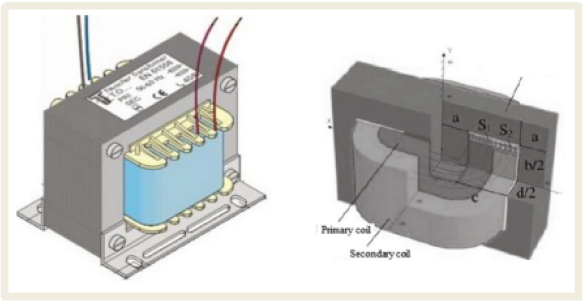
ddl	Unités	Optimisation multi-obj.		
		BB[25]	BP(CE)	
			Marglin	pondération
D	mm	130.2	118.4	128.6
E	mm	7.5	9.7	10.0
L	mm	50.1	50.0	52.3
j	A/mm^2	5.1	5.5	5.9
a	mm	4.0	4.3	4.0
d	mm	4.0	4.0	4.8
p	/	8	8	7
la	mm	4.2	4.0	4.0
C	mm	5.6	4.1	4.1
σ_e	/	1	1	1
σ_f	/	1	1	1
σ_r	/	0	1	0
σ_m	/	1	1	0
σ_{mt}	/	0	0	0
Variables déduites				
B_e	T	0.59	0.50	0.40
K_s	kA/m	13.4	19.5	19.4
k_d	/	0.50	0.48	0.54
B_c	T	1.15	1.20	1.20
B_t	T	1.18	0.96	0.90
R_{int}	mm	52.0	50.0	50.0
R_{ext}	mm	75.9	73.0	73.5
V_c	$10^{-5}m^3$	22.8	15.9	17.0
V_t	$10^{-5}m^3$	7.3	2.9	2.9
V_{co}	$10^{-5}m^3$	5.1	7.1	6.3
V_a	$10^{-5}m^3$	7.6	6.0	7.6
Critères				
M_a	kg	2.85	2.25	2.21
V_g	$10^{-3}m^3$	0.91	0.84	0.88

publication dans la revue IJAEM [158].

La description complète de ce benchmark peut être retrouvée dans [79] [1]. Le problème concerne le pré-dimensionnement optimal d'un transformateur de sécurité abaisseur de tension $230V/24V$ $50\ Hz$ à isolation galvanique de facteur de puissance 0.8. Le transformateur est constitué de tôles E-I à grains orientés, tel que les enroulements primaire et secondaire

1. et sur http ://l2ep.univ-lille1.fr/come/benchmark-transformer.html.

Figure 2.13 Le transformateur de sécurité [156].



soient enroulés sur le support autour du noyau central comme le montre la Fig. 2.13. Le problème mono-critère consiste à minimiser la masse totale du transformateur M_{tot} , au regard de 7 ddl dont les valeurs sont à sélectionner dans un catalogue de composants et de 7 contraintes non-linéaires tel que :

- la température du cuivre T_{copper} soit inférieure à 120^oC (cf. Equ. 2.10) ;
- la température du fer T_{iron} soit inférieure à 100^oC (cf. Equ. 2.13) ;
- le rendement η soit supérieur à 80% (cf. Equ. 2.16) ;
- le courant de magnétisation $\frac{I_{10}}{}$ soit inférieure à 10% de la valeur du courant dans le circuit primaire I_1 (cf. Equ. 2.12) ;
- la chute de tension au secondaire ΔV_2 soit inférieure à 10% de la valeur de la tension au secondaire V_2 (cf. Equ. 2.11) ;
- le coefficient de remplissage de la bobine primaire f_1 soit inférieure à 50%, pour un fil rond (cf. Equ. 2.14) ;
- le coefficient de remplissage de la bobine secondaire f_2 soit inférieure à 50%, pour un fil rond (cf. Equ. 2.15).

$$T_{\text{copper}} \leq 120^oC \tag{2.10}$$

$$\frac{\Delta V_2}{V_2} \leq 0.1 \tag{2.11}$$

$$\frac{I_{10}}{I_1} \leq 0.1 \tag{2.12}$$

$$T_{iron} \leq 100^oC \tag{2.13}$$

$$f_1 \leq 0.5 \tag{2.14}$$

$$f_2 \leq 0.5 \tag{2.15}$$

$$\eta \geq 0.8 \tag{2.16}$$

Table 2.7 Catalogue de valeurs pour les tôles E-I et leur *type* correspondant.

type	*a*	*b*	*c*	*d*
1	0.0225	0.095	0.04	0.077
2	0.0225	0.095	0.04	0.0668
3	0.0225	0.095	0.04	0.057
4	0.02	0.08	0.035	0.0617
⋮	⋮	⋮	⋮	⋮

Parmis les modèles mathématiques disponibles, le modèle analytique avec l'hypothèse de Kapp, qui conduit à négliger la chute de tension créée par I_{10}, a été choisi dans le cadre de cette pré-conception.

Les modèles thermiques et électromagnétiques réalisent un couplage fort. En effet, les propriétés électriques et magnétiques dépendent de la température de fonctionnement (supposé en régime permanent, pas de transitoire). Du point de vue de la résolution, c'est ce couplage qui est généralement difficile à prendre en main pour la plupart des techniques d'optimisation, car elles s'appuient sur une résolution séquentielle des équations. Les boucles algébriques présentes requièrent ainsi l'utilisation de solveurs implicites qui vont simuler le comportement non-linéaire associé [159]. La formalisation de ce problème sous forme du CSP `P3OptMtot` est la suivante (l'ensemble C3 comprend également les équations du modèle présenté sur le site du L2EP indiqué précedemment) :

$$\texttt{P3OptMtot : CSP} = \begin{cases} X3 = \{a, b, c, d, \mathrm{n}_1, n_2, S_1, S_2, \ldots\} \\ D3 = \{\ \mathrm{d}_a, d_b, d_c, d_d, d_{n1}, d_{n2}, d_{S1}, d_{S2}, \ldots\} \\ C3 = \{Equ.\ 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, 2.16, \ldots\} \end{cases}$$

Formalisation du CSP P3OptMtot avec CE

Comme dans le cas d'étude précédent, chaque ligne d'un catalogue correspond à un tuplet de valeurs consistantes définissant le domaine d'un ddl. En introduisant un cinquième ddl *type* dont le domaine est formé par le tuplet $\{a, b, c, d\}$, on peut alors réduire le problème à 5 ddl (cf. Tab. 2.7). La Fig. 2.14 présente un modèle de synthèse pour le CSP `P3OptMtot`.

Figure 2.14 Modéle du problème P3OptMtot (CE).

```
Constant
pi=3.14159265358979323846;
f=50;
fp2=0.8;
acosfp2=acos(fp2);
v1=230;
v2=24;
i2min=8;
Text=40;
q=1;
kr=0.5;
h=10;
lambda=0.15;
eisol=1e-3;
rhocuivre=1.72e-8;
alphacuivre=3.8e-3;
mvfer=7800;
mvcuivre=8800;
muo=4*pi*1e-7;
Function
mur(_x):=1/(2.12e
    -4+(1-2.12e-4)*_x
    ^(2*7.358)/(1.18e6+_x
    ^(2*7.358)));
Variable
type:enumint={1,2,..,62};
a:enumreal
    ={0.002,..,0.0225};
b:enumreal
    ={0.006,..,0.095};
c:enumreal
    ={0.0035,..,0.04};
d:enumreal
    ={0.0052,..,0.465};
s1:enumreal={0.05515e
    -6,..,19.635e-6};
s2:enumreal={0.05515e
    -6,..,19.635e-6};
n1:int=[200,1200];
n2:int=[1,10000];
r1:real=[0,inf];
r2:real=[0,inf];
i2:real=[0,inf];
Alias
#l1spire:=2*(d+2*a)+pi*c
    /2;
#Sferair:=4*a*(b+4*a+2*c)
    +2*d*(6*a+2*c+b);
#Scuivreair:=b*(4*a+2*pi*c
    );
#Mfer:=4*a*d*(2*a+b+c)*
    mvfer;
#Rcond:=eisol/(lambda*b
    *(4*a+2*d));
#l2spire:=2*(d+2*a)+pi*c
    *3/2;
#Bm:=1/4*v1*sqrt(2)/(n1*a*
    d*pi*f);
#R2:=r2+(n2/n1)^2*r1;
#X2:=muo*n2^2*c*(4*a+2*d+
    pi*c)*2*pi*f/(3*b);
#DV2:=i2*(#R2*fp2+#X2*sin(
    acosfp2));
#Rcuivreair:=1/(h*#
    Scuivreair);
#Rferair:=1/(h*#Sferair);
#Lmu:=muo*n1^2*a*d/(2*a+b+
    c)*mur(#Bm/1);
#Pfer:=q*#Mfer*f/50*(#Bm
    /1)^2;
#Pj:=#R2*i2^2;
#P1:=#Pfer+#Pj+v2*i2*fp2;
#Q1:=v1^2/#Lmu/2/pi/f+#X2*
    i2^2+v2*i2*sin(acosfp2
    );
#Tfer:=Text+#Rferair*(#
    Rcuivreair*#Pj+(#
    Rcuivreair+#Rcond)*#
    Pfer)/(#Rcuivreair+#
    Rferair+#Rcond);
#Tcuivre:=Text+#Rcuivreair
    *(#Rferair*(#Pj+#Pfer)
    +#Pj*#Rcond)/(#Rcond+#
    Rcuivreair+#Rferair);
#i10:=sqrt((#Pfer/v1)^2+(
    v1/#Lmu/2/pi/f)^2);
#i1:=sqrt(#P1^2+#Q1^2)/v1;
#Mcuivre:=mvcuivre*(n1*s1
    *#l1spire+n2*s2*#
    l2spire);
#Mtotale:=#Mcuivre+#Mfer;
#ren:=v2*i2*fp2/(v2*i2*fp2
    +#Pfer+#Pj);
#iratio:=#i10/#i1;
#Vratio:=#DV2/v2;
#kr1:=2*n1*s1/(b*c);
#kr2:=2*n2*s2/(b*c);
Constraint
table(5,type,a,b,c,d,
    tabledim);
r2 =rhocuivre*(1+
    alphacuivre*#Tcuivre)*
    n2*#l2spire/s2;
r1=rhocuivre*(1+
    alphacuivre*#Tcuivre)*
    n1*#l1spire/s1;
n2 =n1*(v2+#DV2)/V1;
#Tfer>=0;
#Tcuivre>=0;
#ren>=0.8;
#ren<=1;
#Tcuivre<=120;
#Tfer<=100;
#Vratio<=0.1;
#iratio<=0.1;
#kr1<=kr;
#kr2<=kr;
i2 >=i2min;
#Mtotale<=2.6;
```

Analyse causale (P3OptMtot) L'analyse causale a permis de déceler une sur-contrainte dans le problème tel qu'il était initialement posé. Il n'y avait donc pas de solution admissible à ce problème. Pour le résoudre, il fallait donc relaxer une ou plusieurs contraintes de conception. Le choix de(s) contrainte(s) à relaxer est généralement établi à partir des connaissances et de l'expérience (espace des connaissances). De la même manière que nous avons relaxé la contrainte d'égalité sur le couple électromagnétique pour le problème `P1sol`, la contrainte sur le courant du circuit secondaire $i_2 = 8A$ a été relaxée en une contrainte de fonctionnement minimale : $i_2 \geq 8A$. Ceci explique la valeur de cette dernière dans la Tab. 2.8. En effet, on cherchera en général à ce que le système puisse fournir un minimum de puissance à la charge. Cette puissance est dépendante de i_2. L'analyse causale a également révélé que seules les $type, n1, s_1$ et s_2 étaient de vrais ddl et que leurs instantiations suffisaient à résoudre le problème.

Table 2.8 Solution optmale obtenue avec CE (BP) (transformateur).

ddl							
type	a (mm)	b (mm)	c (mm)	d (mm)	S_1 (mm^2)	S_2 (mm^2)	n_1 (turns)
27	18	54	18	33.5	0.2463	2.54	611
Variables déduites							
R_1 (Ω)	I_1 (A)	R_2 (Ω)	I_2 (A)	n_2			
10.105	0.97778	0.14778	8.0288	69			

Table 2.9 Comparaisons des résultats obtenus CE et benchmark (transformateur).

ddl	Opt. directe SQP	Pro@Design	EE	GA	BB	**BP (CE)**
a (mm)	14.94	18	18	18	18	18
b (mm)	48.43	54	54	54	54	54
c (mm)	15.64	18	18	18	18	18
d (mm)	38.27	33.5	33.5	33.5	33.5	33.5
S_1 (mm^2)	0.310	0.3318	0.2827	0.2376	0.2827	0.2463
S_2 (mm^2)	2.782	2.835	2.27	2.835	2.27	2.54
n_1 (tours)	611.78	722	610	614	611	611
Critère						
M_{total} (kg)	2.294	2.840	2.594	2.633	2.594	2.592
t (s)	4	38	$14 \cdot 10^6$	$3 \cdot 10^3$	816	≤ 8
Exigences de performances						
T_{copper} (^{o}C)	120.65	103.60	109.20	106.25	109.20	109.45
T_{iron} (^{o}C)	100.14	94.19	99.58	97.13	99.58	99.78
η (%)	89.24	88.54	87.60	88.04	87.60	87.58
$\frac{I_{10}}{I_1}$ (%)	10.09	4.90	9.98	9.58	9.98	9.83
$\frac{\Delta V_2}{V_2}$ (%)	7.60	8.20	8.20	7.86	8.20	8.22
f_1 (%)	50.06	49.29	35.50	30.20	35.50	30.96
f_2 (%)	50.09	47.56	32.20	37.98	32.20	36.13

Résolution (P3OptMtot)

Pour la résolution, la *fail-first* associée à la HC ont été utilisées. La solution optimale a été obtenue en mois de 8 secondes. Le Tab. 2.9 regroupe les solutions obtenues par les différentes techniques de résolution (CE et benchmark [79]). Les résultats présentés Tab. 2.9 peuvent également être retrouvés dans [158]. La solution obtenue par BP avec CE est admissible et optimale si on la compare aux résultas trouvés avec les autres techniques. On peut constater que toutes les contraintes de conception imposées par la spécification des exigences ont toutes été respectées. On peut voir que T_{iron} a presque atteint la valeur limite lors de l'optimisation. Pour le domaine de T_{iron}, on peut se demander quel demi-intervalle est exploré en premier après bissection, à savoir celui à droite ou à gauche du centre (ie. quel heuristique d'ordonnancment de valeurs a été appliquée). En effet, on peut observer que pour toutes les solutions optimales proposées, la température du fer est le paramètre de conception le plus critique. On sait désormais que l'exploration devra s'effectuer d'abord dans le demi-intervalle de droite, là où une valeur consistante de T_{iron} est plus susceptible d'être trouvée. On espère ainsi potentiellement améliorer d'avantage les temps de résolution.

Comparaisons entre résultats obtenus dans ces travaux de thèse et ceux des autres équipes (benchmark)

On peut voir que les techniques de résolution exactes de type Exhaustive Enumeration (EE), Branch and Bound (BB), Branch and Prunes (BP) (CE), stochastique Genetic Algorithm (GA) ainsi que Pro@Design trouvent des optimums très proches (mêmes valeurs pour les ddl). Toutes les exigences de performances sont respectées. Les résultats obtenus avec CE sont très proches de ceux trouvés par BB, comme pour les précédents cas d'études. Les principales différences entre les 5 techniques cités précedemment résident d'une part dans les valeurs obtenues pour les sections des bobinages et de n_1. D'autre part, les temps de résolution sont particulièrement importants pour les techniques EE et GA. Les techniques d' EE permettent une gestion des problèmes discrets, mais mènent généralement à une explosion combinatoire pour les problèmes de grandes dimensions. D'un autre côté, les techniques SQP, qui considèrent les ddl discrets comme des variables continues ne permettent d'obtenir que des pseudo-solutions optimales.

2.8 Conclusion

Avec la Programmation Par Contraintes (PPC), le problème de conception peut être représenté de manière explicite et acausale sous forme de CSP. La formulation CSP permet également de poser des contraintes globales et d'utiliser des modèles type boîtes blanches implicites. Mais pour pouvoir construire des modèles de synthèse de systèmes réels efficacement, il faut s'aider des 5 piliers de la formalisation de manière intégrée : un formalisme, un environnement de formalisation, un solveur de contraintes mixtes et un environnement de résolution. Egalement, il faut que le formalisme permette une amélioration de la formulation CSP en étendant celle-ci avec la prise en compte de variables déduites.

En effet, nous avons pu voir qu'un triplet (X, D, C) permet difficilement de spécifier un problème d'ingénierie en capturant la structure du système de manière lisible puisque les composants n'apparaissent pas. La formulation CSP est un modèle *à plat* du modèle gros grains, ce qui ne permet pas de réutiliser les relations du modèle comme par exemple les contraintes. La mise en forme d'un problème de conception, depuis le passage du modèle gros grains au modèle de synthèse, en passant par le choix des ddl, la modélisation des contraintes ou encore la prise en compte de l'optimisation, est une difficulté qui apparaît même sur des problèmes d'apparence simple. Il est notamment difficile de cerner la vraie compléxité du problème (combien de variables indépendantes ?) en raison de la très faible organisation des modèles. Les problèmes de conception présentés ne considèraient par ailleurs qu'un seul point de fonctionnement (le nominal). Il serait intéressant par la suite d'étudier des problèmes de pré-conception où le système physique doit être conçu selon un profil de mission donné. Pour la formalisation des problèmes, il est nécessaire de pouvoir utiliser un formalisme suffisamment expressif permettant d'exprimer des contraintes mixtes et des contraintes catalogue.

Les trois cas d'études tirés de la littérature et étudiés dans ce chapitre ont permis d'illustrer les nombreuses et importantes conséquences de la formalisation sur la résolution lors de la conception. Le formalisme CE et ses outils d'aide à la résolution ont permis notamment d'illustrer une démarche de réflexion et les choix de conception depuis la formalisation jusqu'à la résolution. Il est nécessaire de prendre en compte les deux aspects simultanément pour une pré-conception efficace. En particulier, un formalisme comme CE qui étend la formulation CSP en fait un langage plus adapté pour la conception que la formulation CSP. Les modèles de synthèse sont mieux organisés et ont un meilleur niveau de généricité grâce à

la réutilisation possible de contraintes avec les notions de *Constant*, *Function* et *Alias*. Les temps de modélisation sont ainsi réduits.

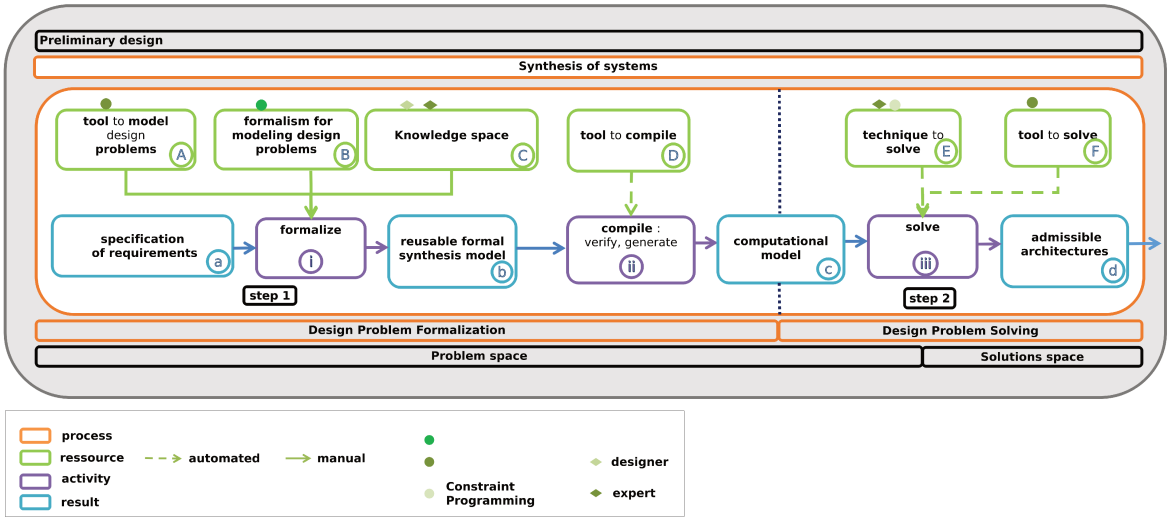
La résolution en PPC (quelque soit la formulation utilisée) est cependant très dépendante de la stratégie de recherche ainsi que des choix d'heuristiques. Sans une véritable connaissance des techniques employées, elle peut devenir moins efficace en termes de temps de résolution. La prise en main de tous ces concepts peut aussi être difficile pour les concepteurs néophytes. Egalement, les modèles construits dans un formalisme proche de celui des CSP peuvent rapidement devenir illisibles et restent généralement plats et peu réutilisables car l'organisation des modèles reste faible. Ils sont structurés mais trop peu pour permettre une réutilisation efficace de ceux-ci. Ces modèles présentent en effet une grande quantité d'informations visuelles organisées de manière *plate* (un jeu d'(in)équations). Il faudrait donc pouvoir occulter certains détails d'implémentations, pour ne pouvoir agir que sur ceux déterminants lors de la conception. Egalement, la PPC ne permet pas de définir des quantités physiques, qui sont pourtant primordiales pour décrire les systèmes physiques. Les problèmes généralement traités en PPC sont discrets et se limitent à l'utilisation des types réel ou entier. Il est ainsi nécessaire de disposer de formalismes de plus haut niveau pour la conception.

Pour répondre au premier verrou, nous retiendrons que la PPC est adaptée pour la résolution, mais qu'il faut un formalisme de plus haut niveau que les CSP et que CE pour représenter des problèmes de conception de manière plus lisibles, génériques et réutilisables. On peut en effet trouver l'ensemble des solutions admissibles voire optimales d'un problème avec la PPC et qui sont *correctes-par-construction*, de manière garantie. Le chapitre 3 présentera une approche de synthèse de la formalisation à la résolution pour les systèmes physiques structurés, avec un accent mis sur une meilleure structuration des modèles de synthèse construits à partir de modèles de type boîte blanche gros grains algébriques. Les problèmes de conception adressés seront ceux du cadre d'étude de ces travaux de thèse, à savoir mixtes, non-linéaires et avec catalogues. Cette approche s'interessera également à la conception selon plusieurs points de fonctionnement sous forme de profils.

Chapitre 3

Représentation structurée des problèmes de conception

Figure 3.1 Formalisation structurée et ré-utilisable des problèmes de conception.



Dans ce troisième chapitre, nous cherchons à répondre au second verrou : la modélisation de problème en GE étant peu lisible et peu réutilisable, comment en améliorer la généricité et le niveau d'abstraction ? Nous nous intéressons ainsi dans ce chapitre à des formalismes plus évolués que les CSPs et que CE mais qui conservent une résolution par contraintes. Les expériences de modélisation nous ont montrés que les relations du problèmes exprimées sous forme de contraintes étaient intéressantes. On cherche donc à améliorer l'organisation des modèles de synthèse mais sans perdre cet aspect. Les outils de conception intégrés seront privilégiés comme au chapitre précédent. En particulier, le formalisme retenu devra être

déclaratif (formalisme de modélisation et non de programmation, comme CE) et permettre de construire des modèles de synthèse génériques et réutilisables pour d'autres projets de conception. Il faut aussi considérer les outils de compilation et de résolution nécessaires à la vérification, à la transformation et à la résolution de ces modèles réutilisables. Nous cherchons ainsi à compléter la Fig. 3.1 par un formalisme (B), un outil de modélisation (A) / résolution (F) et de vérification/compilation de modèles (D). Le résultat (b) de l'activité de formalisation (i) devient un modèle réutilisable en plus d'être formel.

3.1 Formalismes étendus associés à la PPC

Dans cette section, nous présenterons les différentes bibliothèques d'objets, interpréteurs CLP et formalismes étendus associés à la Programmation Par Contraintes (PPC).

3.1.1 Bibliothèques d'objets et fonctions dédiées à la PPC

Les bibliothèques d'objets et fonctions dédiées à la PPC disponibles sont écrites dans des langages génériques de programmation orientée-objet tels que le C++, Python ou encore Java. Comme mentionné au chapitre précedent, elles ne s'interfacent avec aucun solveur de contraintes en particulier, ce qui permet de pouvoir utiliser directement les compilateurs et interpréteurs existants. Les programmes compilés peuvent ensuite être résolus via l'utilisation de solveurs de contraintes génériques tels que Gurobi [140]. Quelques exemples de bibliothèques et outils de ce type sont référencés ci-dessous :

La bibliothèque Elisa [160] La bibliothèque Elisa s'utilise avec un solveur de contraintes continues. Même s'il est mentionné la possibilité de définir des Variables discrètes, il faut tout de même en pratique modifier le solveur pour pouvoir gérer les contraintes mixtes.

Labix [161] La bibliothèque et solveur Labix est écrite en Python et est dédiée à la résolution de CSP discrets. Il n'est pas possible de spécifier des ensembles de valeurs pour les domaines discrets, ni de contraintes globales. La bibliothèque est aussi peu extensible, ce qui ne permettrait pas de facilement y implémenter des variables, domaines et contraintes continues.

Choco [162] La bibliothèque et solveur extensible Choco est écrite en JAVA et est dédiée aux domaines discrets. Il est tout de même possible de déclarer des contraintes globales. Certains travaux ont proposé des extensions aux domaines continus en l'utilisant conjointement

avec le langage de programmation logique ECLiPSe [163] ou encore avec le langage Ibex (cf. Chapitre 2) pour permettre la gestion des domaines mixtes. Cependant, l'implémentation en pratique s'est montrée difficile [47]. D'autres aspects ne permettent pas de facilement spécifier un problème de conception, notamment au regard des opérateurs arithmétiques disponibles. Ils sont principalement linéaires, avec une gestion difficile des opérateurs trigonométriques (erreurs de calculs) et seules les contraintes d'égalités peuvent être spécifiées. Certains travaux mentionnent également la difficulté dans la prise en main du formalisme et le manque de lisibilité des modèles de synthèse construits [46]. Il est cependant possible de résoudre un problème d'optimisation mono- et bi-critère avec Choco, en rajoutant après chaque étape de résolution une contrainte.

Gecode [164] La bibliothèque Gecode est écrite en C++ et est dédiée à la résolution sur des domaines discrets. En revanche, les modèles spécifiés en Gecode sont assez naturels à construire et les problèmes discrets de grande taille y sont très bien gérés.

L'inconvénient majeur d'une bibliothèque est qu'il est nécessaire de maîtriser le langage orienté-objet associé. Comme nous souhaitons utiliser un formalisme suffisamment simple à prendre en main pour les concepteurs (qui ne nécessite pas de connaissances particulières en programmation), nous ne retiendrons pas les bibliothèques et fonctions dédiées. En effet, puisque les formalismes et outils que nous recherchons sont dédiés aux concepteurs métiers, nous savons qu'ils ne les utiliseront pas si ils sont trop difficiles à prendre en main. De plus, très peu d'entre eux permettent le calcul sur des domaines mixtes. Du point de vue de la réutilisabilité, les bibliothèques permettent une réutilisabilité pour les programmeurs mais pas pour les concepteurs. Ces bibliothèques permettent donc de bâtir des applications dédiées et non pas des modèles de synthèse réutilisables, dans l'hypothèse où l'on maîtrise déjà les langages C++ ou Java.

3.1.2 Interpréteurs CLP

Les interpréteurs Constraint Logic Programming (CLP) [165] regroupent à la fois des formalismes et outils de résolution. Les formalismes CLP sont issus de Prolog et sont déclaratifs. Cependant, les modèles sont construits à partir de prédicats et reposent ainsi sur de solides bases mathématiques, ce qui ne permet pas réellement d'en faire un langage dédié à la conception pour les ingénieurs bureau d'études. De manière générale, la programmation logique avec contraintes semble difficile à maîtriser pour des concepteurs non informaticien.ne.s. Parmi les très nombreux formalismes de programmation logique sous contraintes

issus du langage Prolog, la plupart sont dédiés à la résolution de problèmes discrets. ECLiPSe [163] est en théorie un des seuls à pouvoir traiter des problèmes continus. Contrairement à Prolog, il est possible d'y spécifier des contraintes arithmétiques. ECLiPSe permet également l'expression de contraintes globales telles que *alldiff*.

3.1.3 Formalismes étendus

Cette sous-section présente les formalismes étendus qui ne rentrent dans aucune des catégories précédentes (bibliothèques, interpréteurs CLP) mais qui peuvent s'interfacer avec des solveurs de contraintes génériques. Il s'agit donc d'un avantage intéressant puisque le choix du formalisme n'est pas contraint par celui du solveur de contraintes. Ce sont des formalismes plus haut-niveaux que CE utilisant des notions que l'on retrouvent en programmation orienté-objet type C++, telles que les listes.

Zinc permet de traiter des CSP / CSOP discrets et propose deux autres formalismes plus bas-niveaux, MiniZinc et FlatZinc [166, 167]. Les modèles écrits en MiniZinc permettent une expression assez naturelle de la plupart des CSP / CSOP mais reste tout de même assez bas-niveau pour permettre un interfaçage facile avec de nombreux solveurs. Les modèles sont ensuite compilés en FlatZinc pour les solveurs, qui acceptent chacun leurs propres versions de FlatZinc. Par exemple, les modèles écrits en langage EssencePrime peuvent être compilés en Minion Input Language ou encore en FlatZinc pour une résolution avec Gecode.

CHR [168] est moins évident à classer car ce n'est ni un langage, ni un outil, ni une bibliothèque ni même un solveur de contraintes. Il est présenté comme un formalisme déclaratif logique à base de règles. Cependant il est plutôt utilisé en pratique pour étendre d'autres formalismes pour CSP tels que Prolog mais également plus généraux comme le C, ou JavaScript. Il est également utilisé pour écrire des solveurs de contraintes plutôt que d'en être réellement un [169]. L'utilisation de CHR est très variée et s'étend au-delà du paradigme de la PPC, notamment pour les problèmes de test et de vérification. CHR est adapté à la formalisation de CSP discrets. En théorie, les contraintes globales et les domaines continus peuvent être spécifiés, mais l'implémentation de CHR est plutôt orientée vers les formalismes pour CSP discrets de type Prolog. L'inconvénient principal est qu'il est nécessaire de trouver un solveur adéquat, sinon il faut en écrire un.

Comet [170] est un autre formalisme dédié aux CSP et CSOP discrets, plus précisément aux problèmes de *satisfabilité*, qui sont résolus par l'utilisation de solveurs de contraintes

booléennes Satisfactory (SAT) [171].

Numberjack [172] est un formalisme et une librairie écrite en Python pour CSP et CSOP discrets. Les nombreux solveurs visés sont écrits en C/C++ et sont soit dédiés aux CSPs, comme Toulbar2 [173] et Mistral [174] (qui n'est actuellement plus maintenu), soit aux problèmes d'optimisation linéaires MILP avec Gurobi, CPLEX et SCIP ou encore aux problèmes de satisfabilité avec des solveurs SAT.

Ces formalismes étendus nécessitent donc tout de même quelques notions de programmation et sont généralement dédiés à la résolution des CSP/CSOP discrets. L'inconvénient est qu'il faut transformer plusieurs fois les modèles de synthèse construits pour que ceux-ci puissent être compilés par des solveurs tiers tels que Gecode. Nous ne retiendrons pas ces formalismes pour notre approche de conception.

De manière plus générale, si l'on souhaite exprimer la variabilité d'un problème mixte, on peut se tourner vers d'autres langages existants issus de divers domaines. Certaines caractéristiques des langages MBSE comme SysML et Modelica peuvent également être intéressantes pour une spécification naturelle des problèmes de conception des systèmes physiques : haut-niveau, déclaratif, orienté-objet, ou encore la possibilité d'exprimer des quantités issues de la physique (et pas uniquement des types informatiques). Différentes expériences de modélisation nous ont conduit à considérer un formalisme textuel plutôt que graphique (notamment celle présentée en Annexe E). Ainsi, l'ensemble de ces critères sera considéré lors du choix de formalismes et de solveurs associés depuis l'espace des connaissances.

Il existe de nombreux types de langages permettant d'exprimer la variabilité de manière générale : mathématique, de programmation, impératifs, fonctionnels, formels, généraux, dédiées, algébriques, etc. En ce qui concerne les langages pour la conception, la majorité, si ce n'est tous, ont été développés pour l'ingénierie des systèmes logiciels (IL) et très peu, voire aucun, pour celles des systèmes physiques (IS).

3.1.4 Modélisation de la variabilité en Ingénierie des Logiciels

Cette sous-section dresse un état de l'art autour de la formalisation des problèmes de conception en Ingénierie des Logiciels (IL).

Les premiers langages apparus pour la formalisation sont graphiques et reposent sur les *Features Models* et l'approche de modélisation de problèmes Feature-Oriented Domain Analysis (FODA) [27]. Les modèles, présentés sous forme d'arbres, sont faciles à comprendre et la spécification du problème est intuitive. Cette approche permet de modéliser le fait qu'un constituant d'un système soit obligatoire, optionnelle, à choix alternatif ou à choix multiples. Ces formalismes montrent cependant leurs limites lorsque de nombreuses alternatives d'architectures doivent être modélisées. De plus, la variabilité ne peut être que discrète (configuration structurelle). Par la suite, les formalismes textuels se sont de plus en plus développés.

Plusieurs langages et modèles ont été développés en IL pour représenter la variabilité en fonction d'un modèle de base (modèle indépendant), notamment deux langages similaires : le langage Orthogonal Variability Model (OVM), qui cherche à définir les propriétés communes et variables de tous les produits d'une même famille, ou encore le langage Common Variability Language (CVL) [175].

[64] compare les capacités de dix formalismes textuels (parmis 17) à exprimer la variabilité des problèmes de conception de systèmes logiciels. Les 7 autres langages ne sont pas inclus dans la comparaison, principalement par manque de spécification formelle, par difficulté de prise en main ou en raison de leur dépendance à des langages de spécification de systèmes. Ces formalismes textuels permettent de mieux modéliser les problèmes de conception de systèmes logiciels que leurs homologues graphiques [64]. En particulier, ils permettent une meilleure expression des contraintes, des problèmes de grande taille, et ils supportent mieux le typage de données. La nécessité de mettre en place des mécanismes pour exprimer les problèmes de grande taille est soulignée, notamment à travers la relation de composition, l'évolution des modèles (la possibilité d'adapter un modèle) et la modularisation. Les formalismes textuels sont également préférables pour améliorer la lisibilité des modèles.

Les principes de base utilisés pour la modélisation des problèmes de conception de systèmes à dominante logiciels peuvent être repris pour celle des systèmes à dominante physique. De cette revue, on peut extraire les caractéristiques d'un formalisme efficace pour l'expression de problèmes de conception. Un tel formalisme doit fournir au moins une manière de permettre :

- — les liens de composition et de référence ;
- — l'héritage ;
- — le typage de données ;

- — les contraintes arithmétiques et relationnelles (ces dernières pour les types continus) ;
- — l'affectation de valeurs ;
- — la modularité.

De plus, le langage ne devrait pas reposer majoritairement sur une structure de type graphe ou arbre, et doit permettre l'expression de commentaires. Seul le formalisme Integrated Variability Modeling Language (IVML) [176] semble couvrir tout ces critères. IVML s'inspire de plusieurs langages issus de divers domaines, notamment : de programmation (Java), de contraintes (OCL), de spécification de systèmes logiciels (UML), de spécification de problèmes de conception de systèmes logiciel (TVL [1], CLAFER [2] [29]). Cependant, IVML reprend aussi les inconvénients de la syntaxe de OCL pour l'expression des contraintes de conception. Celles-ci sont exprimées en logique propositionnelle, ce qui n'est pas idéal pour les concepteurs métiers. Le formalisme textuel Alloy [177] a aussi été écarté car il ne permet de n'exprimer que des contraintes booléennes à l'heure actuelle.

3.1.5 Solveurs et formalismes

Il n'y a à ce jour que peu de solveurs développés avec un formalisme haut-niveau pour la résolution des problèmes de conception mixtes qui regrouperait l'ensemble des caractéristiques mentionnées auparavant.

Parmi les récents projets en cours de développement, nous pouvons compter le solveur de contraintes mixtes DEPS Studio [137] et son formalisme dédié DEPS [178, 179], tous deux liés à l'encadrement de cette thèse. Ce sont des tentatives d'outillage pour une approche de synthèse pour la conception de systèmes physiques (ex : mécaniques), à dominante logicielle, mécatroniques, embarqués et cyber-physiques. DEPS Studio est développé dans le cadre de l'association DEPSLab, qui regroupe les développeurs du Studio. Le langage DEPS est quant à lui développé au sein de la DEPS Link Association [3] regroupant les contributeurs et utilisateurs du formalisme. DEPS Studio est un environnement intégré de formalisation et de résolution. Celui-ci inclut un compilateur et un solveur de contraintes mixtes basé sur des techniques d'exploration de type BP. DEPS est un formalisme textuel déclaratif haut-niveau et orienté-objet dédié aux problèmes de conception mixtes.

1. Textual Variability Language.
2. CLAss FEatuRe modeling, qui ne permet actuellement que de travailler avec des domaines discrets.
3. www.depslink.com.

DEPS reprend les mécanismes d'abstraction des données et de structuration de l'orienté-objet pour représenter l'organisation d'un système physique (sous-) défini. Les concepts de la modélisation orientée-objet supportés par DEPS sont ceux de classe (appelée `Model` en DEPS), d'instance (appelée `Element` en DEPS), d'agrégation et de composition d'héritage simple, de modularité et de polymorphisme de `Model` à l'aide de signature. Le polymorphisme de `Model` est également supporté à l'aide de signature, ainsi que le typage des Variables. En particulier, il est possible de manipuler des quantités de la physique et de la technologie. Ces quantités peuvent aussi bien être ordinales que cardinales et sont plus proches du vocabulaire des concepteurs de systèmes physiques que les types purement informatiques.

Pour l'expression des contraintes (appelée *properties* en DEPS), DEPS reprend les notions mathématiques nécessaires à la résolution pour modéliser les propriétés gouvernant un système, comme par exemple l'utilisation d'opérateurs algébriques. DEPS utilise différents opérateurs : arithmétiques (+, -, /, *), d'égalité / inégalité et algébriques. Ces opérateurs algébriques sont définis par la norme IEEE 754. Ils portent sur des domaines de valeurs qui peuvent être de 6 types : valeurs entières, valeurs réelles, intervalles d'entiers, intervalles de réels, énumérés d'entiers et énumérés de réels. Une contrainte catalogue peut aussi être spécifiée.

Le langage DEPS est conçu pour l'expression et la résolution de problèmes par synthèse [179], à l'aide de modèles. DEPS permet d'exprimer naturellement les problèmes de conception et apparaît comme assez facile à prendre en main par des concepteurs métiers. L'utilisation de DEPS / DEPS Studio est prévue pour la pré-conception, afin de gérer les problèmes de dimensionnement, de configuration, de synthèse d'architecture et d'allocation. DEPS et DEPS Studio ont déjà été utilisés en aéronautique et robotiques pour des problèmes discrets et continus mais pas en GE ni pour des problèmes mixtes. Il est également possible d'utiliser le solveur CE en plus de celui intégré et natif de DEPS Studio pour résoudre les CSP spécifiés avec DEPS. DEPS et DEPS Studio sont en cours de développement mais ils étaient tous deux disponibles pour être utilisés dans le cadre de cette thèse.

Au regard de la formalisation et de la résolution des problèmes de conception d'ingénierie, nous avons choisi les formalisme DEPS et outil de résolution DEPS Studio afin d'illustrer l'approche de formalisation et de résolution de ces problèmes via la synthèse de systèmes.

3.2 Formalisation de problème de conception

3.2.1 Concepts de la modélisation orientée-objet

Représenter un problème ou des systèmes par des objets permet de comprendre plus naturellement le problème de conception. Un objet est une entité du monde que l'on souhaite représenter de manière abstraite (ex : un transistor). Il peut s'agir d'un système, d'un constituant, d'un système de système, d'une personne, d'une organisation, etc. La structure de ces objets privilégie l'évolution des modèles orientés-objets construits, qui s'adaptent mieux aux changements (par exemple de spécification) et à leur réutilisation. La modélisation orientée-objets [180] est donc adaptée pour développer des systèmes complexes. Elle repose sur 4 concepts : l'abstraction des données dans des classes (modèle d'objet, lien de composition et de référence, instanciation, navigation), l'héritage simple ou multiple (généralisation / spécialisation des modèles), le polymorphisme (signatures de modèle, surcharge de fonctions) et la modularité (découpage en modules : définition de packages). Un modèle d'objet est constitué de deux types d'informations :

- structurelles : ce sont les éléments qui caractérisent structurellement l'objet, typiquement des Variables et des constituants ;
- comportementales : ce sont les relations mathématiques (linéaire ou non, logiques, etc.) qui expriment le fonctionnement interne de l'objet, comme les lois constitutives d'un composant. Ces comportements sont dans la plupart des langages orientés-objets représentés à l'aide d'un ensemble de fonctions membres ou de méthodes.

3.2.2 Le formalisme DEPS

Model et Problem

Chaque modèle DEPS est organisé selon une partie structurelle et une partie dite de propriétés. Le mot-clé `Model` (resp. `Problem`), fait référence à un modèle de système (resp. à un modèle de problème de conception). Un modèle est construit comme suit : le mot-clé `Model` est suivi du nom du modèle puis de la liste de ses arguments entre parenthèses pour la construction de modèles paramétrés. Dans la suite de ce manuscrit, le terme `Models` sera employé pour désigner plusieurs modèles de constituants. La liste des arguments peut être constituée de constantes, d'instances d'autres `Models` ou être vide. Celle-ci est vide pour un `Problem`. Le mot-clé `End` indique la fin de la construction d'un modèle [179, 137, 181].

Partie structurelle : Constants, Variables et Elements Les paragraphes suivant décrivent la partie structurelle de tout modèle. Les constantes sont déclarées et définies dans la section `Constants`. Une déclaration ou une définition se termine par un point-virgule. Dans la section `Variables` sont déclarées les Variables. Les variables déduites sont aussi appelées expressions nommées. Une variable déduite est en effet représentée par une expression nommée (c'est une expression algébrique à laquelle un nom est donné). Elle est déclarée avec le mot-clé `expr`. Les éléments d'un système ou d'un problème peuvent être déclarées et créées dans la section *Elements*. Ce sont tous des instances de `Models` qui ont eux-mêmes déjà été créés auparavant.

Partie propriétés : Properties La partie propriété représente l'ensemble des propriétés nécessairement vérifiées par toute instance du modèle. Les contraintes fonctionnelles et non-fonctionnelles peuvent être définies dans la section *Properties* sous forme de relations algébriques : inéquations, équations et affectations. Notamment, on peut y définir des contraintes telles que *catalogue* sur des modèles de tables.

Représentation des grandeurs : Quantity et QuantityKind

Les constantes et les Variables doivent être déclarées avec un type, qui correspond à une grandeur physique, technologique, ordinale ou cardinale. Une grandeur est représentée par le concept de `Quantity`. Les constantes et Variables sont déclarées de la manière suivante : "`name : Quantity ;` " avec `name` le nom donné à la constante ou Variable. Une constante (resp. un ddl) est défini(e) par une valeur (resp. un domaine de valeurs) de la manière suivante : " `name : Quantity = value ;` " (resp. *in* `[domain-of-values];`).

La structure d'une `Quantity` est illustrée Fig. 3.2 avec l'exemple de la grandeur `INTENSITY` pour l'intensité du courant électrique, de dimension I et d'unités A (pour Ampères). Une `Quantity` est une structure de données composée de quatre sections : `Kind`, `Min`, `Max` et `Unit`. `Kind` désigne le type de la `Quantity` ou `Quantitykind`, qui permet de renseigner la dimension (comme utilisée dans l'analyse dimensionnelle).

Un `Quantitykind` (ex : `ELECURINTENSITY` pour l'intensité du courant électrique) est un modèle constitué de quatre sections : `Type`, `Min`, `Max` et `Dim`. Un type de base est renseigné dans *Type*. Pour une `Quantity` et un `Quantitykind`, les valeurs minimales (resp. maximales) que peuvent prendre une constante ou Variable typée par cette `Quantity` sont indiquées dans `Min` (resp. `Max`). Dans `Unit` est renseignée l'unité de la `Quantity`. L'ab-

Figure 3.2 Quantity et QuantityKind (DEPS).

```
248 QuantityKind ELECURINTENSITY
249 Type : real ;
250 Min  : -maxreal ;
251 Max  : +maxreal ;
252 Dim  : I;
253 End

255 Quantity INTENSITY
256 Kind : ELECURINTENSITY ;
257 Min  : -maxreal ;
258 Max  : +maxreal ;
259 Unit : A ;
260 End
```

Figure 3.3 Un Model spécialisé en DEPS.

```
261 Model ExtendedModelName(arguments list) extends ModelName[
       ModelNameSignature]
262 Constants
263 constant1Name : Quantity1 ;
264 ...
265 Variables
266 variable1Name : Quantity2 ;
267 variable2Name : Quantity3 in [Value3, Value4] ;
268 ...
269 Elements
270 element1Name : ModelName1[Model1NameSignature] ;
271 element2Name : ModelName2(arguments list) ;
272 ...
273 Properties
274 * a set of design constraint * ;
275 End
```

sence d'unité est indiquée par le caractère "u". Dans `Dim` est renseignée la dimension de la `Quantity`. L'absence de dimension est indiquée par le caractère "U".

Héritage / spécialisation de Model

DEPS supporte le concept d'héritage, qui est simple et publique pour le langage. Les constantes, Variables, élements et propriétés d'un `Model` parent sont hérités pour toute instance d'un `Model` spécialisé. Les valeurs des constantes héritées peuvent être redéfinies dans le `Model` spécialisé. Les domaines des quantités, des constantes et des Variables héritées peuvent être redéfinis dans le modèle spécialisé (en respectant l'inclusion). La syntaxe minimale d'un `Model` spécialisé est donné Fig. 3.3 et celle d'un `Problem` Fig. 3.4.

Figure 3.4 Un Problem en DEPS.

```
Problem ProblemName
Constants
constant1Name : Quantity1 = Value1 ;
...
Variables
variable1Name : Quantity2 = Value2 ;
variable2Name : Quantity3 in [Value3, Value4] ;
...
Elements
element1Name : ModelName1(arguments list) ;
...
Properties
* a set of design constraint * ;
End
```

Les expériences de modélisations du Chapitre 2 nous ont montré notamment qu'une des difficultés importantes de la formalisation est de rendre réutilisable des modèles initialement *plats*, sans avoir à re-formaliser tout le problème lorsque la spécification des exigences est modifiée ou encore pour d'autres projets. Une modélisation générique doit être réfléchie et bien pensée en amont lors de la construction des tout premiers modèles de synthèse. Les Sous-sections 3.2.3 et 3.2.4 présentent des étapes et recommendations générales dans le cadre d'une aide à la formalisation générique et réutilisable des problèmes de conception de systèmes physiques, en particulier pour le GE. Celles-ci s'adressent aux constructeurs de modèles mais aussi aux utilisateurs de ceux-ci. Les objectifs sont notamment d'aider à réduire les efforts de modélisation sur le long terme en misant sur la généricité et la réutilisabilité des modèles de synthèse construits et également d'aider au déboggage des modèles de systèmes dont sous-définis.

3.2.3 Etapes de formalisation

Cette sous-section présente quelques étapes pour formaliser un problème de conception de systèmes physiques complexes. Celles-ci permettent de traiter le problème de conception d'un système complexe en se servant de sa décomposition en constituants. Cette décomposition pourrait être appliquée dès lors que l'on souhaite formaliser et résoudre le problème de conception d'un système physique complexe décomposable en sous-systèmes ou encore en plusieurs contraintes fonctionnelles comme non-fonctionnelles. Pour illustrer les réflexions et développement conduits durant cette thèse, trois modèles de problèmes de conception ont été construits avec une amélioration croissante. Une extension du problème avec des exigences environnementales (non-fonctionnelles de type 2) est aussi présentée pour traiter

un problème d'éco-conception. Lors du processus de conception préliminaire d'un système physique, nous pouvons imaginer 3 cas possibles :

Cas idéal : on dispose d'un modèle formel du problème de conception et d'une spécification des exigences. Dans ce cas, il s'agit alors de réutiliser et d'adapter seulement le modèle du `Problem` de conception du système considéré à la spécification des exigences actuelle. On modifiera alors les valeurs limites imposées aux différentes variables.

Cas intermédiaire : on dispose d'une spécification des exigences mais pas de modèle de synthèse formel du problème de conception. Il faut alors construire le premier modèle de problème selon l'approche et les patterns proposée pour qu'il puisse être réutilisable. C'est le cas de formalisation que nous traiterons.

Cas général : on souhaite formaliser un problème de conception qui ne s'incrit pas dans le cadre d'un projet de conception en particulier. On désire alors construire un ensemble de bibliothèques de modèles pour les projets à venir. On peut appliquer alors partiellement l'approche proposée. La différence est qu'il faudrait au préalable étudier un ensemble de spécifications des exigences relatif au système et tenter d'identifier les exigences récurrentes d'un projet à un autre. Ce cas est plus difficile que les précédents car il est difficile de prévoir le cadre d'utilisation du système, qui doit rester assez général.

Traiter un problème de conception peut consister à décomposer celui-ci en plusieurs sous-problèmes. Un premier sous-problème est d'abord formalisé puis résolu. Si au moins une solution admissible est trouvée, le constituant suivant est intégré au problème de conception précédent puis on réitère. Les exigences non-fonctionnelles de type 2 peuvent être intégrées après la résolution de chaque sous-problème et retirées avant l'intégration du constituant suivant. Une telle décomposition aide à s'assurer que le système complet est avant tout fonctionnel et à détecter des contraintes de conception contradictoires.

Cette résolution progressive d'un problème de conception permet d'aider à la mise au point des modèles de synthèse par la correction d'erreurs de syntaxe ou d'implémentation (lié au fonctionnement du compilateur) ou par la détection de contraintes de conception contradictoires. Le coût de formalisation serait l'inconvénient principal de ces étapes et seraient à prendre en compte dans le coût de développement du système. Cet inconvénient devrait ce-

Figure 3.5 Chaîne de conversion électro-magnéto-mécanique et batterie d'un BEV.

pendant être mineur face au gains de temps qui serait acquis en réutilisant les modèles de synthèse pour d'autres projets.

Exemple : Conception d'un système de système BEV avec exigences environnementales Le Battery-powered Electric Vehicle (BEV) est constitué d'une chaîne de conversion électro-magnéto-mécanique et d'une batterie (cf. Fig. 3.5). La chaîne de conversion est composée d'un convertisseur de puissance, d'une machine électrique et d'une transmission mécanique constituée d'un réducteur de vitesse et d'un différentiel. La décomposition du problème peut alors consister à traiter le sous-problème :

1. de conception de la batterie ;
2. d'éco-conception de la batterie.

   Deux situations sont ensuite possibles. On traitera soit le sous-problème :

3. (a) de conception de la batterie et du convertisseur ou ;
   (b) de conception de la batterie et du moto-variateur, si le moto-variateur doit être selectionné à partir d'un catalogue.

   Pour chaque sous-problème, on s'orientera alors vers celui :

4. (a) d'éco-conception de la batterie et du convertisseur ou ;
   (b) d'éco-conception de la batterie et du moto-variateur, si le moto-variateur doit être selectionné à partir d'un catalogue.

   On pourra ensuite considérer le sous-problème :

5. (a) de conception de la batterie, du convertisseur et de la machine électrique ou ;
   (b) de conception de la batterie, du convertisseur et du moto-réducteur, si le moto-réducteur doit être selectionné à partir d'un catalogue ou ;
   (c) de conception de la batterie, du moto-variateur et de la transmission mécanique.

   Pour chacun de ces sous-problèmes de conception, on considérera alors le problème d'éco-conception associé.

6. (a) Pour les sous-problèmes 5b et 5c, le problème d'éco-conception de la chaîne de traction complète est alors résolu.
   (b) Pour le sous-problème 5a, il faut rajouter une étape supplémentaire avec l'intégration de la transmission mécanique puis celui de son éco-conception, afin d'obtenir le problème d'éco-conception de la chaîne de traction complète.

Exemple : Conception d'un système batterie avec exigences environnementales Un problème de conception de batterie peut être considéré uniquement du point de vue fonctionnel et selon un point de fonctionnement unique. Le sous-problème suivant prendra en compte plusieurs points de fonctionnement puis sera complété du point de vue non-fonctionnel en prenant en compte les exigences non-fonctionnelles de type 1. Pour finir, le problème prendra en compte les exigences non-fonctionnelles de type 2 environnementales.

Dans ce chapitre, nous appliquerons les étapes 1 et 2 relatives à la batterie dans le cadre de l'éco-conception d'un VE (BEV ou Plug-In Hybrid Electric Vehicle (PHEV)).

3.2.4 Recommendations générales pour la réutilisabilité

Spécificités et structurations issues de l'orienté-objet

En conception orienté-objet, la construction de modèles réutilisables doit se penser dès le départ, lors de l'analyse des exigences systèmes. Un projet de conception peut être découpé en une organisation hiérarchique de *packages* pour améliorer la réutilisabilité. Il est par exemple possible de regrouper entre elles : les grandeurs, les systèmes et les problèmes de conception. Il est également possible de rendre modulaires les différentes relations algébriques du problème en les intégrant dans des modèles. On peut ainsi réutiliser les contraintes en créant des instances de tels modèles. Un problème de conception peut être modélisé de multiples manières. L'interprétation des exigences conditionnent en partie la résolution. Plus les modèles seront réutilisables, plus la modélisation sera complexe. Il faut également considérer la quantité d'information à révéler et éviter de rendre visibles des détails d'implémentation propres au langage ou au solveur.

Figure 3.6 Modélisation d'un terminal d'interconnexion pour un constituant physique (composant électrique) (DEPS).

```
290 Model Pin()
291 Constants
292 Variables
293 expr i : INTENSITY ;
294 phi : ELECTRICPOTENTIAL ;
295 Elements
296 Properties
297 End
```

Spécificités et structurations issues de la modélisation des systèmes physiques

Le formalisme des réseaux électriques généralisés [182] De manière générale, tout système énergétique peut être représenté par deux variables comportementales et duales d'énergie : une variable *across* et une variable *through* [1]. La variable *across* se mesure entre deux points. Il peut s'agir de la différence de temperature, de la tension électrique différence de potentiels ou de la différence de pression. La variable *through* se mesure en un point. Celle-ci correspond au débit d'une grandeur physique conservative. Il s'agira par exemple de la chaleur, du courant électrique (débit de charge électrique) ou d'une force (variation de quantité de mouvement).

Les deux variables duales sont reliées par la loi constitutive du composant considéré. Le produit de deux variables duales donne généralement une énergie ou une puissance (mais ce n'est pas le cas en représentation thermique par exemple). Pour une représentation électrique, la variable *through* est l'intensité du courant électrique et représente le flux de la charge électrique à l'intérieur d'un composant. La variable *across* est la tension électrique.

Dans ce cadre, le formalisme des réseaux électriques généralisés (Networks methods) peut être employé [182]. Cette représentation énergétique des systèmes et de leurs interconnexions repose sur les deux Lois de Kirchhoff Généralisées. Ces lois duales traduisent la conservation de l'énergie. Avec cette représentation générique, il est possible de générer automatiquement les équations d'interconnexion pour les réseaux de constituants, grâce à des techniques de résolution appropriées. Cette vision du sytème est donc adaptée à la pré-conception et à la résolution par Programmation Par Contraintes (PPC).

Un terminal d'interconnexion est une entité qui représente le lieu d'une connexion physique. Celui-ci peut être modélisé dans un `Model` afin de gagner en généricité. Un exemple

1. Dans la terminologie Bond-Graph, ces variables sont appelées *flow* et *effort*, mais nous préférons utiliser la dénomination *through* et *across*, plus générales.

Figure 3.7 Modélisation d'un terminal d'interconnexion pour un constituant physique (composant mécanique) (DEPS).

```
298 Model Pin2()
299 Constants
300 Variables
301 expr c : TORQUE ;
302 omega : ANGULARSPEED ;
303 Elements
304 Properties
305 End
```

Figure 3.8 Modélisation de l'interconnexion entre deux constituants (DEPS).

```
306 Model Connect(b1, b2)
307 Constants
308 Variables
309 Elements
310 b1 : Pin ; (* component 1 terminal *)
311 b2 : Pin ; (* component 2 terminal *)
312 Properties
313 b1.phi = b2.phi ; (* compatibility of potentials *)
314 b1.i + b2.i = 0 ; (* continuity of flow *)
315 End
```

Figure 3.9 Modélisation de l'interconnexion entre trois constituants (DEPS).

```
316 Model Connect(b1, b2, b3)
317 Constants
318 Variables
319 Elements
320 b1 : Pin ; (* component 1 terminal *)
321 b2 : Pin ; (* component 2 terminal *)
322 b3: Pin ; (* component 3 terminal *)
323 Properties
324 b1.phi = b2.phi ; (* compatibility of potentials *)
325 b2.phi = b3.phi ; (* KVL *)
326 b1.i + b2.i + b3.i = 0 ; (* continuity of flow *)
327 End
```

de représentation pour un terminal d'interconnexion électrique est donné Fig. 3.6. Cette représentation objet générique de modélisation peut être généralisée aux autres systèmes physiques : hydrauliques, thermiques ou encore mécaniques. Par exemple, pour l'arbre d'entrée / sortie d'une machine électrique, on pourra modéliser un terminal d'interconnexion à travers la vitesse angulaire ω (variable *across*) et le couple c (variable *through*) (cf. Fig. 3.7). Avec cette représentation des systèmes physiques, les deux Lois de Kirchhoff Généralisées pourront être encapsulées dans un `Model` (cf. Fig. 3.8 - lignes 313-314). Des modèles similaires peuvent être construits pour interconnecter plus de constituants (cf. Fig. 3.9).

Pour étudier les enjeux de réutilisabilité des modèles trois exemples de modélisation pour un problème de conception de batterie ont été créés puis un quatrième prenant en compte des exigences environnementales. Ces modèles seront présentés dans la section suivante. Ces trois modèles de batterie ont conduit à une modélisation différente de l'exigence de configuration et par la suite à trois modèles du problème. Notamment, le premier modèle construit pour le problème ainsi que les deux suivants représentent en réalité deux problèmes de conception proches mais distincts, et aux niveaux de généricité différents. Nous proposons trois manières différentes d'implémenter la variabilité structurelle liée au choix de composants à différents niveaux de modélisation.

Les 3 niveaux de formalisation proposés Nous considérons 3 niveaux de formalisation pour la création de modèles de synthèse réutilisables : le *Problem Level*, le *Subdefinite System (SS) Level* et le *SubSystem/Components (S/C) Level*. Le *Problem Level* correspond au niveau de formalisation du `Problem`. Le *SS Level* correspond au niveau de formalisation du système sous-défini à concevoir. Le *S/C Level* correspond au niveau de formalisation d'un sous-système ou composant du système sous-défini à concevoir.

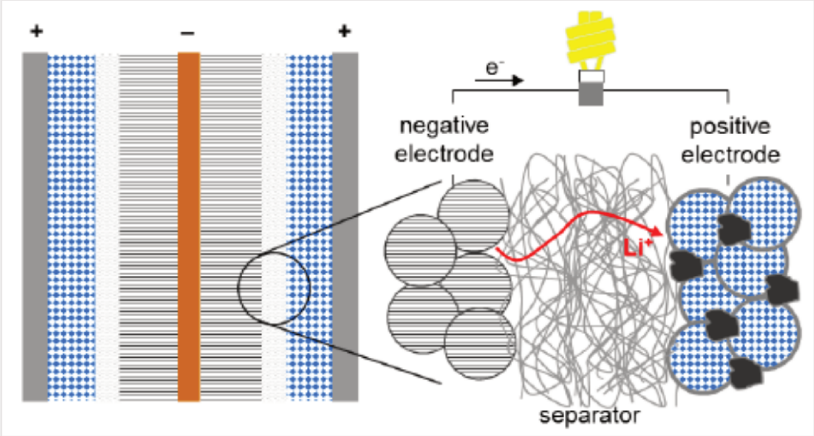
Deux sections de ce chapitre seront consacrées à la conception d'une batterie de Véhicule Electrique (VE). La Section 3.3 ne comprend que des exigences fonctionnelles et non-fonctionnelles de type 1. La Section 3.4 contiendra également des exigences non-fonctionnelles de type 2 (environnementales).

3.3 Application à la conception d'une batterie de VE

3.3.1 Introduction et spécificités du problème de conception

Même s'il n'est pas nouveau, le problème de conception d'une batterie Li-ion demeure très actuel dans le secteur automobile, de par l'attention portée sur les énergies embarquées et renouvelables. Ce cas d'étude relève à la fois des ingénieries chimique et électrique, puisqu'une batterie est un organe de stockage faisant intervenir des processus électrochimiques. Une batterie est notamment un système composé de cellules interconnectées. Chacune des cellules constitue un réservoir d'énergie. Ainsi, le cadre de l'Ingénierie des Systèmes (IS) est également pertinent.

Figure 3.10 Principaux éléments d'une cellule : cathode, séparateur, électrolyte et anode [183].



Le développement de Véhicule Electrique (VE) performants est principalement limité par celui des technologies de batteries. En effet, les performances des batteries sont limitées par leurs tailles, leurs masses et leurs coûts importants. Or la masse et le coût de la batterie déterminent principalement ceux du véhicule. Les véhicules hybrides, dôtés d'une petite batterie, sont pour cette raison plus répandus. L'énergie qu'une batterie est capable de stocker est mesurable par sa densité énergétique ou encore par son énergie massique.

La technologie Li-ion est actuellement la plus performante en termes de densité énergétique (220-400 Wh/l) et une énergie massique parmi les plus importantes (90-200 Wh/kg) [42]. De plus, son taux d'auto-décharge est parmi les plus faibles (2% par mois), et elle peut être utilisée relativement longtemps (500-1000 cycles de charge/décharge). C'est celle que l'on retrouve le plus souvent au sein des VE. Son anode est généralement en graphite et sa cathode parmi une des 4 technologies suivantes [75] : Lithium Fer Phosphate (LFP), Lithium Manganèse Oxide (LMO), Nickel Manganèse Cobalt (NMC) et Nickel Cobalt Aluminium (NCA) (cf. Fig 3.10). Le choix entre ces technologies est essentiellement un compromis entre coût, puissance, énergie, sécurité et durabilité.

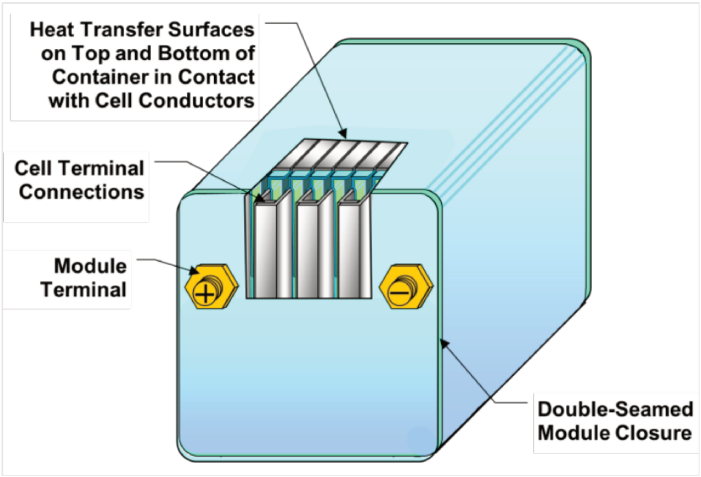
3.3.2 Choix du modèle gros grains pour le contexte de l'étude

Le modèle gros grains de la batterie permet de déterminer son état de charge et le courant à fournir au convertisseur statique. Le modèle présenté dans [22] a été repris comme benchmark pour la batterie.

Table 3.1 Catalogue de cellules disponibles.

Type	Capacité $c_{3_{cel}}$ $(A \cdot h)$	Masse (kg)	Résistance interne $(m\Omega)$	OCV (V)	
				min	max
0	39	1.05	7		
1	25	0.75	16.75	2.7	4.2
2	16	0.68	23		

Figure 3.11 Un module de batterie Li-ion [183].

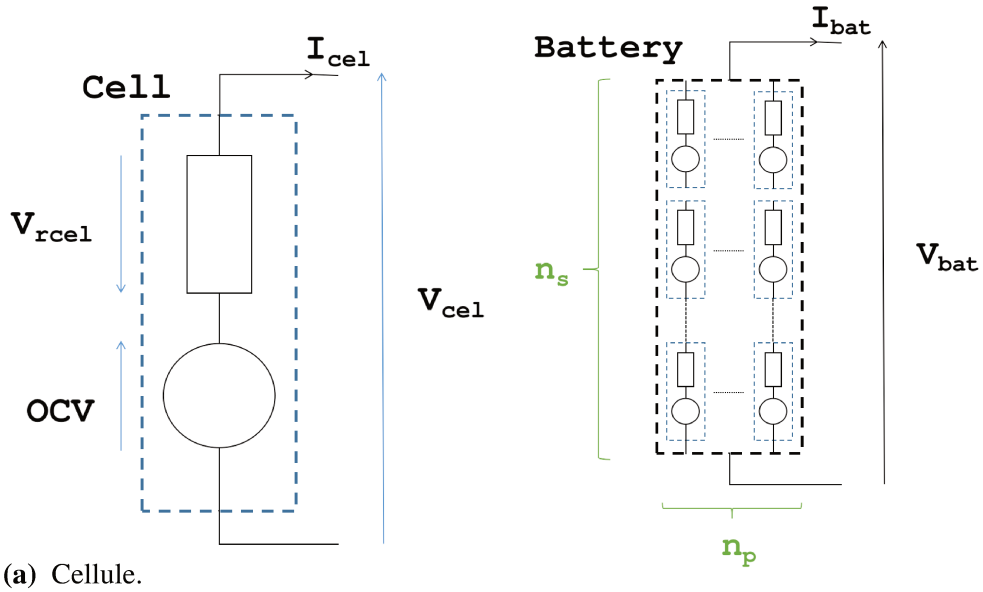


Dans le cas d'étude considéré, seul un type de cellule parmi plusieurs choix possibles peut être choisi pour une architecture admissible de batterie. Les concepteurs disposent généralement d'un catalogue listant les différents types de cellules disponibles comme celui présenté Tab. 3.1. Ce catalogue est la contrainte catalogue de ce problème. Une cellule Li-ion est caractérisée par le quadruplet {type, capacité nominale, masse, résistance interne}.

L'architecture d'une batterie est définie lorsque son type *type* et ses nombres de cellules en série n_s et en parallèle n_p sont déterminés, tel que le produit $n_s * n_p$ représente le nombre total de cellules. On retrouvera généralement entre une dizaine et un millier de cellules au sein d'une batterie de VE. Les cellules sont en réalité fabriquées et assemblées d'abord en modules, qui regroupent une interconnexion de cellules en série-parralèle, des conducteurs thermiques, des régulateurs de tension et des terminaux électriques d'interconnexions. Ces modules sont ensuite assemblés en un pack, formant la batterie (cf. Fig. 3.11).

L'objectif est ainsi de déterminer le type de cellules constitutives ainsi que leur nombre en série et parallèle, afin que la batterie puise fournir l'énergie nécessaire à la charge qu'est

Figure 3.12 Modèle électrique R_{int}.



(a) Cellule.

(b) Batterie.

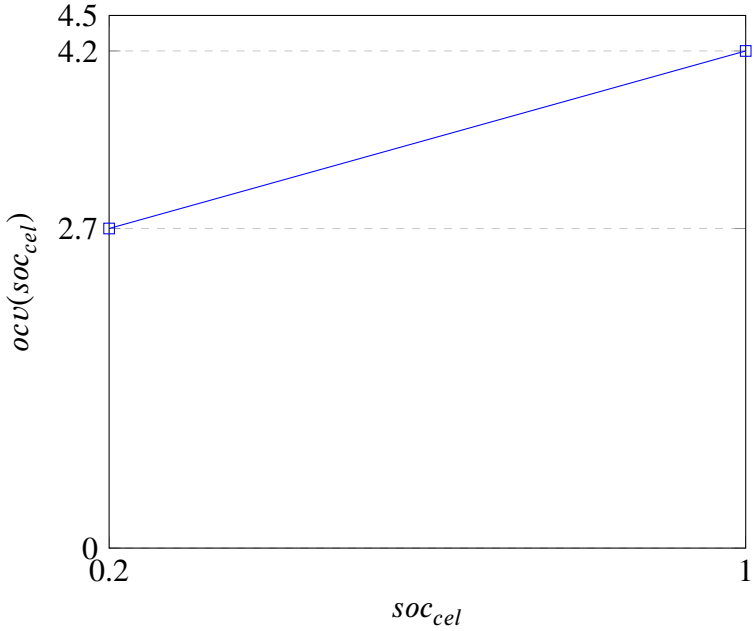
le véhicule, pour que ce dernier puisse se déplacer à la vitesse voulue. Ainsi, *type*, n_s et n_p sont les trois ddl de ce problème de conception. Le choix de ces ddl ainsi que de celui de leur nombre est généralement réalisé de manière implicite au sein de l'espace des connaissances.

Le problème de conception d'une batterie est particulièrement intéressant car il s'agit d'un problème de configuration. Celui-ci est de plus est mixte (à ddl discrets mais à variables comportementales continues). Il peut facilement être étendu en problème de génération d'architectures, en spécifiant les dimensions de chaque cellule (hauteur, largeur et épaisseur).

3.3.3 Hypothèses de modélisation

Les hypothèses suivantes sont valides pour la pré-conception [22]. Le comportement des cellules peut être considéré comme identique en ce qui concerne leur décharge/charge, ceci afin de considérer la batterie comme un ensemble homogène de cellules. Dans le cadre du GE, la batterie sera généralement représentée par un circuit électrique équivalent comme le R_{int} *model* [184] (cf. Fig 3.12a). Une cellule est modélisée comme une source de tension non parfaite constituée de l'association série d'une source de tension idéale (de tension *Open Circuit Voltage (OCV)*) et d'une résistance (de tension V_{rcel}). I_{cel} représente le courant fourni par une cellule et V_{cel} sa tension.

Figure 3.13 Evolution de l'OCV d'une cellule en fonction de son SOC.



Les cellules sont considérées identiques au niveau de la décharge. Une batterie comme présentée Fig. 3.12b peut alors être considérée comme une cellule équivalente homogène à un facteur près qui dépend de n_s et de n_p.

Une autre hypothèse choisie dans le cadre de cette étude considère la résistance interne d'une cellule comme indépendante de la température. La relation entre OCV, V_{cel} et I_{cel} est linéaire pour une cellule. En réalité, l'OCV et la résistance d'une cellule varient au cours du temps selon l'état de charge (State of Charge (SOC)), le courant de décharge et sa température. Cette simplification est possible ici car la variation de température à l'intérieur de la résistance n'est pas prise en compte. La résistance d'une cellule est donc considérée comme constante. Cette hypothèse est très forte puisque la température interne d'une cellule modifie fortement son comportement [22]. L'impact de la température est pris en compte à travers les pertes par effet joules provoquant l'échauffement de la batterie.

Le SOC est un paramètre important pour une batterie. Il quantifie le niveau d'énergie restant de la batterie durant son déplacement, et indique donc si la batterie peut fournir l'énergie demandée. La capacité et le SOC d'une batterie sont équivalents à celui d'une cellule à un instant donné. Il est exprimé en pourcentage de la capacité nominale. Le SOC dépend linéairement de l'OCV, qui dépend de la technologie de la cellule. Pour une cellule Li-ion, l'OCV varie entre 2.7 V et 4.2 V.

Ainsi, une cellule chargée possède un SOC à 1 et une OCV à 4.2 V. Une contrainte de conception sur les cellules Li-ion impose de maintenir un SOC au moins égal à 0.2 tout au long de la mission de circulation, pour éviter la destruction d'une cellule par une décharge trop profonde. La cellule est ainsi considérée comme totalement déchargée pour un SOC à 0.2 et une OCV à 2.7 V (cf. Fig. 3.13). De plus, entre 20% et 100% de la capacité nominale, la valeur de la résistance interne peut être considérée comme constante.

3.3.4 Prise en compte du profil de mission

Les choix de modélisation d'un profil de mission ne sont pas uniques. Ils peuvent être réalisés comme expliqués ci-après.

Le profil de mission correspond à la demande en énergie nécessaire pour déplacer le véhicule. La batterie doit fournir cette énergie. Le profil de de mission impose le courant de décharge que devra fournir la batterie. La connaissance du courant de décharge permet d'estimer le SOC de la batterie. Ce courant de décharge varie au cours du temps. Le profil de mission doit faire apparaître les courants ainsi que la durée de chaque décharge. Il faut donc à présent trouver un modèle gros grains permettant de le représenter.

Etat de l'art pour la description d'une mission

Pour le dimensionnement des systèmes énergétiques, le profil de mission est pris en compte de nombreuses manières en Génie Electrique (GE). Le plus simple consiste à réduire le profil de mission à un point de fonctionnement, généralement le point de fonctionnement nominal [24]. Cette modélisation est courante lorsque la charge et la vitesse d'entrainement sont constantes (ex : les trains électriques). Ce point ne suffit plus à représenter la mission lorsque le courant demandé par la charge varie fréquemment. De plus, il est difficile à déterminer à ce stade de conception [42]. D'autres points de fonctionnement peuvent être choisis comme celui le plus critique (couple et vitesse maximale) ou encore celui le plus fréquent dans le profil de mission. Cependant ces pratiques mènent généralement à un système sur-dimensionné pour le premier cas, et sous-dimensionné pour le second. D'autres points de fonctionnement que ceux mentionnés auparavant, peuvent permettre le dimensionnement de ces systèmes selon certaines contraintes de conception [185, 186].

Une autre pratique courante consiste à n'utiliser qu'une partie du profil de mission, celle la plus exigente pour l'utilisation la plus critique. Pour diminuer les temps de calculs (gé-

Figure 3.14 Conception d'un système avec un profil de mission fictif [12].

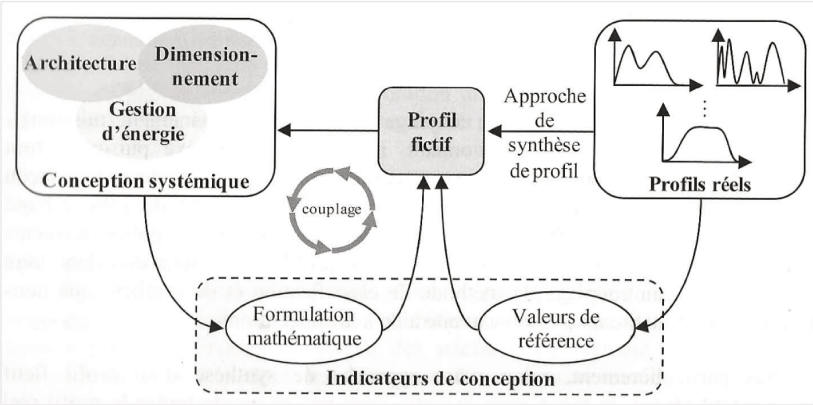
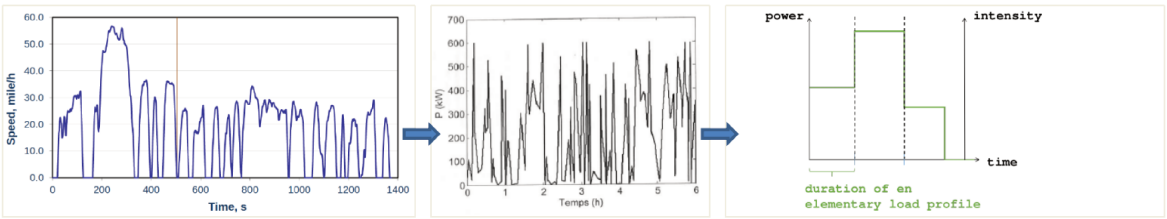


Figure 3.15 Un profil de mission (centre [42]) discrétisé en profils de mission élémentaires depuis un profil de vitesse.



Un profil de vitesse (roue véhicule) | Un profil de mission réel (chaine de traction sauf batterie) | Un profil de mission "gros grains" algébrique (quasi-statique) discrétisé en 3 profils de missions élémentaires.

néralement en simulation), ce sous-profil de mission d'une dizaine de minutes est réduit à quelques minutes [187]. Cette modélisation a l'avantage d'être simple. La difficulté consiste à nouveau à bien choisir le sous-profil de mission.

[42] propose de concevoir des systèmes avec des profils de mission fictifs qui seraient obtenus à partir de plusieurs profils réels (cf. Fig. 3.14). Bien qu'utiliser pour la conception par analyse, cette modélisation est intéressante car elle permet d'obtenir différents profils de mission en fonction des paramètres fonctionnels et non-fonctionnels à privilégier.

Hypothèses de ces travaux de thèse concernant le profil de mission

Pour le cas étudié dans ce chapitre, nous utiliserons un modèle simplifié du profil de mission. En pré-conception, les modèles à privilégier sont ceux représentant le comportement du système de manière quasi-statique, ie en régime permanent ou maximal. Un profil de mission continu et variable est discrétisé en un ensemble de profils de mission élémentaires constants. Ce profil de mission est supposé obtenu à partir d'un profil de vitesse du véhicule (cf. Fig. 3.15). Les relations algébriques sont ainsi conservées en évitant les intégrales et les

Figure 3.16 Modélisation d'un profil de mission élémentaire.

```
328 Model ELProfile(deltat, ploadk, iloadk)
329 Constants
330 ploadk : POWER ; (* required power *)
331 iloadk : INTENSITY ; (* required current *)
332 deltat : TIME ; (* duration *)
333 Variables
334 expr vloadk : VOLTAGE ; (* required voltage *)
335 Elements
336 p : Pin() ;
337 n : Pin() ;
338 Properties
339 p.i := iloadk ; (* receptor convention *)
340 n.i := - p.i ;
341 vloadk := p.phi - n.phi ;
342 vloadk = ploadk / iloadk ;
343 End
```

dépendances explicites au temps. Chacun de ces profils élémentaires est ainsi défini par un courant et une puissance constantes, ainsi qu'une durée. Ainsi, la contrainte de conception qui impose de maintenir un *SOC* au moins égal à 0.2 devra être respectée pour chaque profil de mission élémentaire.

Modélisation d'un profil de mission élémentaire

La Fig. 3.16 montre comment un profil de mission élémentaire peut être modélisé à l'aide du formalisme DEPS. Ce système est défini par une durée de fonctionnement `deltat`, la puissance requise `ploadk` et le courant demandé `iloadk`. Il est ici modélisé comme un récepteur composé de deux terminaux d'interconnexion `p` (positif) et `n` (négatif). Ceci permet notamment de renseigner la tension de la charge résultante *vloadk* en la définissant comme une différence de potentiels (ligne 341). Une convention récepteur est également définie telle que le courant traversant un système entre par son terminal positif `p` (ligne 339). Le courant et la tension sont alors orientés en sens inverse.

3.3.5 Exemple d'une spécification des exigences

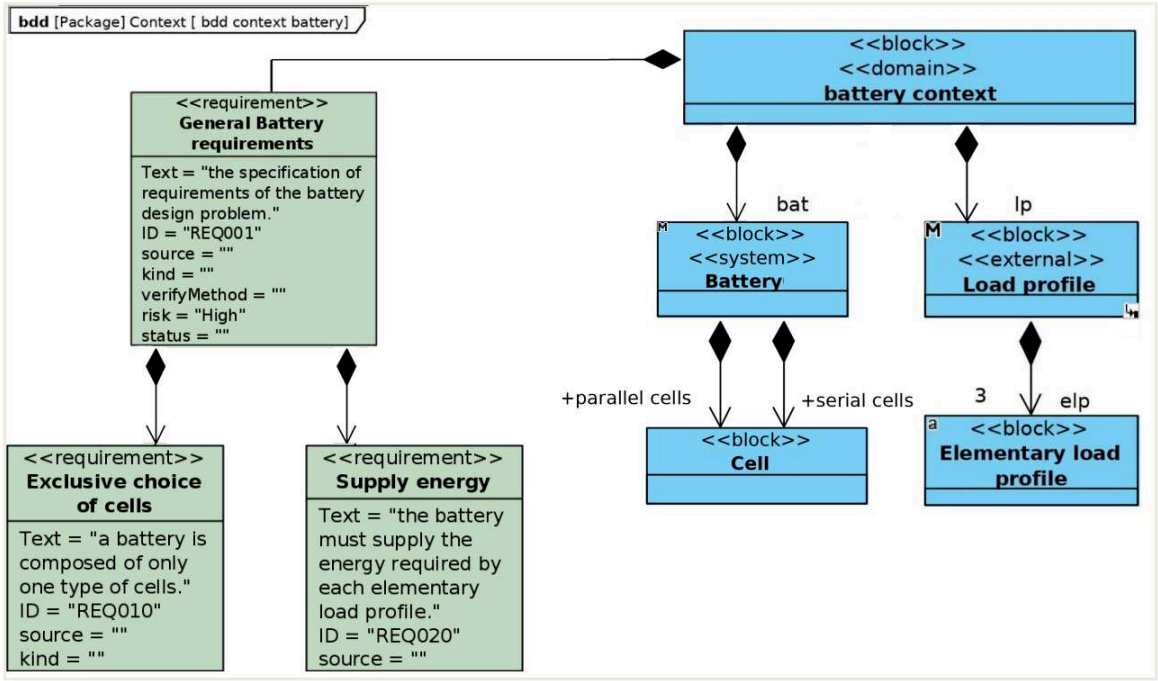
Un exemple de spécification des exigences est définie par les 12 exigences suivantes [181] :

E1 : la batterie doit fournir l'énergie demandée par la mission de circulation (charge principale).

E2 : les pertes par effet joules de la batterie constituent une charge supplémentaire à compenser.

E3 : le SOC de la batterie ne doit jamais être inférieur à 20%.

Figure 3.17 Composantes du modèle de problème de conception de la batterie (SysML).



E4 : le profil de mission est le suivant : la durée totale de la mission est de 30 min. Trois profils élémentaires sont définis : $10831W@22.8A$, $45210W@102A$ et $740W@50A$, chacun pendant $600s$.

E5 : trois types de cellules sont disponibles parmis un catalogue : 0, 1, 2. Chacune est caractérisée par un type, une capacité nominale, une valeur de résistance interne et une masse (cf. Tab. 3.1).

E6 : une batterie est composée d'un unique type de cellule.

E7 : le nombre de cellules en série ne doit pas excéder 5000.

E8 : le nombre de cellules en parallèle ne doit pas excéder 5000.

E9 : la masse de la batterie ne doit pas excéder 800 kg.

E10 : le courant maximal de décharge dans une cellule est limité à 40 A pour protéger les cellules d'une décharge trop profonde.

E11 : l'énergie utile de la batterie ne doit pas excéder 100 kWh.

E12 : le rendement de la batterie doit être d'au moins 90%.

Les éléments du problème de conception sont résumés Fig. 3.17. Pour ce cas de conception, le *System Level* correspondra à un modèle de la batterie et le *S/C Level* à celui d'une cellule. Le *Problem Level* correspondra toujours à un `Problem`. La règle de configuration du

système pour le problème de conception de la batterie est celle du choix exclusif de composants sélectionnés à partir d'un catalogue.

Pour la première version du problème de conception, cette règle a été modélisée au niveau du *Problem Level*. Pour la seconde, elle a été modélisée au niveau du *System Level*. Ces deux versions ont été réalisées en parallèle du développement de DEPS et DEPS Studio (version 2018) qui n'incluaientt pas encore la prise en compte de contraintes catalogues directement. Il a donc fallu trouver d'autres mécanismes pour exprimer le choix exclusif de composants. Enfin, pour la troisième version, les tables et contraintes catalogues pouvaient être directement spécifiés dans DEPS Studio (version 2019). La règle de configuration du système a alors été modélisée au niveau du *Problem Level*. Les trois versions de modélisation seront d'abord présentées de manière séparée puis leurs résolutions de manière regroupée.

3.3.6 1er exemple de modélisation (*Problem Level*)

La formalisation au niveau du *Problem Level* déporte le choix de composants (cellules) à celui du choix de systèmes (batteries).

Modèle d'une batterie sous-définie

Un `Model Battery` pour une batterie sous-définie est spécifié en formalisme DEPS comme sur la Fig. 3.18. Une instance de `Battery` prendra comme argument une instance `cell`. Deux des variables de conception `ns` et `np` ainsi que des expressions nommées (ex : `ibat`) y sont déclarées. En plus, une variable booléenne `bat` est déclarée pour spécifier le choix (`bat` = 1) ou le rejet (`bat` = 0) d'une cellule. L'élément `cell` est une instance d'un `Model LiIonCell` dont la signature est spécifiée entre crochets (ligne 360). Les signatures de `Models` seront remplacées par *[...]* pour des raisons de lisibilité. Le `Model LiIonCell` représentant une cellule sous-définie est partiellement présentée Fig. 3.19.

Le concept orienté-objet d'imbrication (*nesting)* permet de définir les expressions nommées de la batterie en accédant aux Variables de `cell`. Cette imbrication n'est par ailleurs visible que pour un certain ensemble d'expressions nommées (comme pour `ibat` ou `rbat`) et se masque dès lors que cet ensemble est défini (comme pour `copperloss`). Il est également possible de déclarer et de définir des expressions nommées faisant référence à ces variables imbriquées pour éviter de visuellement surcharger les propriétés. La variable déduite boo-

Figure 3.18 Modélisation d'une batterie sous-définie (*Problem Level*)(DEPS).

```
344 Model Battery(cell)
345 Constants
346 Variables
347 ns : INTEGER in [1,10000] ;(* number of serial cells *)
348 np : INTEGER in [1,10000] ;(* number of parallel cells *)
349 expr ibat : INTENSITY ;(* current *)
350 expr ocvbat : VOLTAGE ;(* ocv *)
351 expr vbat : VOLTAGE ;(* terminal voltage *)
352 expr rbat : RESISTANCE ;(* internal resistance *)
353 expr mbat : MASS ;(* battery mass *)
354 expr avenergy : ENERGY ; (* available energy *)
355 expr bat : BOOLEAN ;
356 (...)
357 Elements
358 n : Pin() ;
359 p : Pin() ;
360 cell : LiIonCell[CELLTYPE, RESISTANCE, CHARGECAPACITY, MASS] ;
361 Properties
362 ibat := np * cell.icel ;
363 vbat = ns * cell.vcel ;
364 rbat := ns/np * cell.rcel.r ;
365 copperloss := rbat * (ibat^2) ;
366 powerbat := ibat * vbat ;
367 bat := cell.cgt ;
368 (...)
369 End
```

Figure 3.19 Modélisation d'une cellule sous-définie (*Problem Level*)(DEPS).

```
370 Model LiIonCell(cellType, r0, c3, mcel)
371 Constants
372 peukert : REAL = 1.1 ; (* Peukert coefficient *)
373 cellType : CELLTYPE ; (* type of cell *)
374 c3 : CHARGECAPACITY ; (* nominal capacity *)
375 mcel : MASS ; (* mass *)
376 r0 : RESISTANCE ; (* internal resistance *)
377 (..)
```

léenne `bat` définie ligne 367 est ainsi en réalité une *copie* d'une variable semblable (`cgt`) déclarée dans `LiIonCell`.

Modélisation de la demande d'énergie

Dans un premier temps, afin de modéliser les exigences E1 et E3, une batterie peut être interconnectée séparément à chaque profil élémentaire plutôt qu'au profil de mission comme dans le `Model Todo` (Fig. 3.20 - ligne 387). La demande en puissance est définie en termes de courant et de tension à fournir (resp. lignes 389 et 390). Les pertes par effet joules sont implicitement prises en compte à travers l'expression de la tension de la batterie (`vbat =`

Figure 3.20 Modélisation de la demande de puissance (*Problem Level*)(DEPS).

```
378 Model Todo(p, b)
379 Constants
380 Variables
381 socini : REAL in [0,1] ;
382 expr socfin : REAL ;
383 Elements
384 p : ELProfile[...] ;
385 b : Battery[...] ;
386 g : Ground() ;
387 c1 : Connect(p, b, g) ;
388 Properties
389 p.iloadk <= b.ibat ;
390 p.vloadk <= b.vbat ;
391 socfin := socini - b.cell.icel/b.cell.c3 * p.deltat ;
392 socfin >= 0.2 ;
393 End
```

Figure 3.21 Interconnexion entre batterie et profil de la charge (*Problem Level*)(DEPS).

```
394 Model Supply(p1, p2, p3, b)
395 Constants
396 Variables
397 expr socini : REAL ;
398 expr socfin : REAL ;
399 expr energy : ENERGY ;
400 expr s : BOOLEAN ;
401 Elements
402 p1 : ELProfile[...] ;
403 p2 : ELProfile[...] ;
404 p3 : ELProfile[...] ;
405 b : Battery[...] ;
406 tda : Todo(p1, b) ;
407 tdb : Todo(p2, b) ;
408 tdc : Todo(p3, b) ;
409 Properties
410 socini = 1 ;
411 tda.socini = socini ;
412 tdb.socini = tda.socfin ;
413 tdc.socini = tdb.socfin ;
414 energy := (p1.ploadk * p1.deltat + p2.ploadk * p2.deltat + p3.ploadk
       * p3.deltat) ;
415 b.energy >= energy ;
416 End
```

`ocvbat - rbat * ibat`). Dans un second temps, les profils de missions élémentaires peuvent être interconnectés pour former le profil de mission comme dans le `Model Supply` (Fig. 3.21 - lignes 411-413). Le SOC initial d'une batterie à concevoir est connu et vaut 1 (ligne 410). La demande en énergie est aussi spécifiée dans ce `Model` (lignes 414-415).

Figure 3.22 Modélisation de la contrainte d'exclusion (*Problem Level*)(DEPS).

```
417 Model Exclusion(s1, s2, s3)
418 Constants
419 Variables
420 Elements
421 s1 : Supply[...] ;
422 s2 : Supply[...] ;
423 s3 : Supply[...] ;
424 Properties
425 s1.s + s2.s + s3.s = 1 ;
426 End
```

Modélisation de l'exigence relative au choix de batterie

L'exigence E6 correspond à la règle d'exclusion mutuelle [48]. Cette règle est équivalente à une contrainte logique de type *ou exclusif*. Une solution admissible de batterie ne peut comprendre qu'un type de cellules à la fois (0 ou 1 ou 2). La Fig. 3.22 montre un `Model Exclusion` contenant cette contrainte logique (ligne 425). Chaque instance du `Model` *Supply* fait implicitement référence à un types de cellule. La contrainte porte alors sur les trois variables booléennes `s`, chacune représentant un type de batterie.

Modélisation du problème

La Figure 3.23 montre le problème de conception modélisé comme un choix de batteries. Le modèle du problème fait partie du package `BatteryPackage`. Les packages sont spécifiés par le mot-clé `Package`. Le package `BatteryPackage` utilise les packages `Types` et `Components` (lignes 427-428) via la clause `Uses`. Le `Problem` `BatteryPb1Cell` est créé lignes 430-458.

Des constantes sont ensuite déclarées et définies pour faciliter l'adaptation du modèle pour d'autres projets. Leurs valeurs limitent les Variables structurelles (`nsmax`, `npmax`, `mbatmax`) et comportementales (`icelmax`, `energymax`, `efficiencymin`) de la batterie (lignes 431-437) conformément aux exigences E7-E12.

La section `Elements` comprend les trois instances de cellules, de batteries (lignes 440-445), et de profils de mission élémentaires (exigence E4 - lignes 446-448). Les lignes suivantes traduisent l'interconnexion entre chaque batterie et le profil de la charge relative aux exigences E1-E3 (lignes 449-451). Le choix de batterie est modélisé à travers une instance du modèle d'exclusion (exigence E6).

Figure 3.23 Modélisation du problème de conception (*Problem Level*)(DEPS).

```
427 Package BatteryPackage ;
428 Uses Types, Components ;
429 (* battery problem model - choice of batteries *)
430 Problem BatteryPb1Cell
431 Constants
432 nsmax : INTEGER = 5000 ;
433 npmax : INTEGER = 5000 ;
434 mbatmax : MASS = 800 ;
435 icelmax : INTENSITY = 40 ;
436 energymax : ENERGY = 100000 ;
437 efficiencymin : REAL = 0.9 ;
438 Variables
439 Elements
440 cell0 : LiIonCell(0, 0.007, 39, 1.05) ;
441 cell1 : LiIonCell(1, 0.01675, 25, 0.89) ;
442 cell2 : LiIonCell(2, 0.023, 16, 0.68) ;
443 b0 : Battery(cell0) ;
444 b1 : Battery(cell1) ;
445 b2 : Battery(cell2) ;
446 elp1 : ELProfile(600, 10831, 22.8) ;
447 elp2 : ELProfile(600, 45210, 102) ;
448 elp3 : ELProfile(600, 740, 50) ;
449 supply0 : Supply(elp1, elp2, elp3, b0) ;
450 supply1 : Supply(elp1, elp2, elp3, b1) ;
451 supply2 : Supply(elp1, elp2, elp3, b2) ;
452 e : Exclusion(supply0, supply1, supply2) ;
453 max1 : MaxValues(b0, nsmax, npmax, mbatmax, icelmax, energymax);
454 max2 : MaxValues(b1, nsmax, npmax, mbatmax, icelmax, energymax);
455 max3 : MaxValues(b2, nsmax, npmax, mbatmax, icelmax, energymax);
456 min1 : MinValues(battery, efficiencymin) ;
457 Properties
458 End
```

Les contraintes de conception relatives aux limitations et exprimées par les exigences E7-E12 sont spécifiées (lignes 453-455) comme instances d'un `Model` présenté Fig. 3.24. L'exigence E12 est modélisée dans un `Model MinValues` similaire.

3.3.7 2ème exemple de modélisation (*System Level*)

Dans ce deuxième exemple, le problème est directement modélisé comme un choix entre trois types de cellules au niveau du *System Level* et non de batteries. Cet exemple montre une modélisation du problème de conception avec un plus haut niveau de réutilisabilité et de compacité.

La différence majeure entre cette modélisation et la précédente repose sur le modèle du système. Le principe de modélisation est identique à l'exemple 1 à l'exception que l'exi-

Figure 3.24 Modélisation des performances E7-E11 (*Problem Level*)(DEPS).

```
459 Model MaxValues(b, nsmx, npmx, mbatmx, icelmx, energymx)
460 Constants
461 nsmx : INTEGER ;
462 npmx : INTEGER ;
463 mbatmx : MASS ;
464 icelmx : INTENSITY ;
465 energymx : ENERGY ;
466 Variables
467 Elements
468 b : Battery[...] ;
469 Properties
470 b.ns <= nsmx ;
471 b.np <= npmx ;
472 b.mbat <= mbatmx ;
473 b.cell.icel <= icelmx ;
474 b.avenergy <= energymx ;
475 End
```

gence E6 relative au choix de composants est modélisée directement dans le `Model Battery` comme présenté Fig. 3.25. La section `Elements` contient ainsi trois instances du `Model LiIonCell` ainsi qu'une instance d'un `Model Exclusion` différent de celui présenté dans le premier exemple (Fig. 3.22) mais reposant sur le même principe.

Une autre différence qui en résulte est que chaque propriété de `Battery` doit être pondérée par la variable `cgt` de chaque instance de `LiIonCell` (sinon les solutions obtenues pendant la résolution seront fausses). Le nombre de ces pondérations augmente avec celui des cellules à considérer dans le problème. Ces pondérations sont en revanche rapidement masquées en définissant des variables déduites. Par ailleurs, les seules propriétés à pondérer sont celles du `Model Battery` du système.

Dans l'exemple 1, les propriétés sont aussi pondérées mais une absence de pondération ne fausse pas les résultats obtenus. Cependant, chaque modèle utilisant une instance de `Battery` doit avoir des propriétés pondérées, ce qui se révèle en réalité plus fastidieux que dans l'exemple 2. Les pondérations ont été retirées de l'exemple 1 principalement par soucis de clarté. Cette modélisation du choix de cellules permet de supprimer des lignes et de les alléger par rapport à la modélisation au *Problem Level* (1er exemple). aussi bien pour les `Models` que pour le `Problem`, comme le montre le `Problem BatteryPb3Cells` (Fig. 3.26). `BatteryPb3Cells` est plus générique que le `Problem BatteryPb1Cell`. Il est aussi plus robuste car il dépend moins de l'espace des solutions (les caractéristiques propres aux cellules) et est plus facilement modifiable.

Figure 3.25 Modélisation d'une batterie sous-définie (System *Level* (DEPS)).

```
476 Model Battery(cell0, cell1, cell2)
477 Constants
478 Variables
479 ns : INTEGER in [1,10000] ; (* number of serial cells *)
480 np : INTEGER in [1,10000] ; (*number of parallel cells*)
481 expr ibat : INTENSITY ; (* current supplied to the load *)
482 expr ocvbat : VOLTAGE ; (* open circuit voltage *)
483 expr vbat : VOLTAGE ; (* terminal voltage *)
484 expr rbat : RESISTANCE ; (* internal resistance *)
485 expr mbat : MASS ; (* mass *)
486 expr avenergy : ENERGY ; (* available energy *)
487 (...)
488 Elements
489 n : Pin() ;
490 p : Pin() ;
491 cell0 : LiIonCell[...] ;
492 cell1 : LiIonCell[...] ;
493 cell2 : LiIonCell[...] ;
494 exclude : Exclusion(cell0, cell1, cell2) ;
495 Properties
496 ibat := np * (cell0.icel * cell0.cgt + cell1.icel * cell1.cgt + cell2
       .icel * cell2.cgt) ;
497 vbat = ns * (cell0.vcel * cell0.cgt + cell1.vcel * cell1.cgt + cell2.
       vcel * cell2.cgt) ;
498 rbat := (ns/np) * (cell0.rcel.r * cell0.cgt + cell1.rcel.r * cell1.
       cgt + cell2.rcel.r * cell2.cgt) ;
499 copperloss := rbat * (ibat^2) ;
500 powerbat := ibat * vbat ;
501 (...)
502 End
```

3.3.8 3ème exemple de modélisation (*S/C Level*)

La différence majeure entre ce 3ème exemple de modélisation et les deux précédents est la spécification directe de la contrainte catalogue dans le modèle du SubSystem/Components (S/C). Cet exemple a pu être mis en œuvre car nous disposions alors de la version 2019 de DEPS Studio, ce qui n'était pas le cas lorsque nous avons traité les 2 exemples précédents.

Le catalogue de composants disponibles s'implémente dans un modèle DEPS appelé `Table` dans lequel sont définis les tuplets de valeurs admissibles définissant les cellules (Fig. 3.27). Il est possible de spécifier autant de tuplets que nécessaire et de manière indépendante des `Models` ou du `Problem`, ce qui limite les modifications à apporter à ces derniers.

La contrainte catalogue est modélisée au *S/C Level* dans le modèle `LiIonCell` : le sous-système/composant cellule devient alors sous-défini (ligne 575). Egalement, une unique instance du composant doit être déclarée et créé dans le modèle du système (ligne 553) pour

Figure 3.26 3ème exemple de formalisation du problème (*S/C Level*)(DEPS).

```
503 Package BatteryPackage ;
504 Uses Types, Components ;
505 Problem BatteryPb3Cells
506 Constants
507 nsmax : INTEGER = 5000 ;
508 npmax : INTEGER = 5000 ;
509 mbatmax : MASS = 800 ;
510 icelmax : INTENSITY = 40 ;
511 energymax : ENERGY = 100000 ;
512 efficiencymin : REAL = 0.9 ;
513 Variables
514 Elements
515 cell0 : LiIonCell(0, 0.007, 39, 1.05) ;
516 cell1 : LiIonCell(1, 0.01675, 25, 0.75) ;
517 cell2 : LiIonCell(2, 0.023, 16, 0.68) ;
518 battery : Battery(cell0, cell1, cell2) ;
519 elp1 : PortionLoad(600, 10831, 22.8) ;
520 elp2 : PortionLoad(600, 45210, 102) ;
521 elp3 : PortionLoad(600, 740, 50) ;
522 supply : Supply(elp1, elp2, elp3, battery) ;
523 max1 : MaxValues(battery, nsmax, npmax, mbatmax, icelmax, energymax)
      ;
524 min1 : MinValues(battery, efficiencymin) ;
525 Properties
526 End
```

Figure 3.27 Modélisation du catalogue de cellules (*S/C Level*)(DEPS).

```
527 Table cellTable
528 Attributes
529 type : INDEX ;
530 resistance : RESISTANCE ;
531 capacity : CHARGECAPACITY ;
532 mass : MASS ;
533 Tuples
534 [0, 0.007, 39, 1.05], (* cell 0 *)
535 [1, 0.01675, 25, 0.75], (* cell 1 *)
536 [2, 0.023, 16, 0.68], (* cell 2 *)
537 End
```

implicitement spécifier qu'un seul type de composant sera choisi.

Il n'est plus nécessaire de définir de variables booléennes intermédiaires, d'exclure explicitement des instances ni de pondérer les contraintes. Le gain en termes de coûts de modélisation (en temps et en efforts) ainsi qu'en termes d'expressivité et de généricité est conséquent comparé aux deux exemples de modélisation précédents.

Figure 3.28 Modélisation d'une batterie sous-définie (*S/C Level*)(DEPS)

```
Model Battery()
Constants
Variables
ns : INTEGER in [1,10000] ;(* number of serial cells *)
np : INTEGER in [1,10000] ;(* number of parallel cells *)
expr ibat : INTENSITY ;(* current *)
expr ocvbat : VOLTAGE ;(* ocv *)
expr vbat : VOLTAGE ;(* terminal voltage *)
expr rbat : RESISTANCE ;(* internal resistance *)
expr mbat : MASS ;(* battery mass *)
expr energy : ENERGY ; (* energy *)
(...)
Elements
n : Pin() ;
p : Pin() ;
cell : LiIonCell() ;
Properties
(...)
End
```

Figure 3.29 Modélisation d'une cellule sous-définie (*S/C Level*)(DEPS).

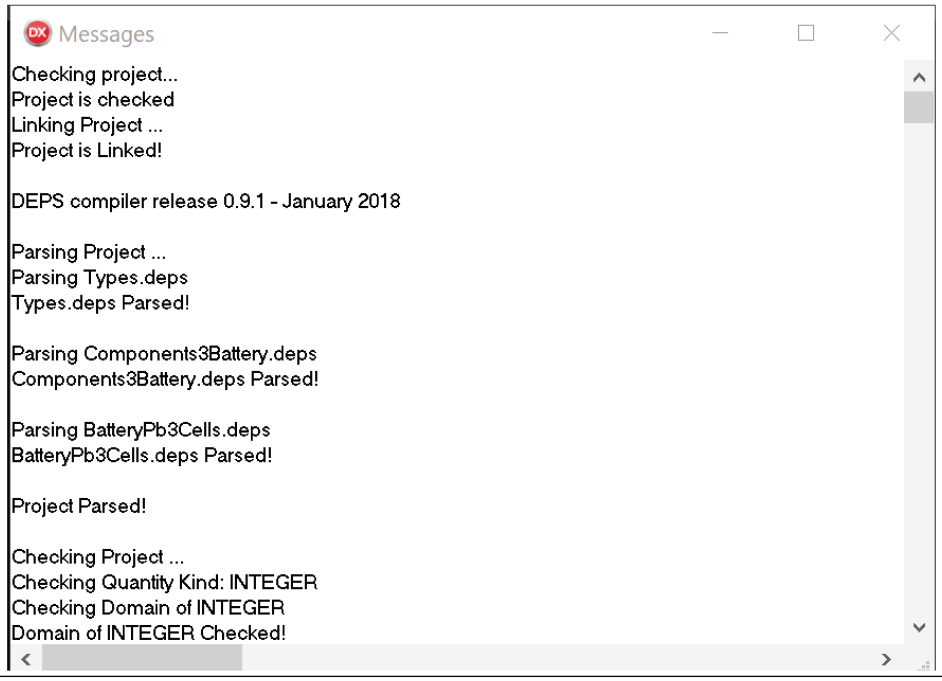
```
Model LiIonCell()
Constants
peukert : REAL = 1.1 ; (* Peukert coefficient*)
Variables
cellType : INDEX ; (* type of cell *)
r0 : RESISTANCE ;  (* static resistance *)
c3 : CHARGECAPACITY ; (* nominal capacity *)
mcel : MASS ; (* mass *)
icel : INTENSITY ; (* current through a cell *)
expr vcel : VOLTAGE ; (* cell voltage *)
(...)
Elements
e0 : VoltageSource(2.7, 4.2) ; (* ideal DC voltage source *)
rcel : Resistor() ; (* resistor *)
n : Pin() ; (* negative pin *)
p : Pin() ; (* positive pin *)
connectER : Connect(e0.p, rcel.p) ;
Properties
Catalog([cellType, r0, c3, mcel], cellTable) ;
(...)
End
```

3.3.9 Vérification et résolution des 3 exemples de modélisation

Construction du problème et vérification de la consistence des modèles

L'environnement DEPS Studio permet en plus de l'édition des `Packages` (du `Problem` et des `Models`, chacun sauvegardé sous forme de fichier) de les organiser en projets, de les compiler puis de les résoudre. Le compilateur permet d'une part de vérifier un par un la

Figure 3.30 Projet vérifié et génération du problème (DEPS Studio).



consistence des modèles (syntaxe, lexique, dépendances entre modèles) et d'autre part de construire le problème [137]. La mise au point des modèles ne dépend pas du langage du solveur cible utilisé pour la résolution. Celle-ci est réalisée directement en DEPS sans avoir à transformer le modèle, ce qui permet une plus grande rapidité d'execution. La remontée des erreurs a lieu à cette étape, avant la résolution (cf. Fig 3.30). Après la compilation, la résolution peut être envisagée.

Des modèles DEPS vers le modèle de calcul et sa résolution

Les modèles de synthèse DEPS ont été compilés séparément pour les 3 exemples de modélisation expliqués précédemment dans DEPS Studio. Trois modèles de calcul ont été produits en vue d'une résolution. Ces modèles de calcul internes et génériques sont de type CSP et conservent la structure des modèles.

Un modèle de calcul peut alors être résolu soit avec le solveur intégré à DEPS Studio, soit par un solveur externe (CE). L'utilisation du solveur de DEPS Studio permet la résolution dans un seul et même environnement. Pour résoudre le problème avec CE, un menu dans DEPS Studio permet de générer automatiquement un modèle CE et de l'ouvrir dans CE.

Table 3.2 Une architecture admissible.

Variables	**Valeurs**	**Unités**	**Exigences**
`type`	0	/	E6
`ns`	118	/	E7
`np`	5	/	E8
`mbat`	619.5	/	E9
`icel`	20.4	A	E10
`ibat`	102	A	E1
`vbat`	478	V	E1
`powerbat`	48831	W	E1, E2
`energybat`	93354	$W \cdot h$	E11
`soc`	73	%	E3
`efficiencybat`	96	%	E12

Le problème de conception a ainsi été résolu pour les 3 modélisations à l'aide des deux solveurs. Les 3 manières de formaliser le problème conduisent bien à des solutions identiques. Le résultat de la résolution est la synthèse d'un ensemble d'architectures admissibles par construction. Parmis toutes les solutions obtenues, une architecture admissible est présentée Tab. 3.2 à titre d'illustration. Pour cette solution, l'architecure de la batterie est définie par 5 cellules en parallèle, 118 en série, de type 0. Toutes les contraintes de conception sont bien respectées. Ces travaux ont fait l'objet d'une communication pour le congrès national SGE 2020 [181] ainsi que d'une publication pour la revue Computers In Industry (CII) [67].

3.4 Extension aux exigences environementales

Dans cette section, nous proposons d'intégrer des exigences non-fonctionnelles de type 2 au problème de conception initial de la batterie Li-ion afin de traiter un problème d'éco-conception. L'objectif est de concevoir la batterie en réutilisant et en étendant les modèles de synthèse construits précédemment à partir de modèles gros grains algébriques couplés. Ce problème est inspiré de l'article [76] dédié à la conception par analyse d'une batterie, dont nous nous servons comme benchmark pour la conception par synthèse.

3.4.1 Problématique d'éco-conception

L'éco-conception d'un VE consiste à le concevoir en limitant son impact environnemental durant son cycle de vie. Le cycle de vie d'une batterie Li-ion pour VE comprend [188, 189, 183] :

- — l'extraction et le transport des matières premières pour sa fabrication ;
- — sa production ;
- — son utilisation ;
- — sa fin de vie.

Il est également important de considérer en plus du mode de production de l'énergie électrique pour la recharge de la batterie [190, 75, 1].

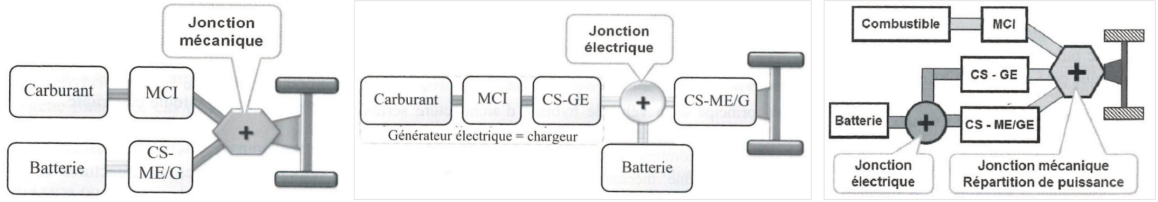
Les différentes catégories de VE

Les termes généraux de VE regroupent en réalité plusieurs catégories de véhicules partiellement ou totalement électrifiés. Il existe les VE à traction purement électrique (BEV), à pile à combustibles ou encore électriques hybrides (HEV, PHEV), ces derniers se déclinant de nombreuses façons, à travers différentes architectures.

Les VE à traction par batterie BEV Une architecture de BEV a été présentée Fig. 3.5. La batterie sera rechargée sur une borne de recharge ou pendant la phase de freinage. Dans le premier cas, il faut prendre en compte les émissions de polluants qui ont lieu durant la production de l'énergie électrique nécessaire à la recharge. Dans le second cas, ces véhicules n'émettent pas de polluants lors de leurs usages.

Les VE hybrides Le VE hybride allie un système de traction via un moteur à combustion interne (utilisant du gazole ou de l'essence), et une machine électrique alimentée par une batterie. Il présente différents niveaux d'hybridation et embarque différentes quantités d'énergie On peut séparer le VE hybride en deux catégories : les Hybrid Electric Vehicle (HEV), à faible et moyenne hybridation et les Plug-In Hybrid Electric Vehicle (PHEV) à pleine hybridation [42]. À cela s'ajoute également différentes architectures possibles pour l'hybridation comme montré Fig. 3.31. Ces véhicules émettent d'autant plus de polluants pendant leur usage que leur hybridation est faible.

Figure 3.31 VE hybride : parallèle (gauche), série (milieu), parallèle-série (droite) [42].



HEV Les HEV reposent principalement sur l'utilisation du moteur à combustion interne. Il se déclinent en micro- et moyen-hybrides. Les micro-hybrides ne sont pas particulièrement intéressants dans le cadre de l'électrification des véhicules car la batterie n'est utilisée que pour aider au démarrage en côte. En effet, la part d'usage du véhicule en électrique ne représente qu'au plus 5% de l'énergie totale consommée (taux d'hybridation). Les moyen-hybrides sont plus intéressants car ils présentent un taux d'hybridation entre 5% et 20%, en permettant une petite part de recharge des batteries par freinage récupératif [12].

PHEV Les PHEV, qui ont un taux d'hybridation supérieur à 20%, sont les plus intéressants des trois catégories de VE hybrides. Ces véhicules dits à pleine hybridation permettent un usage réellement hybride du VE. Le véhicule peut également rouler purement en électrique car les batteries sont rechargeables via l'énergie issue de la combustion interne, via l'énergie issue du freinage récupératif ou du réseau de distribution électrique.

Problématiques environnementales

On retrouve les effets des activités humaines sur l'environnement classés en trois catégories : l'impact sur les ressources naturelles (A), sur la biodiversité (B) et sur la santé humaine (C) [191, 189, 76, 190, 1]. Ces effets sont évalués par des indicateurs d'impact environnemental.

En fonction du polluant, on considèrera des impacts au niveau local (en zone urbaines) ou global (à l'échelle de la planète) [191]. Dans cet exemple, les modèles de synthèse sont étendus par la prise en compte d'impacts par des émissions de Carbon Dioxide (CO_2) au niveau global et de Nitrogen Oxides (NO_x), Carbon Monoxide (CO) et Hydrocarbons (HC) au niveau local.

L'indicateur Global Warming Potential (GWP) de catégorie B évalue l'émission de gaz à effets de serres [191, 192, 188]. Il traduit l'impact sur le réchauffement climatique qui occure à travers l'émission de polluants comme le CO_2 ou le SO_2. Le GWP est l'indicateur

le plus pertinent si on s'interesse aux impacts d'une batterie sur l'environnement, qui interviennent principalement à près de 82% pendant son usage, à env. 11% lors de l'extraction des matières premières et à près de 5% lors de sa fabrication pour un BEV [190, 189, 1]. Le GWP s'exprime en kg équivalent CO_2.

3.4.2 Choix de modélisation

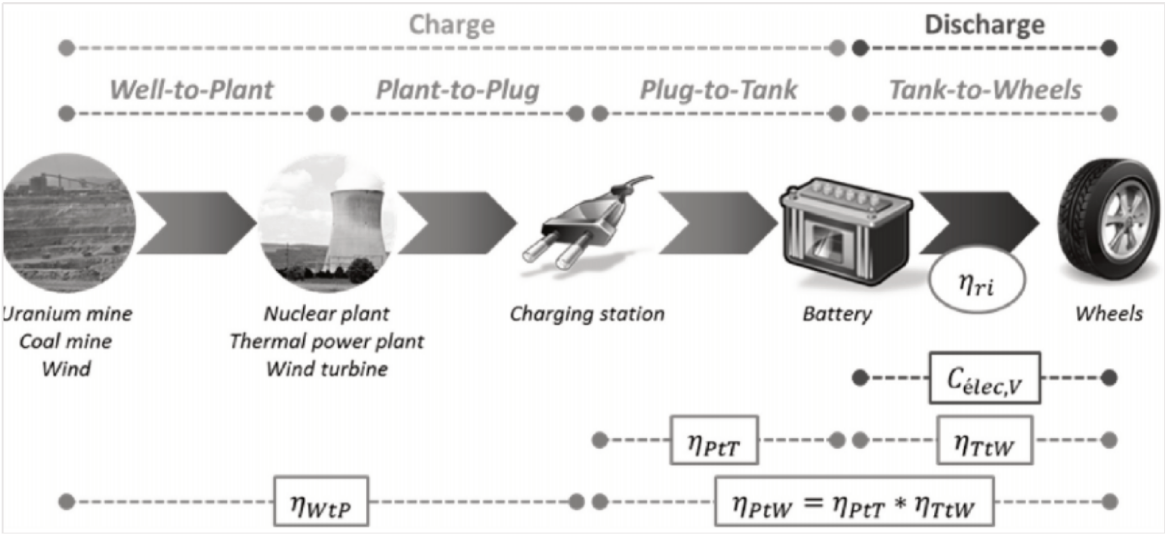
L'exemple proposé dans cette section reprend le modèle gros grains algébrique et le profil de référence environnemental présentés dans le benchmark [76]. De plus, nous reprenons des éléments issus de la section présente pour illustrer la construction de modèles de synthèse par extension d'un problème existant. Ce modèle s'applique aux batteries pour BEV et PHEV. Ce modèle gros grains peut être supplémenté à celui présenté en Annexe C [22] pour la batterie car dans ce cas seule l'énergie demandée importe et aucun modèle ne repose sur la description d'une architecture de VE en particulier (BEV ou HEV). Ce modèle gros grains peut également être utilisé car le couplage avec deux des ddl (`ns` et `np`) peut être réalisé avec la masse de la batterie, son energie utile et son rendement.

Les hypothèses sont décrites ci-après. La part de la batterie sur le GWP est de 30% pour les consommations d'énergies. Les impact locaux considérés ici incluent les émissions de CO, NO_x et de HC. Pour le GWP, nous ne considérons que la masse de CO_2 émis dû au transport de la batterie lors de l'usage du véhicule. Seul un type de carburant sera considéré à la fois (gazole ou essence). L'énergie électrique est principalement issue de centrales nucléaires. Cependant, les procédés d'extraction et les moyens de transport de l'uranium n'est pas connu. Une seule batterie sera utilisée durant l'usage du VE depuis sa mise en service jusqu'à sa fin de vie (pas de remplacement de cellules, de module ou de la batterie). Enfin, nous considérons que la recharge de la batterie a lieu après une décharge selon le profil de mission considéré.

Les concepts de l'approche Well-To-Wheels (WtW)

L'approche WtW [193, 194] est appliquée en pratique pour évaluer les impacts environnementaux liés à la production d'électricité nécessaire pour recharger le VE et transporter la batterie, et ce depuis l'extraction des ressources (uranium, charbon) jusqu'à son usage. Elle complète la Life-Cycle Assessment (LCA) (cf. Annexe D). L'approche WtW se décompose en plusieurs phases, comme présenté Fig. 3.32. L'exemple présenté dans ces travaux de thèse ne reprend pas l'approche WtW comme dans le benchmark [76] mais simplement

Figure 3.32 L'approche WtW pour l'énergie électrique [76].



les concepts de Plug-to-Tank (PtT), de Plug-to-Wheels (PtW) et de Well-to-Plug (WtP) pour répartir les équations du modèle gros grains dans des modèles de synthèse.

Positionnement des travaux de thèse

En éco-conception, on considère l'ajout d'exigences environnementales dès les phases de conception, ce qui n'est actuellement fait qu'en phase de validation en LCA. Le système est sous-défini en éco-conception, alors qu'il est défini en LCA. On ne cherchera pas à évaluer un indicateur environnemental à partir d'une architecture définie mais à le contraindre pour respecter des réglementations environnementales en vigueur ou pour réduire des taxes dûes aux émission de polluants.

Le benchmark [76] s'appuie sur une démarche de LCA couplée à la PPC pour évaluer les impacts environnementaux de la batterie. La PPC est utilisée pour la modélisation sous forme de CSP des problèmes continus avec le formalisme Minibex. Pour la conception, le solveur IbexSolve est utilisé. Un premier problème est traité dans [76] pour évaluer en particulier 4 effets sur l'indicateur GWP de la batterie : du mode de production de l'énergie électrique, du carburant et de l'usage du VE (plutôt BEV ou PHEV). Dans ces travaux de thèse, nous nous intéressons à la définition de l'architecture de la batterie en tenant compte d'exigences environementales, ce qui n'est pas le cas dans le benchmark. Les différences entre le benchmark et ces travaux de thèse sont reportés Tab. 3.3 pour souligner nos apports. Ces travaux de thèse sont à la fois différents et complémentaires du benchmark [76].

Table 3.3 Différences entre le problème présenté dans [76] et celui dans la thèse.

	Processus /Approche	**Niveau d'abstraction**	**Sys. à concevoir**	**Problème traité**	**But**
Benchmark [76]	Analyse : LCA et CSP Minibex IbexSolve (BP)	bas DSML	défini	continus	évaluer les impacts environnementaux du système
Travaux de cette thèse	Synthèse : Eco-conception et CSP DEPS DEPS Studio/CE (BP)	élevé DSML	sous-défini	mixtes	définir le système au regard d'exigences environnementales

Construction des exigences

En plus des exigences définies précédemment à la Sous-section 3.3.5, l'exigence suivante est ajoutée pour le problème d'éco-conception de la batterie Li-ion :

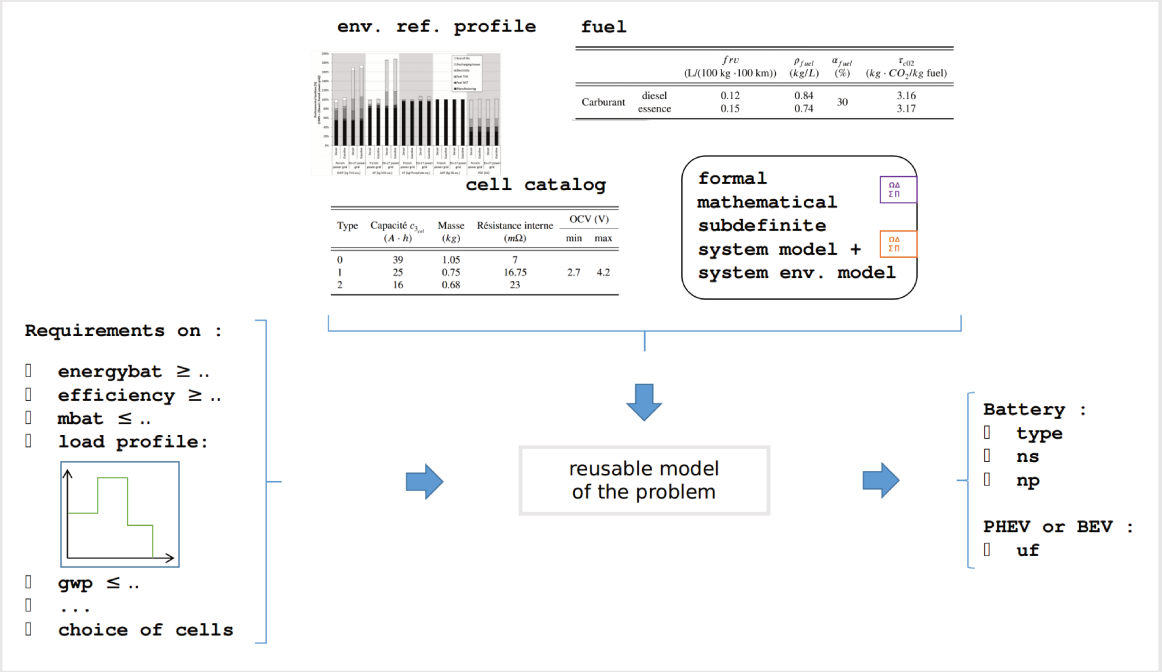
> E13 : l'indicateur d'impact environnemental GWP de la batterie ne doit pas dépasser 10000 kg CO_2 eq. pour l'énergie maximale de celle-ci.

Pour un cas industriel de conception, la valeur de 10000 kg CO_2 eq. serait issue de l'espace du problème (par exemple relative à une norme). Dans cet exemple, elle provient de l'espace des connaissances. Le choix de cette valeur est justifié comme suit. La valeur de référence la plus critique du GWP sera 151 kg équiv. CO_2 par kW.h [192]. Pour la valeur maximale de 100 $kW.h$ pour pour l'énergie maximal de la batterie (exigence E11), la valeur maximale du GWP est intéressante si inférieure ou égale à 15100 kg équiv. CO_2.

Choix des ddl

Les trois ddl de conception (`type`, `ns` et `np`) sont conservées. Nous choisissons d'en ajouter une quatrième : le facteur d'usage `uf`, qui indique la part roulée purement en électrique durant la durée de vie du VE. Ce ddl est intéressant puisqu'il est implicitement lié à l'architecture du véhicule. Si l'on souhaite concevoir par exemple un BEV, il faudra la contraindre à 1, son domaine étant défini par l'intervalle continu [0, 1]. Si l'on souhaite dans un premier temps obtenir l'ensemble des architectures admissibles indépendamment du type de VE, on peut choisir de ne pas le contraindre. Il est alors nécessaire de partiellement restructurer le modèle gros grains algébrique du benchmark, afin de pouvoir exprimer le modèle de synthèse correspondant au problème de manière quasi-statique.

Figure 3.33 Schématisation d'un problème d'éco-conception pour la batterie.



3.4.3 Formalisation de problèmes d'éco-conception

Un problème d'éco-conception pour la batterie peut être schématisé Fig. 3.33. Une des difficultés de formalisation consiste à relier les nouvelles exigences aux modèles de synthèse existants (`Problem` et `Models`), en limitant au plus leurs modifications et en conservant la modularité et la lisibilité de ceux-ci. Pour réduire les coûts de formalisation lors de la réutilisation des modèles de sythèse dans d'autres projets, les exigences non-fonctionnelles de type 2 devraient être découplées le plus possible du reste des exigences. C'est donc une problématique de formalisation qui dépend de la qualité de modularisation de l'ensemble du problème de conception initial. Les mêmes mécanismes de l'orienté-objet ont été employés pour cette extension : la référence, l'héritage, et la composition.

Modélisation d'un polluant

Les émissions de polluants sont quantifiées en termes de quantités pour les impacts locaux et de proportion pour les impacts globaux. Définir un `Model` de polluant général permet d'homogénéiser la formalisation (cf. Fig. 3.34). La spécialisation de ce modèle en deux `Model Polluant` de signatures différentes masque cette différence d'implémentation (cf. Fig. 3.35 pour la modélisation des polluants locaux).

Figure 3.34 Modélisation générale des polluants (DEPS).

```
578 (* General polluant model *)
579 Model Polluant()
580 Constants
581 Variables
582 expr mass : MASS ; (* battery related polluant mass *)
583 expr cfuel : REAL ; (* vehicle fuel consumption *)
584 expr densityfuel : DENSITY ; (* fuel density *)
585 expr mfuel : MASS ; (* battery related fuel mass *)
586 Elements
587 Properties
588 End
```

Figure 3.35 Modélisation des polluants locaux (DEPS).

```
590 Model Polluant(emission) extends Polluant
591 Constants
592 emission : REAL ; (* emission of polluant *)
593 Variables
594 Elements
595 Properties
596 mass := emission * mfuel /((cfuel * densityfuel)/100) ;
597 End
```

Figure 3.36 Modélisation des polluants globaux (DEPS).

```
598 Model Polluant(emissionf) extends Polluant
599 Constants
600 emissionf : FACTOR ; (* emission factor *)
601 Variables
602 expr emission : REAL ; (* emission of polluant *)
603 Elements
604 Properties
605 emission := emissionf * (cfuel * densityfuel)/100 ;
606 mass := mfuel * emissionf ;
607 End
```

Modélisation du VE

Le modèle *Vehicle* du VE sous-défini (cf. Fig. 3.37) permet de relier les exigences du problème de conception et celles du problème étendu. C'est à ce niveau de modélisation qu'est déclarée la variable de conception `uf`. Ce type de modèle est plus proche d'un modèle d'interconnexion `Connect` comme présenté Section 3.2, car il ne représente pas le véhicule réel (seule la batterie est prise en compte).

Figure 3.37 Modélisation du VE (DEPS).

```
609 Model Vehicle(mass, lcd, aer, fuel, battery)
610 Constants
611 mass : MASS ;
612 lcd : LENGHT ;
613 aer : LENGHT ;
614 (..)
615 Variables
616 uf : RATIO ; (* utility factor *)
617 (..)
618 expr efficiencybat : EFFICIENCY ; (* battery efficiency *)
619 expr mbat : MASS ; (* battery mass *)
620 expr avenergybat : ENERGY ; (* battery useful energy *)
621 (..)
622 Elements
623 fuel : Fuel[...] ;
624 battery : Battery ;
625 Properties
626 (..)
627 End
```

Modélisation d'un indicateur environemental

Une autre problématique de formalisation concerne la modélisation du GWP. Contrairement aux émissions locales de polluants qui ont principalement lieu pendant l'usage du véhicule, les émissions globales sont à allouer aux différentes étapes du cycle de production WtW de l'énergie électrique. Il faut également considérer la part d'énergie fossile consommée pour produire cette énergie électrique et celle nécessaire au déplacement de la batterie pendant l'usage du véhicule.

On peut envisager de créer une Variable pour chaque étape du cycle : `gwpWtP` (de la mine à la centrale), `gwpPtP` (de la centrale à la station de recharge), `gwpPtT` (de la station de recharge à la batterie) et `gwpTtW` (de la batterie à la roue). Ce choix de notation alourdit l'écriture des modèles (il faut déclarer des Variables de noms plus ou moins longs) et n'est pas très réutilisable.

Nous proposons une modélisation plus modulaire, qui utilise l'imbrication et la navigation (cf. Fig. 3.38). Un indicateur (ex : GWP) sera défini comme une instance du `Model GlobalEffects`. Les phases du WtW associées sont renseignées dans le modèle, ce qui permet une meilleure réutilisation de celui-ci. Les instances de `Fuel` et de `Electricity` aurait alors accès à ces Variables par navigation, par exemple : `fuel.TtW` et `electricity.TtW`.

Figure 3.38 Modélisation impacts environementaux globaux (DEPS).

```
629 Model GlobalEffects(co2)
630 Constants
631 Variables (* global indicators impacts *)
632 expr total : REAL ;
633 expr totalfuel : REAL ; (* total fuel impact *)
634 (..)
635 expr electricityWtP : REAL ; (* electricity production *)
636 (..)
637 expr manuf : REAL ; (* battery manufacturing *)
638 expr endoflife : REAL ; (* battery end of life *)
639 Elements
640 co2 : Polluant[FACTOR] ;
641 Properties
642 (..)
643 total := totalfuel + electricityWtP + manuf + endoflife ;
644 End
```

Figure 3.39 Modélisation du profil environnemental de référence (DEPS).

```
645 Model EnvironmentalRefProfile(mbatref, avenergybatref, manufref,
       fuelref, elecref, endofliferef)
646 Constants (* global indicators of the reference battery *)
647 mbatref : MASS ;
648 avenergybatref : ENERGY ;
649 manufref : REAL ; (* for the manufacturing *)
650 fuelref : REAL ; (* for producing 1 kg of fuel *)
651 elecref : REAL ;  (* for producing 1 kW.h of electricity *)
652 endofliferef : REAL ; (* for the end of life *)
653 Variables
654 Elements
655 Properties
656 End
```

Modélisation d'un profil environnemental et d'une charge environnementale

Un profil de référence peut être représenté par un modèle paramétré (cf. Fig. 3.39). On peut ainsi le réutiliser en modifiant les valeurs de la masse `mbatref`, de l'énergie utile `avenergybatref` et des impacts environnementaux (ex : `manufref`, `elecref`). Un `Model` de charge environnementale regroupera l'ensemble des impacts environnementaux globaux et locaux à limiter pour que les concepteurs puissent les inclure dans un problème sans avoir à réaliser de distinction (cf. Fig. 3.40).

Formalisation du problème d'éco-conception

Le `Problem` d'éco-conception d'une batterie pour VE à carburant diesel est formalisé par extension du `Problem` correspondant au S/C *Level* (cf. Fig. D.1). Seul le `Model` `MaxValues`

Figure 3.40 Modélisation de la charge environnementale globale (DEPS).

```
Model EnvironmentalBurden(co2, nox, co, hc, envref, vehicle,
    powergrid)
Constants
Variables
Elements
co2 : Polluant[...] ;
nox : Polluant[...] ;
co : Polluant[...] ;
hc : Polluant[...] ;
gwp : GlobalEffects(co2) ; (* GWP *)
envref : EnvironmentalRefProfile[...] ;
powergrid : PowerGrid[...] ;
vehicle : Vehicle[...] ;
charging : ChargePtT(vehicle, powergrid) ;
Properties
gwp.fuelWtT := vehicle.mfuel * envref.fuelref ;
(..)
gwp.manuf := vehicle.mbat * envref.manufref/envref.mbatref ;
gwp.endoflife := vehicle.mbat * envref.endofliferef/envref.mbatref ;
End
```

Figure 3.41 Modélisation des performances relatives au GWP (DEPS).

```
Model MaxValues(envb, gwpmax) extends MaxValues[...]
Constants
gwpmax : REAL ;
Variables
Elements
envb : EnvironmentalBurden[...] ;
Properties
envb.gwp.total <= gwpmax ;
End
```

existant a été modifié. La modélisation de l'exigence E13 est simple à implémenter par spécialisation du `Model MaxValues` construit précédemment (cf. Fig. 3.41). Cette modification est occultée lors de l'instanciation du `Model`. Les instances de `Models` sont paramétrées par des constantes nommées (ex : `nsmax`) pour rester en accord avec la version actuelle de DEPS Studio utilisée pour la résolution.

3.4.4 Résolution du problème d'éco-conception

Un problème d'éco-conception similaire a été traité pour un carburant essence. Les résolutions de ces deux problèmes ont été réalisées avec les solveurs CE (cf. Fig D.2) et DEPS Studio. La Table 3.4 présente 4 des architectures admissibles de batteries obtenues pour les deux problèmes d'éco-conception. Les architectures permettant de réduire les émissions de gaz à effets de serre favoriseraient principalement les véhicules quasi tout électriques (Sol. 2

Table 3.4 Quatre architectures de batteries pour le problème d'éco-conception.

Variables	Unités	Exig.	Gazole		Essence	
			Sol. 1	Sol. 2	Sol. 3	Sol. 4
`type`	/	E6	2	2	2	2
`np`	/	E8	9	9	9	9
`ns`	/	E7	128	125	121	122
`uf`	%	/	17	100	78	98
`mbat`	kg	E9	784	765	741	747
`energybat`	$W \cdot h$	E11	77414	75600	73180	73786
`icel`	A	E10	11.94	11.88	11.88	11.91
`ibat`	A	E1	107.5	107.0	107.0	107.2
`vbat`	V	E1	502	490	475	479
`powerbat`	W	E1	53990	52517	50837	51320
`soc`	%	E3	62.7	62.8	62.8	62.8
`efficiencybat`	%	E12	93.5	93.4	93.4	93.5
`gwp`	$kgCO_2$ eq.	E13	7419.2	4530.3	8158.7	4529.8

et 4 pour des facteurs d'usage `uf` égaux à 100 et 98) mais pas exclusivement (Sol. 3). Aucun carburant n'est favorisé en particulier bien que le gazole pourrait être un meilleur choix. On pourrait également envisager d'inclure une exigence relative à un choix de carburant pour ne résoudre qu'un unique problème d'éco-conception et gagner en temps de modélisation et de résolution.

3.5 Conclusion

Pour répondre au second verrou, dans ce Chapitre 3, nous avons abordés en détail la formalisation des problèmes de conception et d'éco-conception avec un niveau d'abstraction supérieur à celui utilisé au Chapitre 2. Ce chapitre a montré que l'on pouvait prendre en compte simultanément des exigences fonctionnelles et non-fonctionnelles aussi bien dans la modélisation que dans la résolution. À l'aide d'un formalisme comme DEPS qui présente des capacités de structuration pour un système structuré sous-défini, il est possible de formaliser naturellement et explicitement un problème de conception de systèmes physiques. Une aide à la formalisation associée comme proposée offre une modélisation plus extensible et réutilisable. Les modèles de synthèse présentent un certain niveau de généricité et peuvent donc être réutilisées dans d'autres projets. Ces modèles de synthèse ont montré leur potentiel de réutilisabilité au sein de deux projets, la conception et l'éco-conception d'une batterie. Suite à ces résultats encourageants aussi bien au niveau de la formalisation que de la résolution, nous pouvons nous intéresser à la suite du processus de pré-conception, à savoir que

faire des architectures admissibles obtenues ? Peut-on alors les réutiliser directement pour la conception détaillée ? Si oui, lesquelles sélectionner ? Seront-elles toujours admissibles face à un modèle grains fins de système ?

Nous savons déjà que ces architectures sont admissibles par construction puisqu'elles sont le résultat d'un calcul par Programmation Par Contraintes (PPC). Elles n'ont donc pas besoin d'être vérifiées (sauf si les modèles doivent être modifiés et/ou enrichis, ce qui est le cas lors des phases d'analyse). En revanche, ces architectures ne sont admissibles que relativement aux exigences modélisées. L'approche de synthèse proposée se limite aux modèles de type boîte blanche et qui sont gros grains algébriques. Les outils de synthèse actuels comme DEPS Studio sont ainsi limités dans la prise en compte de certaines exigences. L'inconvénient principal des modèles gros grains algébriques est qu'ils ne permettent de modéliser tous les aspects du comportement d'un système, notamment les phénomènes locaux des systèmes dynamiques, à évènements continus ou discrets (ex : évaluer une trajectoire ou éviter certains états critiques du système). De plus, on ne dispose pas nécessairement de modèles gros grains algébriques (que grains fins ou boîte noire). Une approche d'analyse permet en particulier d'étudier ce aspects du comportement d'un système (pas seulement pour la dynamique mais aussi les exigences non-fonctionnelles de type 2 comme la LCA, la sûreté de fonctionnement, etc.) ainsi que d'utiliser des modèles type boîte noire. Dans ce contexte, il semble nécessaire d'étendre l'approche de synthèse par une approche d'analyse, toujours en pré-conception. Le Chapitre 4 cherchera à répondre au troisième et dernier verrou en s'interessant à l'analyse des architectures. Leurs modélisations et leurs simulations seront explorées avec des modèles plus fins ou lors d'ajout d'exigences qui n'avaient pas pû être prises en compte lors de ce premier processus de synthèse.

Chapitre 4

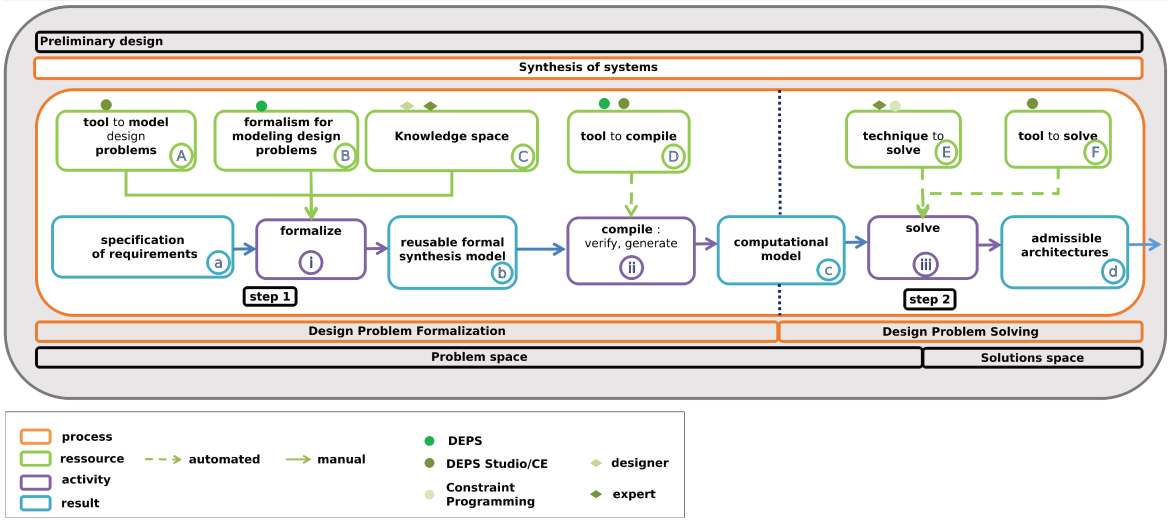
De la synthèse à l'analyse : vers une approche globale et intégrée

4.1 Introduction

Dans ce quatrième chapitre, nous cherchons à répondre au troisième et dernier verrou : les avantages d'une approche d'analyse étant les inconvénients de celle de synthèse, peut-on allier les deux pour concevoir en GE ? Dans ce dernier chapitre, nous proposons ainsi une approche fonctionnelle et intégrée pour la pré-conception de systèmes physiques complexes, en particulier pour le GE. Nous proposons dans cette partie de compléter le processus de synthèse présenté au Chapitre 3 (cf. Fig. 4.1) par un processus d'analyse d'architectures. Nous procéderons en particulier à l'évaluation raffinée d'architectures admissibles obtenues par synthèse.

Une approche alliant synthèse et analyse est présentée dans [96, 113]. Les similarités et différences par rapport à ces travaux de thèse sont reportées au Tab. 4.1. Les travaux présentées dans cette thèse se situent dans la poursuite de [96, 113]. Ces travaux ainsi que [96, 113] traitent de la pré-conception intégrée de systèmes physiques par synthèse et une extension à l'analyse. [96, 113] utilise une approche pour le dimensionnement des systèmes mécaniques principalement axée sur la résolution. La formalisation repose sur la Programmation Par Contraintes (PPC) et est illustrée avec le formalisme Constraint Explorer (CE) et les modèles de synthèse résultants sont peu réutilisables et peu adaptés à l'ingénierie comme vu au Chapitre 2. Les exigences sont fonctionnelles et non-fonctionnelles de type 1 (intégration et performances). Pour l'analyse des architectures, les modèles de systèmes pris en compte sont des modèles d'état.

Figure 4.1 Le processus de synthèse de système complété.



Cette thèse se concentre principalement sur la synthèse et la formalisation des problèmes de dimensionnement / génération d'architectures / configuration des systèmes électriques et de leurs réutilisabilités, pour l'ingénierie et à l'aide de DEsign Problem Specification (DEPS). Ces travaux proposent d'inclure également les exigences non-fonctionnelles de type 2 (comme environnementales, coûts, avec un modèle mathématique différent). Les modèles boîte blanche que nous choisissons pour représenter le comportement dynamique du système en pré-conception lors de l'analyse pourront être non linéaires et gros grains algébro-différentiels. Dans ce chapitre, nous avons fait le choix de center l'analyse sur le comportement dynamique du système (exigences fonctionnelles et non-fonctionnelles de type 1). L'utilisation de Modelica [195] et OpenModelica comme exemples pour représenter et simuler le comportement dynamique du système sera évoqué dans la Section 4.3.2 de ce chapitre. Les modèles gros grains algébro-différentiels peuvent parfois être vus comme complémentaires aux algébriques car les outils de simulation comme OpenModelica ne peuvent pas simuler tous les modèles gros grains algébriques [195].

4.2 Besoins de phases de validation

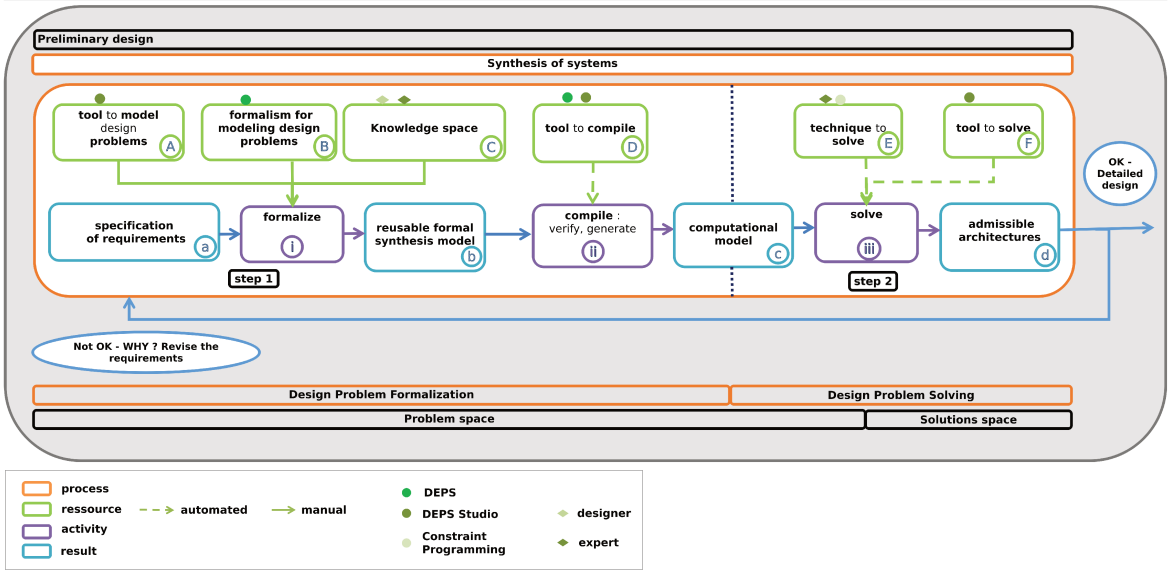
Une approche de conception inclut en pratique des phases de validations, car il n'est jamais garanti de pouvoir trouver la solution au problème sans avoir à revenir aux phases de définition des exigences ou à celles de de conception.

Table 4.1 Similarités et différences entre notre approche et [96, 113].

	Travaux de thèse proposés	**[96, 113]**
	Différences	
Synthèse		
Formalisation illustrée avec	DEPS	CE
Niveau de formalisation	modèle structuré et réutilisable	modèle "plat"
Axé	bureau d'études	"mathématique"
Problèmes	génération, configuration, dimensionnement	dimensionnement
	mixtes	mixtes
Domaines	sys. électriques	sys. mécaniques
Exigences	fonctionnelles, non -fonctionnelles type 1 et 2	fonctionnelles, non -fonctionnelles type 1
Analyse		
Résolution des équations pour simulation	analyse numérique (Modelica)	par intervalles
Modèles	algebro-différentiels, différentiels	différentiels
	Similarités	
Synthèse		
	Comportement du système quasi-statique Résolution par PPC	
Analyse		
	Comportement du système dynamique	

Les étapes de validation proposées dans ce chapitre s'interessent aux causes de non-validation d'architectures admissibles et aideront à réduire le nombre d'itérations entre les phases de définition des exigences / pré-conception et de pré-conception / conception détaillée. Cette section explore ainsi les causes de non-solvabilité du problème à l'issue de la synthèse (le problème de conception n'admet pas de solution) et de perte d'admissibilité à l'issue de l'analyse. Des propositions seront aussi faites pour répondre à ces besoins de validation qui constituent la principale limite de conception de l'approche globale de conception proposée dans ces travaux de thèse.

Figure 4.2 Révision de la spécification des exigences.



Nous avons identifié 3 raisons pour lesquelles ces itérations entre phases pourraient tout de même subsister : le problème n'admet pas de solution, le modèle gros grains utilisé lors de la synthèse ou celui pour l'analyse n'est pas ou plus valide respectivement.

Problème non-solvable

Aucune architecture admissible n'a été trouvée à l'issue de la synthèse de systèmes. Puisque l'utilisation de la PPC implique que les solutions sont correctes-par-construction, il faut se tourner dans un premier temps vers la spécification des exigences afin de vérifier que le problème a été bien formalisé. Cette vérification consiste à s'assurer qu'il n'y a pas eu d'erreurs de formalisation des exigences (comme une valeur de performances minimale mal transcrite - cf. Fig. 4.2).

Il faut également vérifier les choix réalisés depuis l'espace des connaissances, en particulier, les choix de modèle gros grains pour le problème (ex : hypothèses trop fortes, modèle non réaliste) et de domaines initiaux pour les ddl (trop restreints). Dans un second temps, il faut s'assurer que le bon problème a été formalisé (interprétation des exigences). C'est la validation du modèle de synthèse. Dans un troisième temps, si la vérification et la validation ont été approuvées, on peut supposer que c'est la spécification des exigences qui doit être revue entre les différentes parties prenantes, notamment avec la clientèle et il faudra relxer des exigences (cf. Fig. 4.2 - *Not OK - Why ? Revise the requirements*).

Avant d'évaluer si une architecture peut perdre son admissibilité, nous avançons trois hypothèses qui pourraient y mener.

Invalidité du modèle gros grains algébrique

Cette première hypothèse suppose que le modèle gros grains algébrique utilisé pour formaliser le problème durant la synthèse n'est plus valable. C'est le cas notamment lorsque de nouveaux et meilleurs modèles gros grains algébriques du système sont disponibles. C'est également le cas si un mauvais modèle a été choisi (ex : qui ne représente pas le bon système). Dans ce cas, ce modèle ne concorde plus avec le modèle gros grains utilisé pour les phases de validation (simulations à l'aide de modèles algébro-différentiels plus précis et valides).

Invalidité du modèle gros grains algébro-différentiel

Cette seconde hypothèse suppose l'inverse : le modèle gros grains algébrique utilisé pour la synthèse est valide contrairement aux modèles gros grains algébro-différentiel pour l'analyse. Si une seule des deux hypothèses précédentes se valide, il faut alors mettre à jour le modèle correspondant en réévaluent le choix du modèle gros grains depuis l'espace des connaissances. En revanche, si aucun des deux modèles gros grains n'est valide mais qu'ils sont en accord, une architecture restera admissible après synthèse et analyse, bien qu'elle ne le soit pas en réalité. Eventuellement, il peut être utile de s'appuyer sur un modèle grains fins différentiel (sous réserve qu'il soit valide). Etant mieux représentatif du comportement du système, il permettrait de trancher sur l'admissibilité d'une architecture.

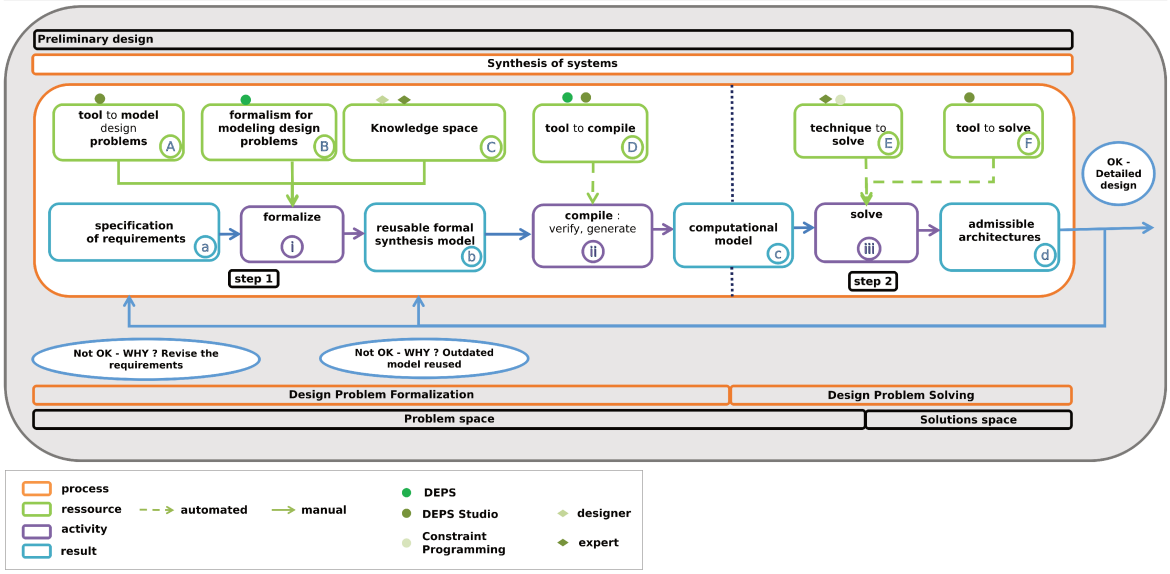
Nous pouvons ainsi compléter le processus de synthèse de systèmes par 2 boucles de conception comme sur la Fig. 4.3 (*Not OK - Why ? Outdated model reused*).

Architecture à la limite de l'admissibilité

Nous avons évoqué la possibilité qu'une architecture perde son admissibilité après sa simulation. Nous pouvons nous questionner sur les causes qui y conduiraient. Ces causes potentielles seront étudié dans le processus d'analyse d'architectures. Nous proposerons également d'aider au choix d'architectures qui ne perdraient pas leur admissibilité après leurs simulations.

Cette troisième hypothèse suppose que l'architecture est à la limite de l'admissibilité pour une ou plusieurs exigences fonctionnelles ou non. Il s'agirait en particulier de contraintes

Figure 4.3 Révision des modèles gros grains.



d'inégalités. Cette architecture se trouve à la limite de validité du modèle gros grains algébrique. Le raffinement dynamique de l'architecture permet de réaliser qu'elle n'est en fait pas admissible. Les étapes présentées ci-après proposent de construire le processus d'analyse présenté en partant d'architectures qui seraient issues de la synthèse, seraient modélisées plus finement puis seraient simulées.

4.3 Approche complète : conception, validations et itérations

4.3.1 Résumé des étapes de la synthèse de système (step 1 à 2)

Cette sous-section résume les étapes de la synthèse complétées par les formalismes et outils choisis au Chapitre 3.

L'étape *step 1* propose la pose formelle et haut-niveau d'un problème de conception et sa résolution dans un environnement intégré. Elle s'articule autour du Domain-Specific Modeling Language (DSML) DEPS et DEPS Studio, qui donnent un cadre formel pour la modélisation de problèmes en pré-conception. Le cahier des charges textuel exprimant le problème en langage "naturel" est formalisé avec DEPS. L'espace des connaissances permet à ce stade de choisir des modèles gros grains et réalistes pour la pré-conception du système à concevoir

et de ceux intervenant dans le problème, et du profil de mission. L'ensemble de ces modèles gros grains constituent le modèle gros grain du problème. Ce modèle gros grain du problème doit être algébrique (en représentant les différents systèmes de manière quasi-statique).

La spécification des exigences est modélisée avec DEPS dans DEPS Studio. Lors de la modélisation du problème, les connaissances expertes peuvent aiguiller dans le choix de domaines de valeurs pour certaines variables. Les enjeux liés à la réutilisabilité des différents modèles issus de cette formalisation sont aussi explorés à ce stade. Ces enjeux sont notamment les choix des "objets" à définir dans DEPS, l'emploi des relations d'héritage, de composition, etc. Un ou plusieurs modèles formels du problème peuvent être construits. Ces modèles de synthèse chercheront à faciliter la compréhension du problème pour les concepteurs. Ils sont ensuite vérifiés en DEPS par le compilateur de DEPS Studio.

L'étape *step 2* traite de la résolution du problème. Un modèle de calcul générique de type Constraint Satisfaction Problem (CSP) est généré pour la résolution. À partir de ce modèle, l'ensemble des architectures admissibles est synthétisé avec le solveur intégré à DEPS Studio ou le solveur Constraint Explorer (CE) à l'aide de techniques de Programmation Par Contraintes (PPC).

Si aucune solution n'a été trouvée à l'issue de la synthèse de système, il faut revenir à l'étape *step 1* et revoir la formalisation (i) de la spécification des exigences (a) (*Revise the requirements*) voire la construction des modèles de synthèse (*Outdated model reused*).

4.3.2 Validation par analyse d'architectures (step 3 à 5)

L'analyse des architectures doit permettre de concevoir le système plus finement. Elle repose sur des langages, outils et techniques pour la spécification de systèmes, permettant leurs modélisations et leurs simulations. Ce processus occure depuis l'espace des solutions et celui des connaissances.

Etude et choix d'architectures admissibles (step 3)

L'objectif de cette étape *step 3* est d'étudier les architectures qui auraient le plus de chances de rester admissibles après l'analyse-simulation (d'être validées) pour filtrer d'avantage les résultats obtenus par la synthèse (qui est un premier filtre pour la conception). Depuis l'espace des connaissances, l'expertise métier selectionne une architecture parmi l'ensemble obtenu après synthèse. Cette selection pourrait être effectuée après une étude de cet ensemble

par l'expertise métier. On peut alors s'intéresser aux contraintes d'inégalités les plus critiques (comme fonctionnelles ou celles qui seront raffinées pendant l'analyse), qui peuvent constituer une raison de non-validation d'architectures après simulation comme évoqué précédemment. Cette étape de choix d'architectures admissibles est étudiée avec l'exemple de la batterie en Section 4.4.

Communication et validation par analyse-description (step 4)

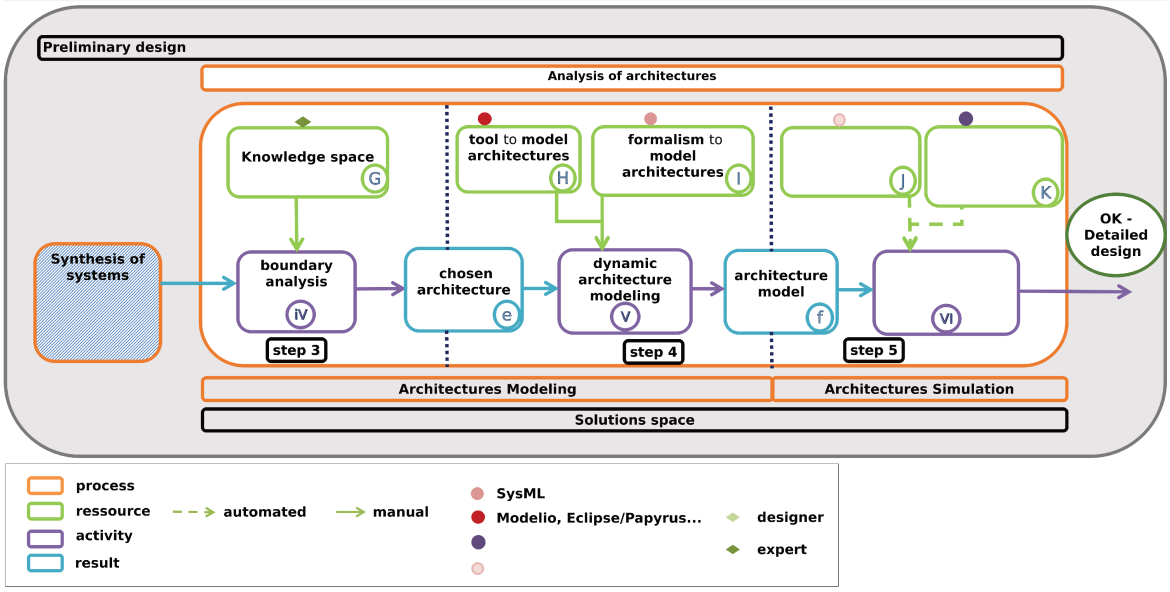
A l'étape *step 4*, une ou plusieurs architectures sont modélisées à l'aide du DSML SysML et d'un environnement de modélisation comme Eclipse/Papyrus. La modélisation est algébrique. Le modèle gros grains algébrique peut d'abord être transcrit de manière informelle au sein d'une étape d'analyse-description. Cette étape d'analyse-description pourra permettre de communiquer entre parties prenantes les architectures sélectionnées à l'étape précédente. Elle est réalisée à l'aide d'un langage permettant de séparer la formalisation des exigences de celle des systèmes comme SysML. Comme vu au Chapitre 1 en Section 1.3.3 et détaillé en Annexe E, SysML serait adapté car ce langage se prête actuellement plus à la description d'architectures qu'à leurs simulations (les diagrammes n'étant pas tous simulables et ceux qui le sont doivent être correctement formalisés avec OCL). Les exigences non-fonctionnelles de type 2 peuvent être représentés au sein du diagramme d'exigences Requirement (req), bien que seulement de manière informelle. L'Annexe E étudie également les besoins de modélisation en comportement du système pour une analyse-description avec SysML (en plus des possibilités et limites de SysML pour la conception par synthèse) en reprenant l'exemple de la batterie introduit au Chapitre 3.

Simulation et validation par analyse-simulation (step 5)

A l'étape *step 5*, les architectures sont modélisées et simulées. L'analyse par un modèle plus fin permettrait de valider ou non une architecture initialement admissible. Nous recherchons des outils et formalismes dédiés permettent de simuler une architecture dont le comportement serait décrit par un modèle algébro-différentiel pour pouvoir compléter la Fig.4.4.

L'analyse plus fine d'une architecture repose aussi sur la modélisation de systèmes de mesures (capteurs) permettant de surveiller le comportement du système pendant sa simulation. Ces informations nécessaires à la validation, ajoute notamment des calculs supplémentaires durant la simulation. Les mesures des grandeurs comportementales sont nécessaires afin de

Figure 4.4 L'analyse-description (step 4).



vérifier que les contraintes de conception sont toujours respectées. Les modèles à construire pour l'analyse-simulation sont le modèle du système défini considéré, du profil de mission et de ceux des capteurs et celui de simulation (qui est une interconnexion de tous les modèles). Pour cette étape, les architectures peuvent être modélisées et simulées à l'aide du DSML Modelica et d'un environnement de modélisation et de simulation comme OpenModelica comme justifié ci-après.

Avantages de Modelica Modelica est un langage textuel de modélisation déclaratif bien qu'il ressemble aux langages de programmation orienté-object comme le C++ et que les modèles puissent inclure des algorithmes.

Modelica est une standardisation de formalisme de modélisation multiphysique pour la simulation et non la conception comme DEPS. Les modèles Modelica sont des modèles d'analyse. Le système à concevoir doit donc être défini. Avec Modelica, les concepteurs travaillent dans l'espace des solutions (à partir d'architectures obtenues avec DEPS/DEPS Studio). Les modèles DEPS sont quant à eux des modèles de synthèse et le système à concevoir est sous-défini. Avec DEPS, les concepteurs travaillent dans l'espace du problème (pour arriver à celui des solutions).

En ce qui concerne la modélisation des systèmes, Modelica possède un haut niveau d'expressivité qui permet la représentation de nombreux types d'interactions entre systèmes. Notamment à travers la combinaison de la modélisation par vision causale (via la définition de blocs) et acausale (les sous-système/composants sont connectés via des terminaux d'interconnexion). Modelica n'utilise maintenant quasiment que la représentation en termes de variables *across* et *through* pour décrire des constituants physiques en termes objet.

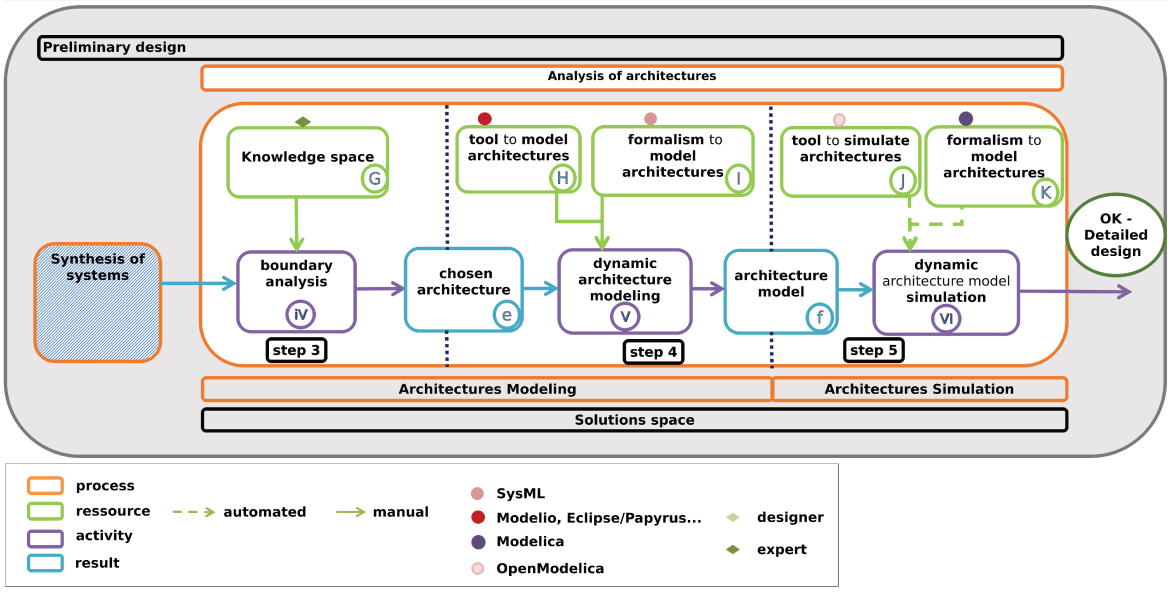
Le calcul symbolique permet une automatisation des phases de résolution. En particulier, les équations sont automatiquent réorganisées pour leur résolution. On préferera les outils Modelica à ceux tel que Matlab Simulink qui ne permettent qu'une vision purement causale du comportement du système.

Un autre avantage est l'existence de nombreux outils intégrés de formalisation et de simulation d'architectures implémentant Modelica (Amesim, Dymola, Mathematica, Maple ou encore Open Modelica). Un autre des avantages de Modelica est qu'il est pensé en termes de réutilisabilité (orienté-objet) et propose donc de nombreuses bibliothèques que nous pouvons utiliser dans notre étape d'analyse-simulation. Ces bibliothèques génériques proposent un large choix de modèles d'analyse multi-physiques (électrique, thermique, mécanique, électronique) et sont donc adaptés pour l'ingénierie.

Limites de Modelica Modelica a été conçu pour la simulation du comportement dynamique d'architectures. On ne cherche pas à modéliser le systèmes sous différentes vue. Il n'existe en effet pas de vue spécifique permettant d'exprimer les contraintes de conception et plus particulièrement la spécification des exigences. Il n'y a donc pas de traçabilité entre les exigences, les contraintes de conception et les différents modèles qui doivent les satisfaire (contrairement à SysML). En ce sens, Modelica se situerait à la limite entre la pré-conception et la conception détaillée. En particulier, il ne sera pas possible d'évaluer le respect des exigences non-fonctionnelles de type 2. Celles-ci nécessitent des outils métiers plus spécifiques (ex : le logiciel GREET [196] pour la LCA).

Les différentes étapes de validations du processus d'analyse d'architectures en pré-conception peuvent être organisées comme sur la Fig. 4.5.

Figure 4.5 Processus d'analyse d'architectures complété après analyse-simulation (step 5).

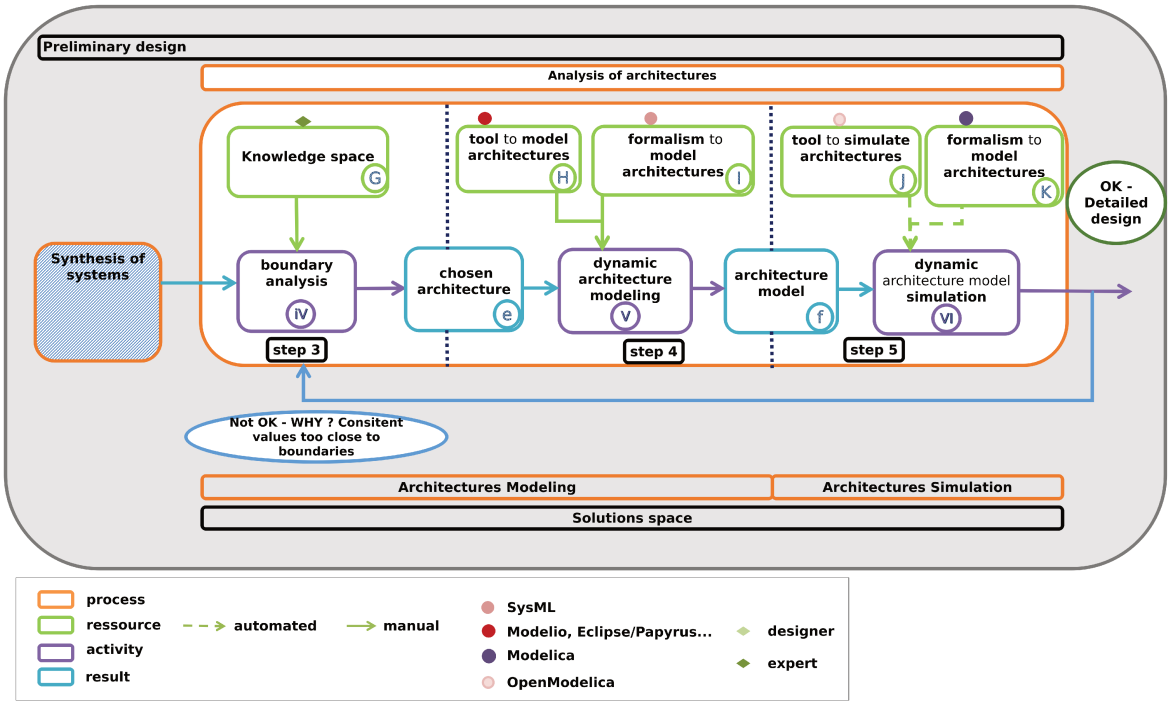


4.3.3 Perte d'admissibilité après analyse

On peut envisager qu'une architecture admissible perde son admissibilité après analyse avec un modèle plus fin et de la prise en compte de contraintes additionnelles. Dans ce cas, il faut retourner vers l'étape d'étude des architectures (step 3) et sélectionner une autre architecture admissible, s'il en existe. S'il n'en existe aucune, deux possibilités sont envisagées. Il faut soit sélectionner une autre architecture à analyser issue de la synthèse. Sinon, si l'on peut réinjecter les contraintes issues de l'analyse (faisant que l'architecture n'est plus admissible) directement dans les modèles de synthèse, on génèrera ainsi de nouvelles architectures admissibles au regard des nouvelles contraintes injectées.

Nous proposons ainsi de compléter l'analyse comme présenté Fig.4.6 (*Not OK - WHY ? Consistent values too close to boundaries*). Si aucune solution n'a été trouvée à l'issue de l'analyse des architectures, il faut revenir à l'étape *step 3* pour choisir une nouvelle architecture voire à l'étape *step 1* (*Revise the requirements* ou *Outdated model reused*) si aucune autre architecture n'est disponible. Si l'analyse des architectures confirme l'admissibilité de la solution simulée, la pré-conception se termine. La conception se poursuit vers la phase de conception détaillée.

Figure 4.6 Révision des architectures à sélectionner.

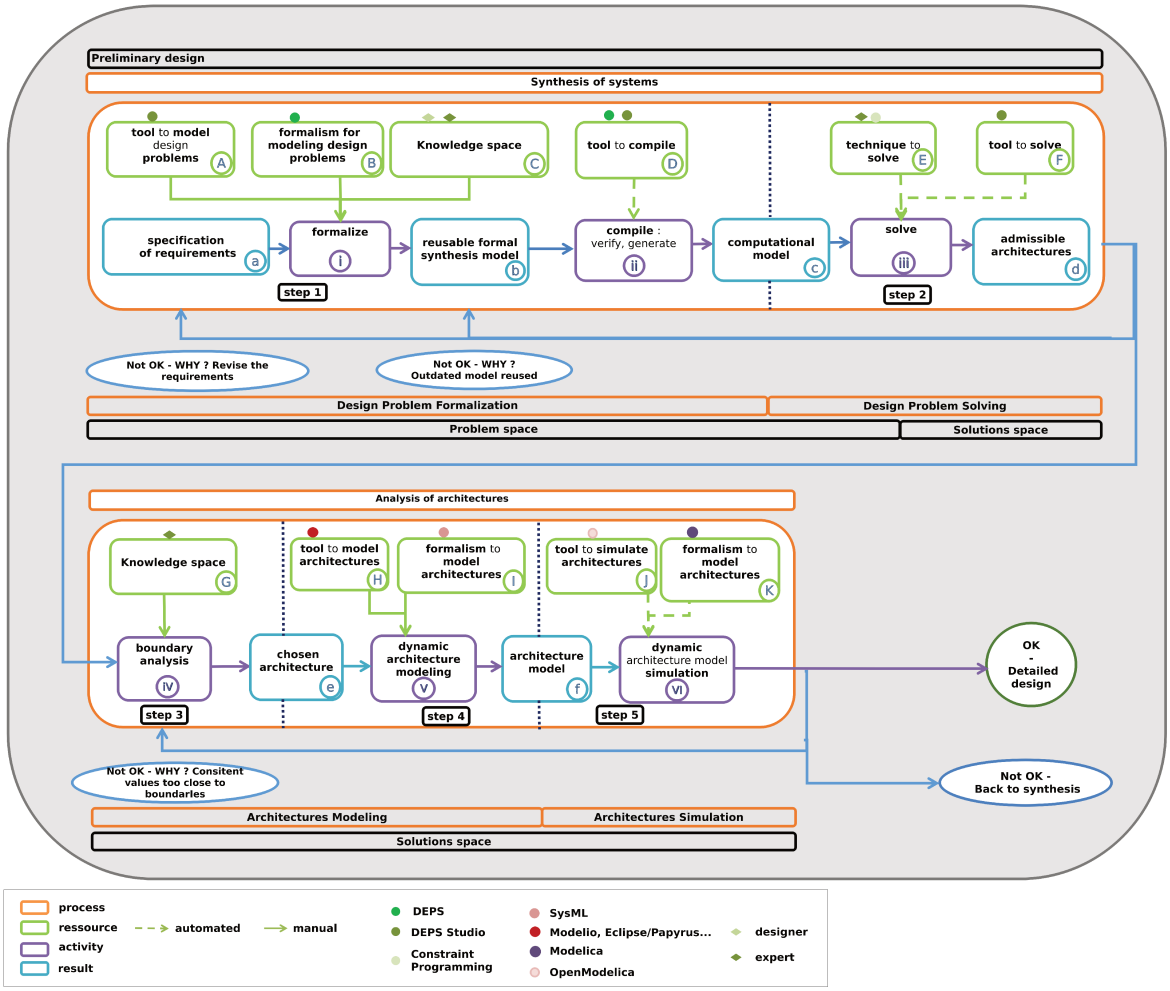


L'approche complète basée modèles pour la pré-conception intégrant de manière complémentaire le processus de synthèse de système ainsi que celui d'analyse / validations d'architectures est ainsi complété telle que présentée Fig.4.7. Cette approche pour la pré-conception se compose ainsi de 2 processus (conception par synthèse et validation par analyse), chacun décomposé en sous-processus. Chaque sous-processus est composé d'activités, de ressources et de résultats réparties en 5 étapes (step 1 à 5). Cette approche est finalement complétée par des itérations qui peuvent être nécessaires entre les processus de conception.

4.4 Application à l'analyse d'une batterie Li-ion

Cet exemple d'illustration reprend le cas d'étude de conception proposé au Chapitre 3 et a deux objectifs. Nous cherchons d'une part à valider ou non le choix menant aux choix d'architectures admissibles à modéliser (*step 3*) en se basant sur leurs valeurs pour les contraintes d'inégalité (la contrainte est activée ou l'est presque). D'autre part, nous cherchons à valider l'étape *step 5* qui cherche à valider ou non des architectures initialement admissibles par leurs simulations plus fine (est-ce que ce nouveau filtrage est efficace ?).

Figure 4.7 Approche de conception basée modèle pour la synthèse et l'analyse de systèmes en pré-conception.



Les architectures admissibles obtenues par synthèse sont étudiées dans cette section avec l'outil de modélisation/simuation OpenModelica. Les modèles présentés réutilisent ou étendent des modèles de systèmes et composants présents dans les librairies Modelica Standard Library (MSL) et Electrical Energy Storage (EES) [197]. Les exigences environnementales ne pouvant pas être évaluées avec les outils Modelica, elles ne sont pas modélisées. Nous traitons ainsi les architectures obtenues pour le problème de conception et non d'éco-conception de la batterie Li-ion.

Figure 4.8 Un *model* en Modelica.

```
model ModelName
parameter Param //optional - arguments ;
output OutMes //optional - mesure ;
input In //optional - deduced variable
Class instance ;
...
equation //
connect(component1.pin1, component2.pin1)
    ;
...
end ModelName ;
```

Figure 4.9 Un *bloc* en Modelica.

```
block BlockName
// structural part
parameter Param //optional - argument ;
output OutMes //optional - mesure data ;
input In //optional - input data
Class instance ;
...
// behavior part
equation
connect(component1.pin1, component2.pin1)
    ;
...
end BlockName ;
```

4.4.1 Construction des modèles Modelica

La syntaxe utile à la compréhension d'un `model` Modelica comme présenté dans ce Chapitre est donné Fig. 4.8. Les lignes commentées par "//optional -" sont optionnelles. La syntaxe utile à la compréhension d'un `bloc` Modelica est donné Fig. 4.9. C'est le modèle causal de composant sous forme de schéma-bloc (entrées/sorties). Nous ne pouvons rappeler toutes les bases de Modelica dans cette thèse, qui peuvent être retrouvées dans [198, 195].

4.4.2 Modélisation d'une architecture de batterie

Modèle électrique R_{int}

Le modèle d'architecture de batterie selon le modèle gros grains électrique équivalent R_{int} est présenté Fig. 4.10. En plus de la résistance interne `r` (cf. ligne 717) et de la source de tension parfaite *ocvbat* (cf. ligne 715), une table de correspondance `ocv-soc` sont présentes pour réaliser les mesures. Ce modèle est paramétré par les nombres série `ns` et parallèle `np` de cellules et le SOC initial (cf. lignes 709-712).

Calcul du SOC d'une cellule/batterie

La variation réelle de l'état de charge $soc(t)$ d'une cellule à l'instant t est à présent donnée par l'équation 4.1, avec resp. t_{ini} et t_{stop} les instants de début et de fin de simulation. $t_{stop} - t_{ini}$ correspond à la duration du profil de mission :

$$soc(t) = soc_{ini} - \frac{\int_{t_{ini}}^{t_{stop}} i_{cel}(t)dt}{c_i(t)} \tag{4.1}$$

Figure 4.10 Modélisation d'une batterie (adapté de EES - Modelica).

```
model StaticResistanceScaled "Battery stack model with a static internal impedance"
parameter Integer ns "number of serial cells";
parameter Integer np "number of parallel cells";
extends
parameter Real SOCini "Initial state of charge";
Modelica.Electrical.Analog.Interfaces.NegativePin pin_n "Negative pin";
Modelica.Electrical.Analog.Interfaces.PositivePin pin_p "Positive pin";
Modelica.Electrical.Analog.Sources.SignalVoltage ocvbat ;
Modelica.Electrical.Analog.Sensors.CurrentSensor ibat ;
Modelica.Electrical.Analog.Basic.HeatingResistor r(final R_ref = cellParameters.r.R0 * ns / np,..);
Modelica.Blocks.Math.Gain coef1(final k = ns);
Modelica.Blocks.Math.Gain coef2(final k = 1 / np) ;
Modelica.Blocks.Sources.Clock clock(final startTime = tini);
Components.Calculators.CellCalculator cellCalculator(final SOCini = SOCini, final Z0 = cellParameters.r.R0, final capacity = cellParameters.capacity) ;
Modelica.Blocks.Tables.CombiTable1D sococvTable(final tableOnFile = cellParameters.SOCOCV.OCVtableOnFile, final table = cellParameters.SOCOCV.OCVtable,..);
final parameter Modelica.SIunits.Time tini "Initial time";
parameter Battery.EES.CellRecords.StaticResistance.StaticResistanceParameters cellParameters;
equation
connect(ocvbat.n, pin_n);
connect(r.p, ocvbat.p);
connect(coef1.y, ocvbat.v);
connect(coef2.u, ibat.i);
connect(cellCalculator.i, coef2.y);
connect(ibat.p, r.n);
connect(ibat.n, pin_p);
connect(sococvTable.y[1], coef1.u);
connect(sococvTable.u[1], cellCalculator.SOC);
connect(cellCalculator.t, clock.y);
end StaticResistanceScaled;

```

Figure 4.11 Modélisation pour le calcul du SOC d'une cellule/batterie (EES - Modelica).

```
block SOC "State of Charge calculator"
    extends Icons.Block;
  parameter Real SOCini "Initial state of charge";
  Modelica.Blocks.Interfaces.RealInput C "Connector of Real input signal";
  Modelica.Blocks.Interfaces.RealOutput SOC "Output signal connector";
  Modelica.Blocks.Interfaces.RealInput i "Connector of Real input signal";
  Modelica.Blocks.Continuous.Integrator integrator(y_start = SOCini, k = -1);
  Modelica.Blocks.Math.Division divide;
equation
  connect(i, divide.u1) ;
  connect(C, divide.u2) ;
  connect(divide.y, integrator.u);
  connect(integrator.y, SOC);
end SOC;
```

La dépendance au temps dans cette équation est modélisée sous forme causale dans le domaine temporel par un schéma-bloc (instance `integrator` du bloc `Integrator` - cf. Fig. 4.11). Cette instance `integrator` doit notamment être paramétrée par une valeur initiale pour le SOC, soc_{ini}.

Calcul de la capacité nominale C d'une cellule

La capacité nominale d'une cellule diminue au cours du temps (vieillissement calendaire) et de son utilisation (vieillissement dû aux cycles de charge/décharge) [197] comme présenté Fig. 4.12. $Qabs$ représente la somme de la quantité de charge initialement stockée Q_{ini} et celle

Figure 4.12 Modélisation pour le calcul de la capacité d'une cellule (Modelica).

```
752 block Capacity "Capacity calculator"
753   extends Icons.Block;
754   parameter Battery.EES.CellRecords.
        Components.ChargeCapacity capacity
        "Charge capacity";
755   Modelica.Blocks.Interfaces.RealInput
        Qabs "Connector of Real input
        signal";
756   Modelica.Blocks.Interfaces.RealInput T
        "Connector of Real input signal";
757   Modelica.Blocks.Interfaces.RealInput t
        "Connector of Real input signal";
758   Modelica.Blocks.Interfaces.RealOutput C
        "Output signal connector";
759 equation
760   C = capacity.C0 * (1 + capacity.aging.
        kt * t + capacity.aging.kQabs *
        Qabs) * (1 + capacity.temperature.
        alpha * (T - capacity.temperature.
        Tref));
761 end Capacity;
```

de la charge transférée au cours de l'analyse-simulation, tel que :

$$Q_{abs} = Q_{ini} + \int_{tini}^{tstop} i_{cel}(t)dt \tag{4.2}$$

Dans le modèle R_{int}, le vieillissement d'une cellule n'est pas pris en compte, tel que $k_t = k_{Qabs} = 0$. k_t et k_{Qabs} représentent resp. les coefficients de vieillissement calendaire et de vieillissement dû aux cycles. Nous ne prenons pas en compte non plus l'influence de la température, tel que "`alpha` $= 0$". La capacité nominale est alors réduite à sa valeur maximale, lorsque la cellule est neuve.

Modèle de calcul global des paramètres de simulation d'une cellule : CellCalculator
Ce modèle permet simplement de rassembler tous les paramètre de simulation à calculer pour une cellule, en reliant les entrées et sorties requises pour chacun. En particulier pour ce modèle de décharge, on s'intéressera à la valeur calculée de la capacité en sortie de l'instance `c` du bloc `Capacity`, qui est réinjectée en entrée de celle permettant le calcul du SOC (cf. Fig. 4.13).

4.4.3 Modélisation d'une cellule

Pour ce modèle équivalent R_{int}, on peut également directement modéliser la batterie comme une cellule. Les variables d'entrées et de sorties ainsi que les caractéristiques d'une cellule seront alors accessibles via la structure de données paramétrable StaticResistance-Parameters (cf. Fig. 4.14) pour ses données structurelles (masse, type, etc.) et à travers une instance du bloc `CellCalculator` pour ses données comportementales (courant, SOC, etc).

Figure 4.13 Modélisation pour le calcul de la capacité d'une cellule (Modelica).

```
762 block CellCalculator "Calculator for the
       capacity, SOC, SOH,..."
763   import EES = EES;
764   extends Icons.Block;
765   parameter Real SOCini "Initial state of
         charge";
766   parameter Modelica.SIunits.Resistance
        Z0 "Sum of all initial ohmic
        impedances Rs0+Rd0[1]+...+Rd0[ns]";
767   parameter Battery.EES.CellRecords.
        Components.ChargeCapacity capacity
        "Charge capacity";
768   parameter Battery.EES.CellRecords.
        Components.SOH SoH "State of health
         relevant parameters";
769   Modelica.Blocks.Interfaces.RealInput i;
770   Modelica.Blocks.Interfaces.RealOutput
        SOC;
771   Modelica.Blocks.Interfaces.RealOutput
        SOH;
772   Modelica.Blocks.Interfaces.RealOutput C
         "Connector of Real output signal";
773   Modelica.Blocks.Interfaces.RealOutput
        cycles "Connector of Real output
        signal";
774   Battery.EES.Batteries.Components.
        Calculators.SOC sOC(SOCini = SOCini
        ) ;
775   Battery.EES.Batteries.Components.
        Calculators.Capacity c(capacity =
        capacity);
776   Battery.EES.Batteries.Components.
        Calculators.SOHSOS sOH(C0 =
        capacity.C0, Z0 = Z0, SoH = SoH) ;
777   Battery.EES.Batteries.Components.
        Calculators.Qabs q(Qini = capacity.
        aging.Qini);
778   Battery.EES.Batteries.Components.
        Calculators.Cycles EquivalentCycles
        (C0 = capacity.C0, Qini = capacity.
        aging.Qini) ;
779   Modelica.Blocks.Interfaces.RealInput t
        ;
780   Modelica.Blocks.Interfaces.RealInput T
        ;
781   Modelica.Blocks.Interfaces.RealInput Z
        "Connector of Real input signal";
782 equation
783   connect(sOC.SOC, SOC) ;
784   connect(i, q.i) ;
785   connect(i, sOC.i) ;
786   connect(EquivalentCycles.cycles, cycles
        ) ;
787   connect(q.Qabs, c.Qabs) ;
788   connect(t, c.t) ;
789   connect(sOH.SOH, SOH);
790   connect(i, EquivalentCycles.i) ;
791   connect(T, c.T) ;
792   connect(sOH.Z, Z);
793   connect(c.C, C) ;
794   connect(c.C, sOH.C);
795   connect(c.C, EquivalentCycles.C) ;
796   connect(c.C, sOC.C) ;
797 end CellCalculator;
```

Figure 4.14 Modélisation des données structurelles d'une cellule (Modelica).

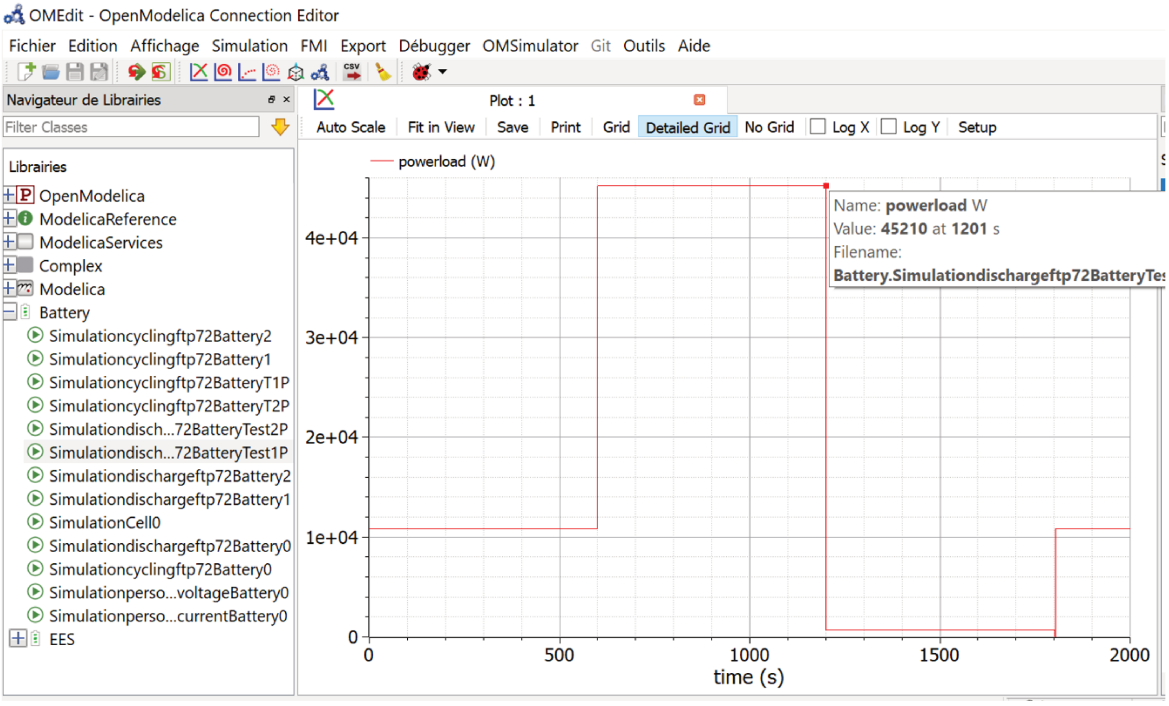
```
798 record StaticResistanceParameters "Basic
       parameter record for StaticResistance
        models"
799   parameter Battery.EES.CellRecords.
        Components.SOCOCV SOCOCV "SOC vs
        OCV curve relevant parameters";
800   parameter Battery.EES.CellRecords.
        Components.ChargeCapacity capacity
        "Charge capacity";
801   parameter Battery.EES.CellRecords.
        Components.Resistance r "Resistance
         parameters";
802   parameter Battery.EES.CellRecords.
        Components.SOH SoH "State of health
         relevant parameters";
803   constant Real n = 1.1 " Peukert number
        : n = 1.1 for Li-ion cells";
804   parameter Modelica.SIunits.Mass mass(
        start = 0.5) " Mass of the cell ";
805   constant Integer catalognum(min = 0) "
        catalog number of the cell ";
806 end StaticResistanceParameters;
```

La structure de données `StaticResistanceParameters` réutilise un modèle de la librairie EES, pour pouvoir spécifier entre autres la masse, la résistance interne, la capacité nominale, le type (`catalognum`) et la relation entre le SOC et l'OCV, via la table de valeurs `SOCOCV` pour une cellule. Un modèle pour une cellule de type 0 est présentée Fig. 4.15.

Figure 4.15 Modélisation d'une cellule de type 0 (Modelica).

```
record CellType0
  extends Battery.EES.CellRecords.
      StaticResistance.
      StaticResistanceParameters(SOCOCV =
       Battery.EES.CellRecords.Components
      .SOCOCV(OCVtableOnFile = false,
      OCVtable = [0.2, 2.7; 0.25,
      2.79375; 0.3, 2.8875; 0.35,
      2.98125; 0.4, 3.075; 0.45, 3.16875;
       0.5, 3.2625; 0.55, 3.35625; 0.6,
      3.45; 0.65, 3.54375; 0.7, 3.6375;
      0.75, 3.73125; 0.8, 3.825; 0.85,
      3.91875; 0.9, 4.0125; 0.95,
      4.10625; 1, 4.2]), capacity =
      Battery.EES.CellRecords.Components.
      ChargeCapacity(C0 = 39 * 3600), r =
       Battery.EES.CellRecords.Components
      .Resistance(R0 = 0.007), mass =
      1.05, catalognum = 0);
end CellType0;
```

Figure 4.16 Analyse-simulation avec le profil de mission.



4.4.4 Modélisation du profil de mission

Le profil de mission (cf. Fig. 4.16) est une instance d'un modèle `Load` (cf. Fig. 4.17) paramétré par les valeurs de puissances demandées par la charge et à partir de la tension mesurée à ses bornes. Le courant de décharge est calculé et imposé à la batterie via une source de courant.

4.4.5 Modèle de simulation

Partie structurelle complètement définie Le modèle de simulation d'une batterie Li-ion définie est présenté Fig. 4.18 et est expliqué ci-après. Il s'agit d'un modèle au type de cel-

Figure 4.17 Modèle pour un profil de mission (EES - Modelica).

```
model Load
  Modelica.Electrical.Analog.Sources.SignalCurrent signalCurrent ;
  Modelica.Electrical.Analog.Interfaces.PositivePin pin_p;
  Modelica.Electrical.Analog.Interfaces.NegativePin pin_n;
  Modelica.Blocks.Interfaces.BooleanInput on "Connector of Boolean input signal" ;
  Modelica.Electrical.Analog.Sensors.VoltageSensor voltageSensor ;
  Modelica.Blocks.Math.Division division ;
  parameter Real table[:, 2] = [0, 0; 1, 0; (..) 1334, 0; 1335, 0; 1336, 0; 1337, 0; (..) 1367, -2.41891E-10; 1368, 0; 1369, 0; 1370, 0] "Load power table (time = first column; power in kW= second columns)";
  Components.Load_Power Load_Power(gain = gain, table = table);
  parameter Modelica.Blocks.Interfaces.RealOutput gain = 1 "Gain";
equation
  connect(signalCurrent.p, pin_p) ;
  connect(signalCurrent.n, pin_n) ;
  connect(pin_p, voltageSensor.p) ;
  connect(voltageSensor.n, pin_n) ;
  connect(voltageSensor.v, division.u2) ;
  connect(Load_Power.y, division.u1) ;
  connect(on, Load_Power.on) ;
  connect(division.y, signalCurrent.i) ;
end Load;
```

lules fixé paramétré à chaque analyse-simulation par `ns` et `np` (lignes 831 - 832). Il a donc fallu créer deux autres modèles similaires pour les deux autres types de batteries (0 et 2). Les paramètres de sorties du modèle sont les performances à mesurer : `ibat`, `vbat`, `SOC`, `icel`, `copperlossbat`, `powerload`, `avenergybat`, `efficiencybat` et d'autres permettant d'avoir une vue globale sur l'architecture tels que `rbat` et `cbat` (lignes 833 à 844).

Ensuite l'architecture de batterie est renseignée, en particulier le type de cellule (ici 1) et le SOC initial (ligne 846). D'autres instances de modèles de systèmes définis (capteurs) et une tension seuil sont également spécifiés pour resp. effectuer les mesures et arrêter la simulation dès que la valeur de tension à vide minimale `ocvmin` est atteinte, ce qui est nécessaire si on souhaite analyser-simuler l'architecture de batterie sur plusieurs profils de mission à la suite. Le profil de mission discrétisé est spécifié ligne 851.

Partie comportementale Les différentes instances de modèles sont interconnectées dans la section `equation`. En particulier, nous avons défini deux assertions d'avertissement relatives aux exigences sur le SOC et le courant `icel` maximal qui indiquent si les contraintes n'ont pas été respectées (lignes 866 et 866).

4.4.6 Architectures de batteries à tester issues de la synthèse

Quatre architectures de batteries obtenues par synthèse et présentées Tab. 4.2 ont été simulées dans le cadre de l'analyse-simulation avec le modèle de simulation. Les valeurs en gras sont celles proche de la valeur limite fixée dans la spécification des exigences. L'ob-

Figure 4.18 Modèle de décharge pour une batterie définie (Modelica).

```
model
      SimulationdischargeelprofileBattery1
parameter Integer ns = 105;
parameter Integer np = 5;
output Modelica.SIunits.Current ibat =
      powerSensor.currentSensor.i;
output Modelica.SIunits.Voltage vbat =
      voltageSensor.v;
output Real SOC = battery.cellCalculator.
      SOC;
output Modelica.SIunits.Voltage ocvbat =
      battery.Uo.v;
output Modelica.SIunits.Resistance rbat =
       battery.r.R_ref;
output Modelica.SIunits.Mass mbat =
      battery.cellParameters.mass * ns * np
      ;
output Modelica.SIunits.Power powerload =
       powerSensor.power;
output Modelica.SIunits.Current icel =
      battery.cellCalculator.i;
output Modelica.SIunits.Power
      copperlossbat = battery.r.LossPower;
output Modelica.SIunits.Energy
      avenergybat = energyCounter.energy;
output Modelica.SIunits.ElectricCharge
      cbat = battery.cellCalculator.C * np;
output Real efficiencybat = vbat*ibat/(
      vbat*ibat + copperlossbat);
Modelica.Electrical.Analog.Basic.Ground
      ground ;
Battery.EES.Batteries.Stacks.Basic.
      StaticResistanceScaled battery(final
      ns = ns, final np = np,
      cellParameters = Battery.EES.
      CellRecords.StaticResistance.
      CellType1(), SOCini = 1) ;
Modelica.Electrical.Analog.Sensors.
      VoltageSensor voltageSensor ;
Modelica.Blocks.Logical.GreaterThreshold
      greaterThreshold(threshold = 2.7 * ns
      ) ;
Modelica.Electrical.Analog.Sensors.
      PowerSensor powerSensor ;
Battery.EES.Sensors.EnergyCounter
      energyCounter ;
Battery.EES.Sources.Loads.Cycles.FTP72
      elprofile(gain = 1, table = [0,
      10.831; (..) ; 599, 10.831; 600,
      10.831; 601, 45.210; (..) 1201,
      45.210; 1202, 0.740; 1203, 0.740;
      (..) 1804, 0]) " load : time in s,
      power in kW" ;
      equation(*@
connect(greaterThreshold.y, elprofile.on)
       ;
connect(elprofile.pin_n, ground.p) ;
connect(elprofile.pin_p, energyCounter.
      pin_nc) ;
connect(powerSensor.nc, energyCounter.
      pin_pc) ;
connect(energyCounter.pin_nv,
      voltageSensor.n) ;
connect(energyCounter.pin_pv,
      voltageSensor.p) ;
connect(powerSensor.pc, battery.pin_p) ;
connect(powerSensor.nv, voltageSensor.n)
      ;
connect(powerSensor.pv, voltageSensor.p)
      ;
connect(voltageSensor.v, greaterThreshold
      .u) ;
connect(voltageSensor.n, battery.pin_n) ;
connect(voltageSensor.p, battery.pin_p) ;
connect(battery.pin_n, ground.p) ;
assert(SOC >= 0.2, " requirement : cell
      soc should be upper than 20% of the
      rated capacity ", AssertionLevel.
      warning);
assert(icel <= 34, " requirement : cell
      current should be lower than 34A ",
      AssertionLevel.warning);
end SimulationdischargeelprofileBattery1;
```

Table 4.2 Architectures de batteries issues de la synthèse à évaluer.

Var.	Exig.	Sol. 1	Sol. 2	Sol. 3	Sol. 4
type	E6	0	2	0	0
np	E8	5	9	5	3
ns	E7	118	129	114	119
mbat	E9	619.5	**789.5**	598.5	374.8
icel	E10	20.4	17.3	21.8	**34.0**
ibat	E1	**102**	**103**	**103**	**107**
soc	E3	0.738	0.45	0.6	0.75
efficiencybat	E12	0.96	**0.9**	0.99	0.95

jectif est ici de confirmer ou d'infirmer l'hypothèse relative à la perte d'admissibilité d'une achitecture basée à partir d'une analyse des valeurs les plus critiques.

En particulier `ibat`, `icel`, `efficiencybat`, `SOC` sont directement impactées par le raffinement du comportement. La masse qui n'est pas directement impactée peut être intéressante à analyser aux limites. On peut noter que les exigences portant sur les nombres de cellules maximales sont de loin respectées, ainsi que pour icel, et ce pour les 4 architectures. On remarque en particulier que pour la Sol. 1, c'est le courant `ibat` qui est le plus proche de la valeur limite (voire égal). D'après nos hypothèses et le raffinement du modèle, cette architecture pourrait éventuellement perdre son admissibilité. Pour la Sol. 2, quatre Variables ont des valeurs à la limite d'exigences : `mbat`, `efficiencybat`, `soc` et `ibat`. Pour les Sol. 3 et 4, seul `ibat` est à surveiller.

Etude des résultats des analyse-simulations

Nous pouvons noter que pour les 4 architectures sélectionnées, on observe un pic de courant pour `ibat`sur la fin du deuxième profil de mission élémentaire, qui correspond à la plus forte contrainte de conception sur cette Variable de comportement (`ibat` supérieur ou égal à 102 *A*).

Pour les Sol. 1 et 2, la valeur du courant `ibat`fourni par la batterie n'est plus consistante avec le modèle grossier affiné. Elle est en réalité inférieure à 102A (resp. 98.99 - cf.Fig. 4.20 et 94.3A). En conséquent, les *SOC* respectifs sont meilleurs que prévu (cf. Fig. 4.21 pour la solution Sol. 2). La Sol. 2 qui semblait la plus défavorable (batterie plus lourde, cellule de type 2, et exigences aux limites pour plusieurs Variables) apparait comme telle parmi les 4 architectures testées. On peut ainsi penser qu'il vaut mieux éviter de sélectionner en priorité une architecture de batterie qui peine à fournir le courant demandé par la charge.

Pour la Sol. 4, c'est en revanche la contrainte de conception sur `icel` qui n'est plus respectée, ce qui rend cette architecture non admissible (ce n'est a priori pas une contrainte, que l'on pourrait relaxer) alors qu'elle est respectée pour toutes les autres architectures (cf.Fig. 4.19).

Etrangement, c'est la plus petite architecture (Sol. 3), qui est aussi la deuxième plus légère, qui reste admissible. Il est assez difficile d'interprêter ce résultat car la valeur de `ibat` est identique à celle de la Sol. 2 qui n'est plus admissible. Il est possible que d'autres paramètres aux valeurs non critiques rentrent en jeu comme le `type`, `ns` ou `np`.

Sol. 4 : n'est plus admissible (icel) Sur ces 4 architectures admissibles de batterie, l'hypothèse qui proposait la perte d'admissibilité en raison de valeurs consistantes trop

Figure 4.19 Sol. 1 : icelmax = 19.79*A*

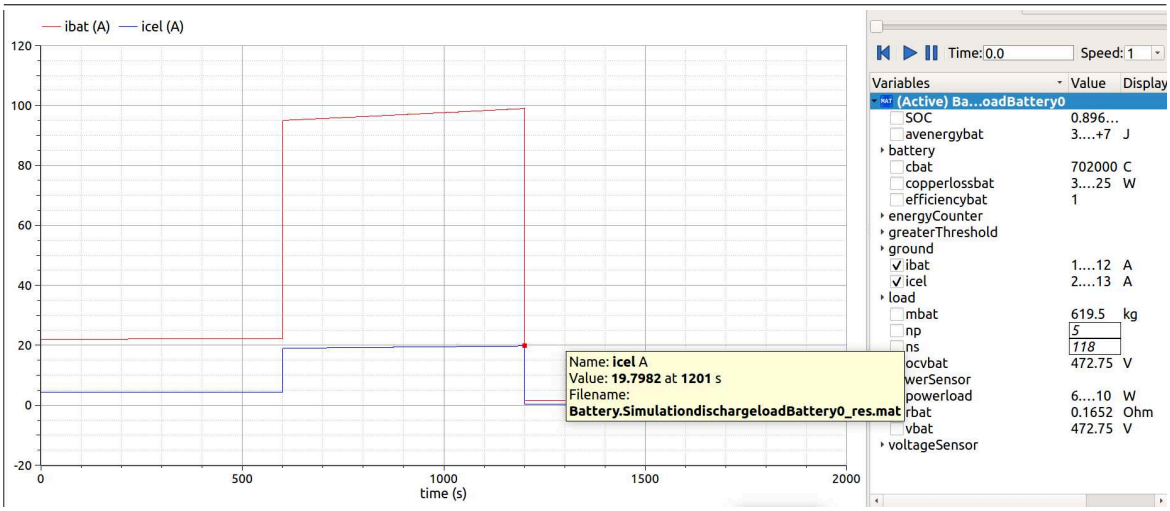
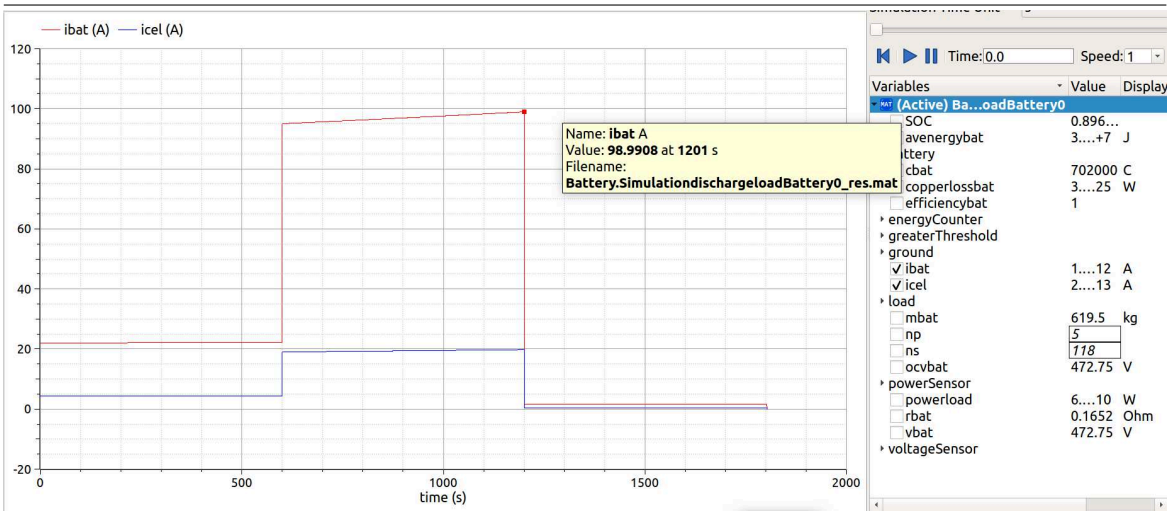
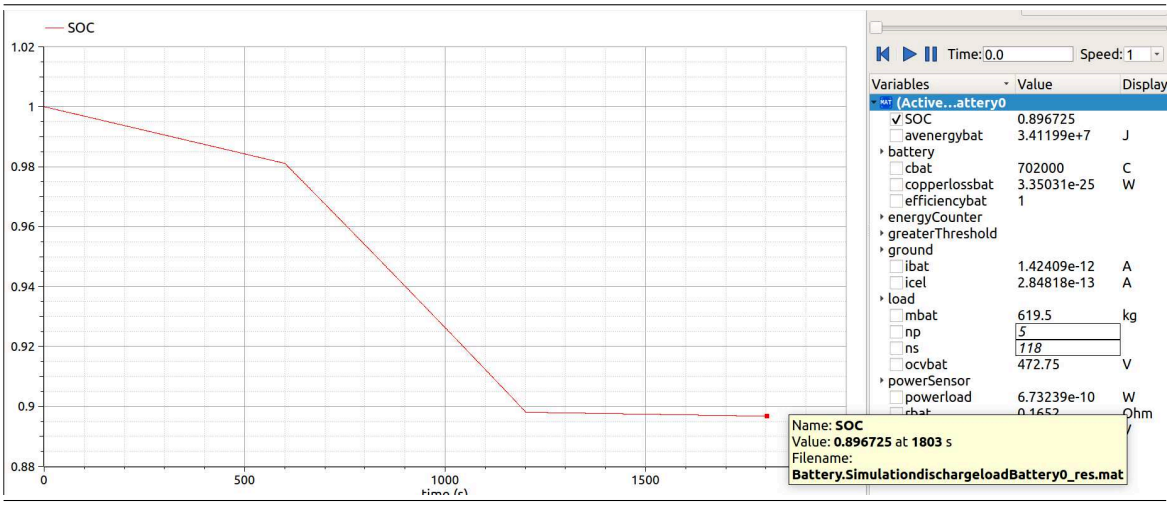


Figure 4.20 Sol. 1 : ibatmax = 98.99*A*



proches des exigences semble correcte (sous réserve d'effectuer plus de tests, et sur d'autres types de batteries, comme l'architecture de batterie 1). Celle-ci est validé au sein de notre approche générale. La contrainte la plus problématique, et qu'il faut essayer de respecter au mieux, semble être celle imposant le courant de décharge de la batterie, puis le courant de décharge d'une cellule. En conclusion, nous pouvons confirmer que l'étude des architectures admissibles (step 3) peut être envisagée pour la sélection d'architectures. Dans la mesure du possible, leurs valeurs "critiques" seront les plus éloignées (au regard de la spécification des exigences). (on pourrait par exemple choisir, pour un domaine continu, sa valeur médiane). Cet exemple a ainsi montré sur quels critères choisir une architecture mais également comment la valider ou non pas simulation.

Figure 4.21 Sol. 1 : socmin = 0.89



4.5 Conclusion

Pour répondre au dernier verrou, dans ce chapitre nous avons présenté une approche outillée et fonctionnelle à base de modèles, qui se veut suffisamment générale pour la conception des systèmes physiques complexes, et qui a été appliquée pour le GE. Celle-ci devrait être pratique à employer pour des concepteurs métiers. En particulier, nous avons pu compléter le processus de synthèse par celui d'analyse des architectures pour valider ou non des solutions obtenues lors de la synthèse et ainsi permettre un meilleur filtrage. L'exemple illustratif a également permis de valider une des hypothèses liée à la perte d'admissibilité d'une architecture. Les limites principales identifiées concernent la non-possibilité d'évaluer des exigences non-fonctionnelles de type 2 avec l'outil/formalisme choisi pour l'analyse-simulation (Modelica).

Conclusion générale

Ces travaux de thèse ont proposé une approche intégrée et fonctionnelle pour la préconception de systèmes physiques, en particulier pour le Génie Electrique (GE). Cette approche orientée problème cherchait à proposer une manière générale de concevoir un système physique en intégrant de manière complémentaire le processus de synthèse de système ainsi que celui d'analyse d'architectures lorsque les modèles disponibles sont de type boîte blanche et gros grains. Les problèmes de conception auxquels s'adressent ces travaux sont pour des systèmes structurés, décomposables, non-linéaires, avec choix de composants depuis catalogues et dont la configuration et/ou les dimensions sont à déterminer selon un profil énergétique.

Au Chapitre 2, nous avons pu voir que les techniques de Programmation Par Contraintes (PPC) étaient appropriées pour résoudre un problème de conception formulé comme un Constraint Satisfaction Problem (CSP), bien que peu d'outils traitent de problèmes mixtes (le solveur Constraint Explorer (CE) utilisé dans ces travaux de thèse est une exception). Pour la formalisation, la Programmation Par Contraintes (PPC) permet de résoudre la problématique d'inversion des modèles mathématiques directs grâce à son aspect déclaratif. Les trois cas d'étude de conception de systèmes électriques présentés ont montré les difficultés de décrire un problème de conception du point de vue de l'ingénierie et non de manière purement mathématique. Les choix de degré de liberté (ddl) et la taille du problème considéré impactent aussi bien la lisibilité d'un modèle de synthèse construit que sa résolution. Les problèmes de génération d'architectures peuvent être spécifiés sous réserve d'utiliser les bons outils et formalismes (comme CE). La conception par optimisation est également possible avec la PPC mais est à réaliser manuellement avec les outils actuellement disponibles. Nous avons pu voir qu'une représentation algébrique des systèmes était adaptée pour obtenir des résultats en des temps corrects tout en conservant une vision claire du système. La formulation CSP n'est cependant pas idéale pour les concepteurs métiers car elle manque de sémantique. On ne peut pas y déclarer de variables déduites. Les formalismes l'implémantant ne permettent

pas non plus de construire des modèles de synthèse réutilisables. Il fallait donc trouver un formalisme déclaratif permettant de construire des modèles de synthèse réutilisables.

Au Chapitre 3, nous avons vu que la modélisation orientée-objet peut permettre d'obtenir un meilleur niveau d'abstraction pour les modèles de synthèse et de les rendre réutilisables. Il fallait cependant proposer une approche innovante de construction de modèles pour l'ingénieur.e intégrant les notions de variabilité nécessaire à la formalisation des problèmes de conception de systèmes physiques. Le formalisme DEsign Problem Specification (DEPS) a été utilisé pour illustrer l'approche pour la synthèse de système. L'approche illustrée propose 3 niveaux de formalisation pour spécifier une contrainte de configuration pour un système, notamment pour un choix de composants dans un catalogue. En particulier, nous avons montré comment étendre des modèles de synthèse existants d'un problème aux exigences fonctionnelles et d'intégration / performances avec des exigences non-fonctionnelles comme environnementales, plus difficiles à traiter. La résolution a également été abordée pour chaque modélisation avec succès avec DEPS Studio et CE. Le processus de synthèse a permis d'obtenir un ensemble d'architectures admissibles pour l'exemple de la batterie de VE. Il faut tout de même disposer de l'outil DEPS Studio pour la mise en pratique de l'approche de synthèse, qui n'est pas (encore) accessible publiquement.

Au Chapitre 4, nous avons exploré l'extension de l'approche de conception par synthèse en incluant a posteriori un processus d'analyse. Avec l'analyse, la conception d'un système peut être raffinée et les architectures réellement admissibles filtrée d'avantage en utilisant des modèles dynamiques algébro-différentiels (voire differentiels). Les langages pris pour exemples sont SysML pour communiquer une architecture et Modelica pour raffiner son comportement et valider ou non l'architecture par simulation. Les limites de l'approche ont été abordées dans le cas où il les concepteurs ne disposaient pas ou plus de solutions admissibles après la synthèse ou après l'analyse. Cet exemple a montré sur quels critères choisir une architecture mais également comment la valider ou non.

En résumé, la principale contribution scientifique de ces travaux de thèse a été de proposer une approche de pré-conception dans le cadre du Model-Based System Engineering (MBSE) pour le GE. Cette approche inclue deux processus distincts. Le premier que nous pouvons qualifier de "synthèse de conception" (qui correspond aux apports décrits dans les Chapitres 2 et 3) s'intéresse principalement à la formalisation et la résolution par PPC d'un problème de conception. Le deuxième que nous pouvons qualifier de validation de conception (qui

correspond aux apports décrits dans le Chapitre 4) s'intéresse aux étapes d'analyse suivant la synthèse. Ces étapes permettent de vérifier et valider la conception par synthèse puis éventuellement de conduire à des choix entre plusieurs solutions si plusieurs restent admissibles (la PPC génère en général non pas une seule mais un ensemble de solutions admissibles). Ce deuxième processus repose essentiellement sur des outils d'analyse par simulation qui permettent alors de raffiner les modèles utilisés et d'explorer des phénomènes non pris en compte durant la première phase de synthèse (par exemple par prise en compte de l'évolution temporelle, de la LCA, d'utilisation de modèles plus riches et plus complets, etc). Comme toute conception réelle, il est assez rare qu'une conception soit "réussie" du premier coup ce qui impliquera que des itérations de conception devront aussi être envisagées. Ces itérations ont lieu autour de la synthèse si il n'y a pas de solution admissible et de l'analyse vers la synthèse si les solutions initialement admissible perdent leur admissibilité lors d'un raffinement de modèle ou de la prise en compte de nouveaux phénomènes lors des phases de validation et d'analyse. Ces itérations peuvent conduire à une amélioration des modèles ou à une révision des exigences de conception suivants les indications (ou les pistes) que nous aurons apporté les phases de validation. L'ensemble de la démarche a été implémenté à l'aide d'outils suffisamment haut niveau pour être accessibles à des concepteurs métiers au sein de bureaux d'études industriels.

Perspectives

Nous proposons ci-après des perspectives d'amélioration de l'approche proposée dans ce manuscrit.

Couplage de la formalisation avec l'outil d'analyse causale

Un outil d'analyse causale analogue à celui présent dans CE pourrait être intégré dans DEPS Studio au sein de l'approche durant la formalisation des problèmes. Un tel outil serait d'une grande aide pour les concepteurs dans les choix des degré de liberté (ddl) et la détection de problèmes sur-contraints.

Enrichissement des bibliothèques de modèles de synthèse

Les modèles de synthèse présentés à titre d'exemple dans ce manuscrit représentent les modèles électriques des systèmes électriques. En Génie Electrique (GE), la conception inclut généralement de considérer le comportement d'un système selon l'aspect mécanique,

thermique ou magnétique. Des bibliothèques additionnelles de modèles de synthèse DEPS peuvent être construite de manière similaires à ceux présentés Chapitre 3 pour représenter des terminaux physiques et interconnexions autre qu'électriques. L'enrichissement des bibliothèques permettrait sur le long terme de vrais gains sur les coûts de modélisation.

Complétion de la modélisation du problème d'une chaîne de traction

A la suite de ces travaux de thèse, une perspective serait de compléter la conception de la chaîne de traction d'un Véhicule Electrique (VE) à travers une intégration des autres sous-systèmes de la chaîne étape par étape comme explorée au Chapitre 3.

Extension de la synthèse à l'optimisation

Au Chapitre 3, nous avons étudié la recherche d'architectures admissibles avec DEPS et DEPS Studio. Ces travaux de thèse s'interessent avant tout à la conception plutôt qu'à l'optimisation. Cependant, la synthèse ayant été validée, nous pouvons envisager l'ajout d'une optimisation. Cette piste peut être envisager de deux manières.

Comme au Chapitre 2 avec CE, on peut envisager avec DEPS d'implémenter une optimisation mono-, bi- ou tri-critères par intérations. Pour l'optimisation bi- et tri-critères, on emploirait à nouveau les techniques de Marglin ou des facteurs de pondération. Dans ce cas, nous devons contraindre de plus en plus une des exigences jusqu'à trouver l'optimum. L'avantage de cette optimisation est de pouvoir facilement et rapidement optimiser l'architecture d'un système. L'étape la plus compliquée a déjà été réalisée pendant la résolution (la recherche de solutions). L'inconvénient est tout de même de devoir réaliser l'optimisation à la main avec les outils actuellement disponibles.

Une autre possibilité serait de selectionner des architectures admissibles et de les améliorer/optimiser selon une exigence choisie. Cette seconde option peut paraître moins intéressante que la précédente, qui elle propose directement une conception optimale. On peut tout de même y voir un avantage potentiel. Les techniques d'optimisation actuelles pourraient se servir d'une architecture qui au moins sera correcte par construction. Les concepteurs n'auraient donc pas à travailler avec des solutions non admissibles, qui n'auraient fournies aucun résultats à l'issue de l'optimisation.

Une piste pour cette étude supplémentaire peut être de réaliser une analyse des compromis. Dans cette analyse de compromis, on s'interesserait aux deux exigences les plus

critiques (un ordre de priorité peut être spécifié pour les exigences lors de l'analyse des besoins). Les exigences choisies peuvent être de type non-fonctionnelles 1 (une d'intégration et l'autre de performances) et/ou de type 2. Le choix d'exigences de type 2 pour représenter ce Front de Pareto pourrait être intéressant. En particulier si les outils d'analyse choisis ne permettent pas d'évaluer les exigences de type 2. Cette analyse de compromis se terminerait par la selection d'architectures respectant au mieux les deux exigences (comme on s'interesserait aux contraintes d'inégalité, les valeurs de Variables les plus éloignées de la valeur limite).

Génération automatique de modèles d'analyse

Nous pouvons envisager une automatisation des étapes pour l'instant réalisée à la main. Dans l'approche, 3 types de modèles sont construits, avec 3 langages : DEPS, SysML et Modelica. On pourrait alors envisager la génération d'un modèle SysML de l'architecture depuis les résultats synthétisés avec DEPS/DEPS Studio. Les travaux de recherches existant sur ModelicaML visent à convertir des modèles SysML et Modelica entre eux et pourraient éventuellement être exploités [199–202].

ModelicaML et extension de Modelica à la LCA et autres exigences

Pour la prise en compte des exigences non-fonctionnelles de type 2 avec Modelica, il faudrait étendre les librairie Modelica. Une des difficultés serait d'obtenir les données d'entrées pour l'évaluation de ces exigences, généralement détenues par les industriels. Une seconde perspective pour la gestion de ce type d'exigences serait de coupler les outils de simulation Modelica avec les outils métiers addressés (ex : l'outil GREET [196] pour la Life-Cycle Assessment (LCA) des batteries Li-ion). Mais dans la pratique, disposer de l'ensemble de ces outils d'évaluation semble peu réaliste (il en faudrait un pour la sûreté de fonctionnement, pour LCA, pour le coût, etc.). Le couplage CSP/LCA proposé dans le benchmark [76] évoqué au Chapitre 3 pourrait être une piste prometteuse également.

Extension de l'approche à la conception détaillée

L'approche proposée dans ce manuscrit s'intéresse principalement à la conception préliminaire mais peut également s'étendre à la phase de conception détaillée et ainsi couvrir la conception entière. Il s'agirait alors de se reposer sur des modèles grains fins pour une analyse-simulation encore plus fine des architectures admissibles ainsi que sur des outils et

formalismes adéquats (par ex. la suite d'outils Abaqus FEA destinée entre autre à l'analyse EF).

Modèle de type boîte noire et analyse de sensibilité

Une autre perspective concerne l'utilisation de modèle de type boîte noire en analyse et le cas de perte d'admissibilité de toutes les architecture après analyse. Dans ce cas, on peut par exemple envisager une analyse de sensibilité [203] en faisant varier les variables en entrée, déterminer lesquelles sont les plus influentes sur les performances du système et modifier (relaxer ou contraindre d'avantage) des exigences.

Utilisation de l'approche dans le cadre de la norme ISO/IEEE 42020

Récemment, la norme ISO/IEEE 42020 [204] a été proposée pour offrir un processus de conception plus centré sur la conceptualisation des architectures, c'est-à-dire la synthèse de systèmes. Pour s'appuyer sur cette norme, il faut pouvoir utiliser des modèles de synthèse formels comme proposés dans ces travaux de thèse. Les travaux présentés dans ce manuscrit se positionnent donc de manière complémentaire à la norme ISO/IEEE 42020.

La difficulté principale est qu'il n'existe pas encore de lignes directrices formellement documentées en conception de systèmes physiques pour formaliser de manière réutilisable et à un haut niveau d'abstraction leurs problèmes de conception, car c'est un domaine encore peu exploré. Une piste de développement dans la suite de ces travaux pourrait être de réaliser des *design pattern* [205] comme il en existe déjà en Ingénierie des Logiciels (IL). Les *design pattern* s'intéresse aux méthodologies de construction des modèles orientés-objets réutilisables pour des problèmes de conception récurrents afin d'aider les concepteurs dans cette tâche difficile.

Annexe A

Conception d'une machine DC brushless à aimants permanents sans encoches

A.1 Modèle gros grains algébrique [24]

$$C_{em} = \frac{\pi}{2(D/L)}(1 - K_f)\sqrt{k_f \beta E_{ch} E} D^2 (D + E) B_e \tag{A.1}$$

$$K_f \simeq 1.5 p \beta \frac{e + E}{D} \tag{A.2}$$

$$C = \frac{\pi \beta B_e}{4 p B_{iron}} D \tag{A.3}$$

$$D = \frac{p \Delta p}{\pi} \tag{A.4}$$

$$E_{ch} = k_r * E * J_{co} \tag{A.5}$$

$$B_e = \frac{2 l_a P}{D \log(\frac{D+2E}{D-2(l_a+e)})} \tag{A.6}$$

$$K_f \leq K_{fmax} \tag{A.7}$$

$$K_f \geq K_{fmin} \tag{A.8}$$

$$K_{fmin} = 0.01 \tag{A.9}$$

$$K_{fmax} = 0.5 \tag{A.10}$$

$$k_f = 0.70 \tag{A.11}$$

$$P = 0.90T \tag{A.12}$$

$$B_{iron} = 1.5T \tag{A.13}$$

$$E_{ch} = 10^{11} A^2/m^3 \tag{A.14}$$

$$\Delta p = 0.100m \tag{A.15}$$

$$C_{em} = 10Nm \tag{A.16}$$

$$C_{em} \geq 10Nm \tag{A.17}$$

$$V_u = \pi L(D + E - e - l_a)(2C + E + e + l_a) \tag{A.18}$$

$$V_m = \pi \beta l_a L(D - 2a - l_a) \tag{A.19}$$

$$P_j = \pi \rho_{co} L(D + E) E_{ch} \tag{A.20}$$

$$V_a \leq V_{amax} \tag{A.21}$$

$$V_u \leq V_{umax} \tag{A.22}$$

$$P_j \leq P_{jmax} \tag{A.23}$$

$$f_c = \frac{V_u}{V_{u0}} + \frac{V_a}{V_{a0}} + \frac{P_j}{P_{j0}} \tag{A.24}$$

avec : V_{u0}, V_{a0} et P_{j0} les minimums obtenus à partir des optimisations mono-critère, resp. de V_u, V_a et P_j.

Table A.1 Nomenclature des Variables [150, 24].

ddl	Définition	Unités	Types	Domaines
C_{em}	Couple électromagnétique	$N{\cdot}m$	continu	$[10, +\infty]$
β	Coefficient d'arc polaire	/	continu	[0,1.0]
p	Nombre de paires de pôles	/	discret	[|1,10|]
L	Longueur de la machine	m	continu	[0,0.5]
B_e	Induction magnétique à vide dans l'entrefer	T	continu	[0.1,1.0]
E	Epaisseur des bobinages	m	continu	[0.001,0.05]
e	Entrefer	m	continu	[0.1e-4,5e-4]
l_a	Epaisseur d'un aimant	m	continu	[0.003,0.05]
Variables déduites				
K_f	Coefficient de fuites interpolaires	/	continu	$[0, +\infty]$
E_{ch}	Echauffement approximatif de la machine	A^2/m^3	continu	$[0, +\infty]$
k_f	Coefficient de remplissage des bobinages	/	continu	[0, 1]
Δp	Pas polaire	-	continu	$[0, +\infty]$
λ	Facteur de forme de la machine	/	continu	[0, 1]
D	Diamètre d'alésage	m	continu	$[0, +\infty]$
A	Charge linéique	A/m	continu	$[-\infty, +\infty]$
J_{co}	Densité de courant linéique dans le cuivre	A/m^2	continu	$[0, +\infty]$
C	Epaisseur de la culasse statorique	m	continu	$[0, +\infty]$
B_{iron}	Induction maximale dans les laminations en fer	T	continu	$[0, +\infty]$
P	Polarisation magnétique d'un aimant	T	continu	$[0, +\infty]$
ρ_{co}	Résistivité spécifique du cuivre	$\Omega \cdot m$	continu	$[0, +\infty]$
Critères				
V_u	Volume des parties actives	m^3	continu	[0,30]
V_a	Volume des aimants	m^3	continu	[0,50]
P_j	Pertes par effet Joule	W	continu	[0,150]

Annexe B

Conception d'une machine AC/DC brushless à AP

B.1 Modèle gros grains algébrique [25]

$$C_{em} = k_t D(D + \bar{\sigma}_e S(\sigma_r)E)LB_e K_s \tag{B.1}$$

$$K_s = k_r Ej(\sigma_e \frac{a}{a+d} + \bar{\sigma}_e) \tag{B.2}$$

$$k_t = \frac{\pi}{2}(\sigma_f(1 - K_f)\sqrt{\beta} + \bar{\sigma_f}\frac{\sqrt{2}}{2}\sin(\beta\frac{\pi}{2})) \tag{B.3}$$

$$K_f = 1.5p\beta\frac{E+g}{D}\bar{\sigma}_e\sigma_f \tag{B.4}$$

$$B_e = \frac{2J(\sigma_m)l_a}{k_c S(\sigma_r)D\ln\frac{D+2ES(\sigma_r)\bar{\sigma_e}}{D-2S(\sigma_r)(l_a+g)}} \tag{B.5}$$

$$k_c = \frac{1}{1 - \sigma_e\frac{N_e a^2}{5\pi Dg+\pi Da}} \tag{B.6}$$

$$N_e = \pi\frac{D + ES(\sigma_r)}{a+d} \tag{B.7}$$

$$N_e = 2pqm \tag{B.8}$$

$$B_t = B_e\frac{a+d}{d} \tag{B.9}$$

$$B_c = B_e\frac{D}{2pC}(\frac{\pi}{2}\beta\bar{\sigma}_f + \sigma_f) \tag{B.10}$$

$$k_d = \sigma_e\frac{d}{a+d} \tag{B.11}$$

$$\phi_{Th} = \frac{\theta_J - \theta_e}{R_e} + \frac{\theta_J - \theta_c}{R_c} \tag{B.12}$$

$$\phi_{Th} = \rho_{cu}\pi j L K_s (D + \bar{\sigma}_e S(\sigma_r) E) \tag{B.13}$$

$$R_c = \frac{1}{\pi L h_a (D + 2ES(\sigma_r)(E + C))} \tag{B.14}$$

$$R_e = \frac{1}{\pi L h_e D} \tag{B.15}$$

$$B_c \leq B_M(\sigma_{mt}) \tag{B.16}$$

$$\sigma_e * B_t \leq B_M(\sigma_{mt}) \tag{B.17}$$

$$R_{int} = \frac{D}{2} - C - E\bar{\sigma}_r - (g + l_a)\sigma_r \tag{B.18}$$

$$R_{int} \geq C_{R_{int}} \tag{B.19}$$

$$R_{int} \geq (\frac{2}{\pi}\tau_m)^{\frac{3}{2}} \tag{B.20}$$

$$R_{ext} = \frac{D}{2} + C + E\bar{\sigma}_r - (g + l_a)\sigma_r \tag{B.21}$$

$$R_{ext} \leq C_{R_{ext}} \tag{B.22}$$

$$V_m = \beta\pi L l_a (D - (2g + l_a) S(\sigma_r)) \tag{B.23}$$

$$V_c = 2\pi L C (D + (E - g - l_a) S(\sigma_r)) \tag{B.24}$$

$$V_d = \pi L E (D + ES(\sigma_r))((1 - \beta)\sigma_e + \frac{d}{a + d}\bar{\sigma}_e) \tag{B.25}$$

$$V_{co} = k_r \pi L E (D + ES(\sigma_r))(\beta\bar{\sigma}_e + \frac{a}{a + d}\sigma_e) \tag{B.26}$$

$$V_g = \frac{\pi L}{4}(\sigma_r (D + 2(E + C))^2 + \bar{\sigma}_r (D + 2(g + l_a + C))^2) \tag{B.27}$$

$$M_a = V_m \rho_{PM}(\sigma_m) + V_c \rho_{CM}(\sigma_{mt}) + V_d(\rho_{CM}(\sigma_{mt})\sigma_e + \rho_{Al}\bar{\sigma}_e) + V_{co}\rho_{co} \tag{B.28}$$

$$M_a \leq M_{amax} \tag{B.29}$$

$$V_g \leq V_{gmax} \tag{B.30}$$

$$q = 3 \tag{B.31}$$

$$m = 1 \tag{B.32}$$

$$V_g \leq V_{glimite} \tag{B.33}$$

$$min[\rho \cdot (\frac{M_a}{M_{a0}}) + (1-\rho) \cdot \frac{V_g}{V_{g0}}], \tag{B.34}$$

avec $0 \leq \rho \leq 1$ et M_{a0} et V_{g0} les minimums obtenus à partir des optimisations mono-critère, resp. de M_a et V_g. p, σ_e, σ_f, σ_r, σ_m et σ_{mt} sont les seules Variables discrètes.

Table B.1 Nomenclature des Variables.

Constantes	Signification	Unités	Domaines
ρ_{cu}	Résistivité du cuivre	$\Omega \cdot m$	1.8e-8
h_e	Coefficient de convection dans l'entrefer	W/Km^2	12
h_a	Coefficient de convection du milieu ambiant	W/Km^2	4
τ_m	Contrainte de cisaillement	Pa	1e7
ρ_{co}	Densité du cuivre	mm	8900
ρ_{Al}	Densité de l'aluminium	mm	2700
ddl			
σ_e	Type d'armatures (encoches)	/	{0,1}
σ_f	Type de forme d'onde	/	{0,1}
σ_r	Type de configuration rotorique	/	{0,1}
σ_m	Type d'aimants permanents	/	{pl, mo}
σ_{mt}	Type de conducteur magnétique	/	{pd, st}
C_{em}	Couple électromagnétique	$N \cdot m$	10
B_e	Densité de flux magnétique à vide	T	[0.4,0.9]
D	Diamètre d'alésage du stator	mm	[1,300]
E	Epaisseur des bobinages	mm	[4,100]
L	Longueur utile du moteur	mm	[50,150]
k_r	Coefficient de remplissage des bobinages		[0.7,0.7]
j	Densité de courant surfacique	A/mm^2	[3,6]
a	Longueur d'une encoche	mm	[4,+∞]
d	Longueur d'une dent	mm	[4,+∞]
p	Nombre de paires de pôles	/	{1,16}
g	Entrefer mécanique	mm	[1,1]
la	Èpaisseur d'un aimant	mm	[4,100]
q	Nombre de phases	/	[3,3]
m	Nombre d'encoches par pôles par phases	/	[1,1]
Variables déduites			
C	Epaisseur du carter	mm	[4,100]
θ_J	Température des bobinages en cuivre	C	[130,130]
θ_e	Température dans l'entrefer	C	[55,55]
θ_c	Température de l'air ambiant	C	[50,50]
K_s	Densité de courant linéique	kA/m	[0.5,100]
k_d	Rapport de denture	/	[0.4,0.6]
B_c	Densité de flux dans la culasse statorique	T	$[0.9, B_M(\sigma_{mt})]$
B_t	Densité de flux dans les dents	T	$[0.9, B_M(\sigma_{mt})]$
R_{int}	Rayon intérieur	mm	[50,+∞]
R_{ext}	Rayon extérieur	mm	[0,80]
V_g	Volume global	$10^{-3}m^3$	[0, +∞]
V_c	Volume du carter	$10^{-5}m^3$	[0, +∞]
V_t	Volume des dents	$10^{-5}m^3$	[0, +∞]
Critères			
M_a	Masse des parties actives [1]	kg	[0, +∞]
V_{co}	Volume des conducteurs électriques	$10^{-5}m^3$	[0, +∞]
V_a	Volume des aimants	$10^{-5}m^3$	[0, +∞]

Annexe C

Conception d'une batterie Li-ion pour VE

C.1 Modèles gros grains algébrique [22]

$$c_3 = i_3/3 \tag{C.1}$$

$$c_{3_{cel}} = \frac{i_{3_{cel}}}{3} \tag{C.2}$$

$$c_i(t) = c_{3_{cel}} \left(\frac{i_{cel}(t)}{i_{3_{cel}}} \right)^{1-peukert} \tag{C.3}$$

$$c_{i_k} = c_{3_{cel}} \left(\frac{i_{cel_k}}{i_{3_{cel}}} \right)^{1-peukert} \tag{C.4}$$

$$soc(t) = soc_{ini} - \frac{i_{cel}(t)}{c_i(t)} t \tag{C.5}$$

$$\Delta soc_k = \frac{i_{cel_k}}{c_{ik}} \Delta t_k \tag{C.6}$$

$$soc_k = soc_{k-1} + \Delta soc_k \tag{C.7}$$

$$i_{cel}(t) = \frac{i_{bat}(t)}{n_p} \tag{C.8}$$

$$i_{cel_k} = \frac{i_{bat_k}}{n_p} \tag{C.9}$$

$$ocv_{cel}^k(soc_{cel}^k) = \frac{ocv_{cel}^{max} - ocv_{cel}^{min}}{soc_{cel}^{max} - soc_{cel}^{min}} (soc_k^{cel} - 1) + ocv_{cel}^{min} \tag{C.10}$$

$$v_{cel_k} = ocv_{cel_k}(soc_{cel_k}) - r_{cel} * i_{cel_k} \tag{C.11}$$

$$r_{bat} = r_{cel}\frac{n_s}{n_p} \tag{C.12}$$

$$p_j = r_{bat}\left(i_{bat}\right)^2 \tag{C.13}$$

$$ocv_k(soc_{bat_k}) = n_s * ocv_k(soc_{cel_k}) \tag{C.14}$$

$$v_{bat_k} = n_s * v_{cel_k} \tag{C.15}$$

$$p_{tot_k} = p_j + p_{load_k} \tag{C.16}$$

$$p_{bat_k} \geq p_{tot_k} \tag{C.17}$$

$$m_{bat} = m_{cel} * n_s * n_p \tag{C.18}$$

$$c_{bat} = c_3 * n_s * n_p \tag{C.19}$$

$$e_{u_{bat}} = c_{bat} * v_{bat_k} \tag{C.20}$$

$$e_{tot_{bat}} = c_{3_{bat}} * v_{bat_k} \tag{C.21}$$

$$e_{load} = \int_0^n p_{load}(t)\,\mathrm{d}t \tag{C.22}$$

$$e_{load} = \sum_{k=1}^{n} p_{load_k}\Delta t_k \tag{C.23}$$

$$p_{bat_k} = i_{bat_k} * ocv_k(soc_{bat_k}) \tag{C.24}$$

Annexe D

Eco-conception de la batterie Li-ion pour VE

D.1 Modèle gros grains algébrique [76]

$$m_{fuel,bat} = \frac{frv}{10^4} \cdot m_{bat}\rho_{fuel} \cdot D \cdot (1 - uf) \tag{D.1}$$

$$m_{fuel,bat} = \alpha_{fuel} \cdot \frac{m_{bat}}{m_{veh}} \cdot \frac{c_{fuel,veh}}{100} \cdot \rho_{fuel} \cdot D \cdot (1 - uf) \tag{D.2}$$

$$m_{c02,bat} = \tau_{c02} \cdot m_{fuel,bat} \tag{D.3}$$

$$\begin{pmatrix} m_{c0,bat} \\ m_{nox,bat} \\ m_{hc,bat} \end{pmatrix} = \alpha_{fuel} \cdot \frac{m_{bat}}{m_{veh}} \cdot \begin{pmatrix} e_{c0} \\ e_{nox} \\ e_{hc} \end{pmatrix} \cdot D \cdot (1 - uf) \tag{D.4}$$

$$aer = \frac{e_u}{m_{veh} \cdot k_{elec}} \tag{D.5}$$

$$aer = \frac{e_u}{c_{elec,veh}} \tag{D.6}$$

$$e_{elec,bat} = \frac{e_u}{aer} \cdot \left[\frac{\alpha_{elec} \cdot m_{bat}}{\eta_{PtT} \cdot m_v} + (1 - \eta_{ri}) \right] \cdot D \cdot (1 - uf) \tag{D.7}$$

$$e_u^{ref} = \frac{e_u}{m_{bat}} \cdot m_{bat}^{ref} \tag{D.8}$$

$$uf(aer) = 1 - \exp\left(- \sum_{i=1}^{6} C_i \cdot (aer/(1.609344 \cdot 400))^i \right) \tag{D.9}$$

Table D.1 Caractéristiques des carburants.

		frv (L/(100 kg ·100 km))	ρ_{fuel} (kg/L)	α_{fuel} (%)	τ_{c02} ($kg \cdot CO_2/kg$ fuel)
Carburant	diesel	0.12	0.84	30	3.16
	essence	0.15	0.74		3.17

Table D.2 Emissions des polluants CO, NO_x et HC en fonction du carburant.

Emissions des polluants (10^{-6} kg/km)	Carburant	
	gazole	essence
e_{co}	500	1000
e_{nox}	80	60
e_{hc}	90	100

Table D.3 Nomenclature des Variables (continues).

ddl	Définition	Unités	Domaines
uf	Facteur de fonctionnement en électrique	%	[0,100]
Variables déduites			
frv	Coefficient de réduction du carburant	$L/(100kg \cdot 100 \cdot km)$	$[0,+\infty]$
m_{bat}	Masse de la batterie	m	$[0,+\infty]$
D	Distance cycle de vie	km	$[0,+\infty]$
$m_{fuel,bat}$	Masse de carburant relatif au transport de la batterie	kg	$[0,+\infty]$
α_{fuel}	Coefficient relatif à la consommation de carburant	%	[0,100]
m_{veh}	Masse du véhicule	kg	$[0,+\infty]$
$c_{fuel,veh}$	Consommation de carburant du véhicule	$L/100km$	$[0,+\infty]$
ρ_{fuel}	Masse volumique du carburant	kg/L	$[0,+\infty]$
τ_{c02}	Facteur d'émission de CO_2	$kg \cdot CO_2/kg$ fuel	$[0,+\infty]$
$m_{c02,bat}$	Quantité de CO_2 émis durant la combustion	kg	$[0,+\infty]$
$m_{c0,bat}$	Masse du polluant CO	kg	$[0,+\infty]$
$m_{nox,bat}$	Masse du polluant NO_x	kg	$[0,+\infty]$
$m_{hc,bat}$	Masse du polluant HC	kg	$[0,+\infty]$
e_u	Energie utile de la batterie	$kW \cdot h$	$[0,+\infty]$
k_{elec}	Consommation du véhicule	$kW \cdot h/(km \cdot kg)$	$[0,+\infty]$
$c_{elec,veh}$	Consommation électrique du véhicule	$kW \cdot h/km$	$[0,+\infty]$
aer	Mode tout électrique	km	[0,200]
α_{elec}	Coefficient relatif à la consommation électrique	30%	[0,100]
η_{ri}	Rendement interne (batterie)	$kW \cdot h$	$[0,+\infty]$
η_{PtT}	Rendement PtT	$kW \cdot h$	$[0,+\infty]$
e_u^{ref}	Energie utile de la batterie de référence	$kW \cdot h$	$[0,+\infty]$
m_{bat}^{ref}	Masse de la batterie de référence	kg	$[0,+\infty]$

D.2 LCA : objectifs et limites

La méthode d'analyse LCA vise à quantifier et à évaluer les potentiels impacts des produits, des procédés et des activités humaines sur l'environnement [206]. Elle comprend quatre phases, dont la Life-Cycle Inventory (LCI), pour aider à estimer les émissions et les pertes énergétiques associées au flux de matières et d'énergies à chaque étape du cycle de vie d'un système.

La LCA repose sur l'utilisation de modèles de systèmes définis, qui sont construits à partir d'une base de données obtenue durant la LCI [207]. Cette base de données permet d'obtenir des informations sur l'ensemble du cycle de vie d'un système comme sur les procédés employés pendant la fabrication du véhicule. Ces informations constituent les données d'entrées primaires des modèles d'analyse [208].

Il faut répartir les données primaires entre chaque constituant (du véhicule, de la batterie et de ses cellules) et donc supposer une architecture pour sa sa chaîne de traction pour évaluer la part de la batterie dans les impacts environnementaux [188]. Les données relatives à une batterie sont détenues par son constructeur et son fournisseur. Elles peuvent être difficiles à obtenir pour les constructeurs automobiles pour des raisons de confidentialité [75]. Elles doivent souvent être complétées par des données secondaires issues de la littérature, qui résultent généralement d'autres analyses LCA [188].

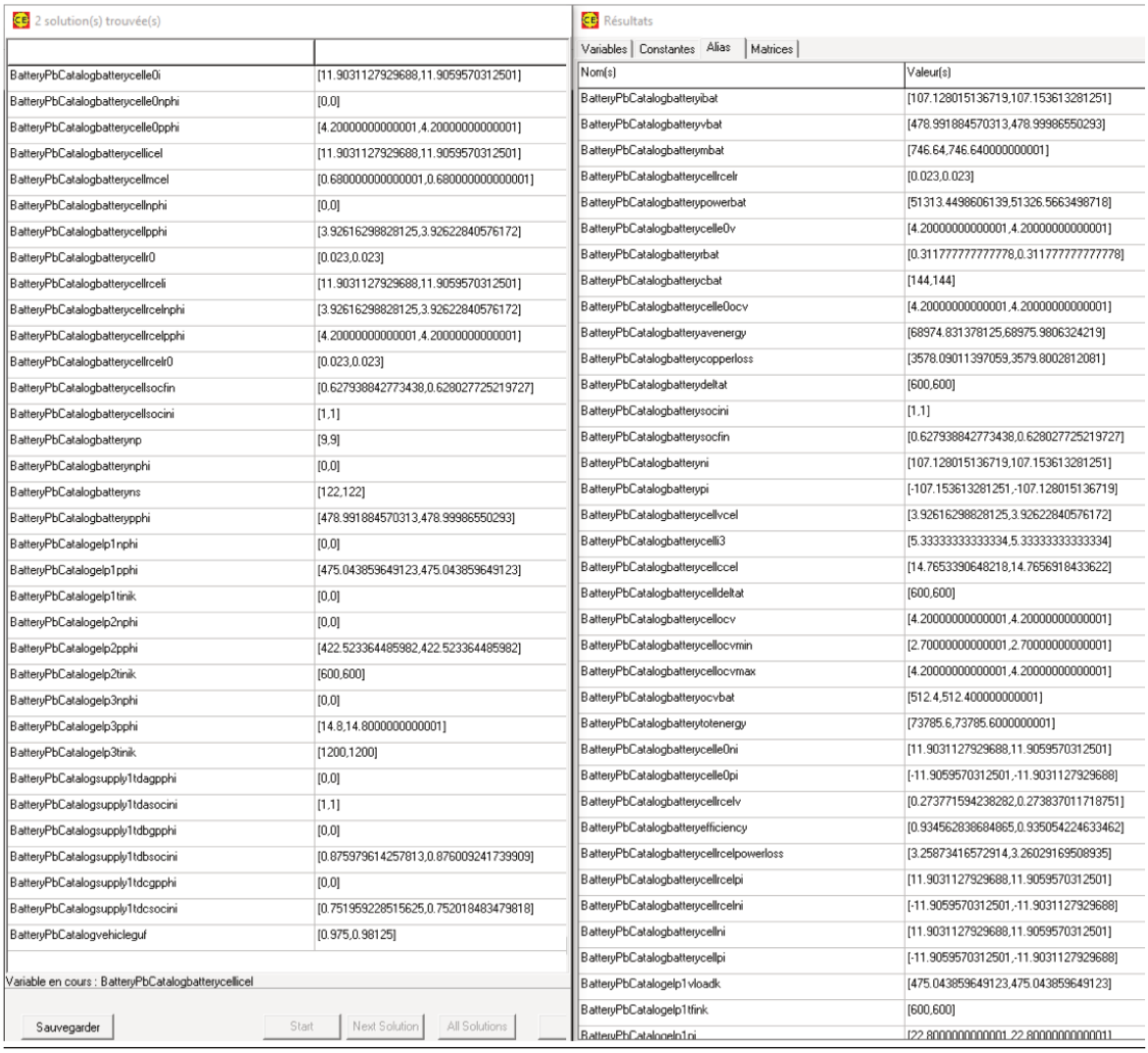
Le résultat de la LCA est très dépendant de la technologie de batterie Li-ion, de celle du véhicule et du mode de production d'électricité choisis pour la simulation. Les plus mauvais résultats sont obtenus pour les centrales à charbon [209]. Dans ce cas, il existe un risque de transfert de pollution entre celle générée lors de l'usage du VE et celle durant la production d'énergie. L'impact environnemental du VE devient mitigé voire défavorable par rapport aux Internal Combustion Engine Vehicle (ICEV) [210].

D.3 Modèles DEPS pour l'éco-conception

Figure D.1 Formalisation du problème d'éco-conception (DEPS).

```
868 Package BatteryPb ;
869 Uses Types, Components ;
870 Problem BatteryPbCatalogEnv
871 Constants
872 (..)
873 efficiencymin : REAL = 0.9 ;
874 gwpmax : REAL = 10000 ;
875 (..)
876 iloadk2 : INTENSITY = 50 ;
877 densityd : DENSITY = 0.12 ;
878 frvd : RATIO = 0.84 ;
879 mveh : MASS = 2100 ;
880 lcd : LENGHT = 150000 ;
881 aer : LENGHT = 175 ;
882 emissioncod : REAL = 500e-6 ;
883 emissionnoxd : REAL = 80e-6 ;
884 emissionhcd : REAL = 90e-6 ;
885 emissionfco2d : FACTOR = 3.16 ;
886 efficiencyptt : EFFICIENCY = 0.9 ;
887 fuelrefgwp : REAL = 0.56 ; (* GWP for producing 1 kg of fuel *)
888 elecrefgwp : REAL = 0.088 ; (* GWP for producing 1 kW.h of
       electricity *)
889 manufrefgwp : REAL = 834.94 ; (* reference battery manufacturing GWP
       *)
890 endofliferefgwp : REAL = 91.08 ; (* reference battery end of life GWP
        *)
891 mbatref : MASS = 200 ; (* reference battery mass *)
892 avenergybatref : ENERGY = 14500 ; (* reference battery useful energy
       *)
893 Variables
894 Elements
895 battery : Battery() ;
896 elp1 : ELProfile(deltat0, ploadk0, iloadk0) ;
897 (..)
898 cod : Polluant(emissioncod) ;  (* CO - diesel *)
899 noxd : Polluant(emissionnoxd) ; (* NO - diesel *)
900 hcd : Polluant(emissionhcd) ;  (* HC - diesel  *)
901 co2d : Polluant(emissionfco2d) ;  (* CO2 - diesel  *)
902 diesel : Fuel(densityd, frvd, cod, noxd, hcd, co2d) ;
903 vehicled : Vehicle(mveh, lcd, aer, diesel, battery) ;
904 powergrid : PowerGrid(efficiencyptt) ;
905 envrefd : EnvironmentalRefProfile(mbatref, avenergybatref,
       manufrefgwp, fuelrefgwp, elecrefgwp, endofliferefgwp) ;
906 environmentd : EnvironmentalBurden(co2d, noxd, cod, hcd, envrefd,
       vehicled, powergrid) ;
907 max1 : MaxValues(battery, nsmax, npmax, mbatmax, icelmax, energymax,
       environmentd, gwpmax) ;
908 min1 : MinValues(battery, efficiencymin) ;
909 Properties
910 End
```

Figure D.2 Une architecture admissible au problème d'éco-conception (carburant essence - CE).



Annexe E

Modélisation du problème de conception de la batterie en SysML

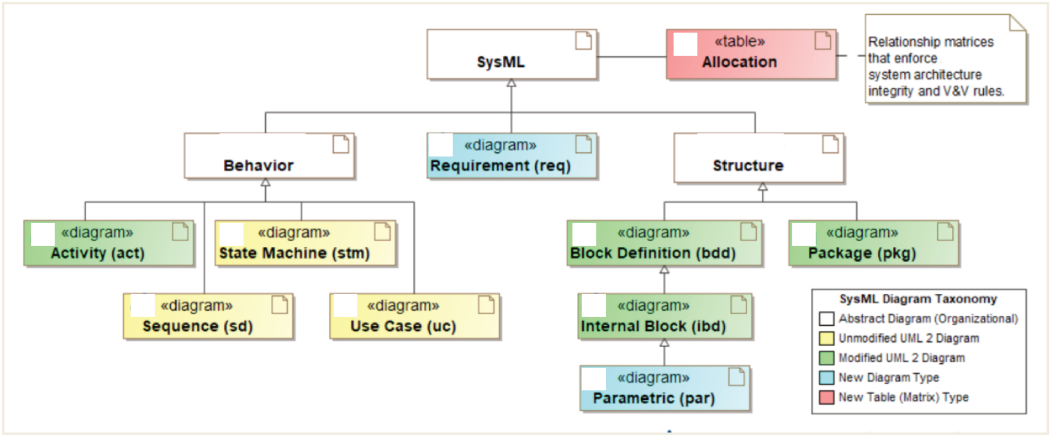
Dans cette annexe, les possibilités et limites de modélisation avec le langage SysML sont explorées. Nous avons choisi d'évaluer les capacités de formalisation des problèmes de conception avec SysML du point de vue de la modélisation, afin de décider s'il s'agit ou non d'un langage ou formalisme de spécification de problèmes ou de systèmes. Si oui, alors on pourrait l'utiliser directement au sein d'une approche MBSE orientée problème, sous réserve de développer des outils de résolution adéquats. Il s'agit également du langage le plus mature pour l'Ingénierie des Systèmes à ce jour, il est donc pertinent à étudier et à maîtriser un minimum.

E.1 Approches MBSE et outils pour SysML

Le langage SysML comprend 9 diagrammes : de structure (dont paramétrique), de comportement, d'exigences (cf. Fig. E.1).

Différentes approches MBSE ont été développées (ARCADIA [212], SYSMOD [213], Agile MBSE [74], etc.). La norme SysML [211] ne recommende pas d'approche MBSE en particulier mais SysML est généralement associé à l'Object-Oriented Systems Engineering Method (OOSEM) [214]. L'OOSEM traite en particulier de l'analyse et de la traçabilité des exigences, une étape consistant à analyser les possibilités de réutilisation des objets, après avoir évalué les différentes solutions alternatives du système. L'OOSEM oriente les concepteurs sur l'ordre d'utilisation des diagrammes et mécanismes (allocation, traçabilité) sans pour autant l'indiquer précisément.

Figure E.1 Les 9 diagrammes SysML [211].



Quelques environnements de modélisation et modules intégrés ont été développés par les industriels pour supporter SysML, mais seul Papyrus-SysML [1] est actuellement libre, gratuit et complet. L'environnement de modélisation Modelio [2] est incomplet dans sa version gratuite (pas de diagramme d'exigences) et le logiciel Visual Paradigm Enterprise [3] n'est accessible qu'en version d'essai uniquement. L'environnement de modélisation Visual Paradigm Enterprise a été utilisé pour la construction des modèles en SysML présentés dans ce manuscrit. Les industriels ont par ailleurs développés différents outils de simulation pour SysML : MagicDraw (No Magic), Cameo Systems Modeler (No Magic), EA Sparx, Windchill Modeler (anciennement Integrity Modeler). On retrouve également des solveurs sous forme de plug-in dédiés à la simulation des modèles SysML [215] comme ParaSolver ou encore ParaMagic [4], Melody [5] et Solvea [6].

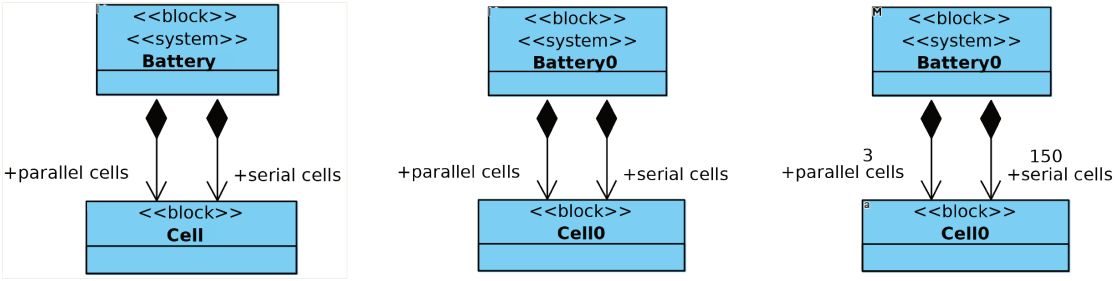
E.2 Modélisation du problème en SysML

E.2.1 Modélisation des éléments du problème

Les éléments du problème de conception de la batterie peuvent être modélisés comme sur la Fig. 3.17. Ce Block Definition Diagram (bdd) est composé du système batterie *bat* et du profil de mission composé de 3 profils élémentaires `elp`. Cette vue extérieure du système

1. plug-in développé par le CEA et intégré dans l'environnement de développement intégré Eclipse.
2. https ://www.modelio.org/.
3. https ://www.visual-paradigm.com/.
4. https ://www.nomagic.com/product-addons/magicdraw-addons/paramagic-plugin.
5. http ://intercax.com/products/melody/.
6. http ://intercax.com/products/solvea/.

Figure E.2 Une batterie sous-définie, sous-définie de type 0 et définie (SysML).



(a) type, n_s, n_p inconnues. **(b)** n_s, n_p inconnues. **(c)** type, n_s et n_p connues.

ne spécifie pas si le système est défini ou sous-défini. Si le type de composants n'est pas exprimé explicitement (cellules de type 0, 1 ou 2), on peut considérer qu'une telle modélisation représente un système sous-défini.

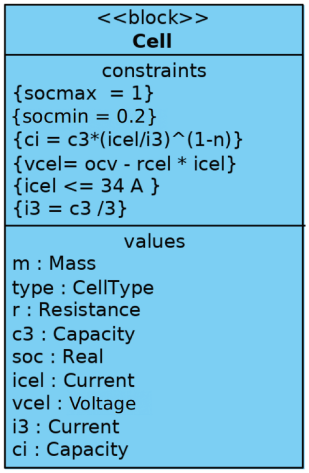
E.2.2 Modélisations pour une batterie sous-définie et définie

La représentation externe d'une batterie sous-définie, partiellement sous-définie et définie est présentée dans le bdd Fig. E.2a, E.2b et E.2c. En SysML, les arguments d'un bloc (ex : `mcel`, `type`, `rcel`, `c3`) ne sont pas différenciés des Variables (ex : `ci`, `soc`), faisant perdre une information importante de modélisation (cf. Fig. E.3a). Il y a cependant la possibilité d'ajouter des précisions au modèle par annotation mais cela encombre visuellement celui-ci.

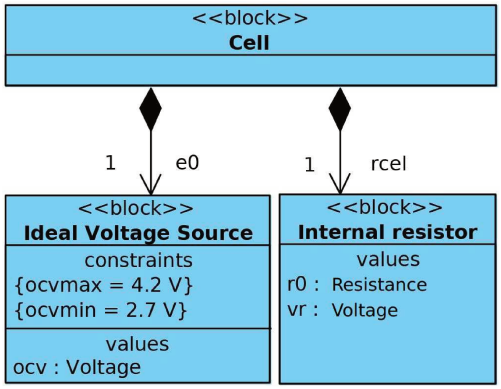
Une source de tension idéale sera définie par son OCV (cf. Fig. E.3b). Les relations algébriques peuvent facilement être exprimées sous forme de contraintes, dans un compartiment dédié de manière déclarative. Les contraintes de conception spécifiant la variation de l'OCV en fonction du SOC peuvent également être modélisées directement dans le bloc ou dans un bloc de cellule de contrainte pour être réutilisables dans un diagramme Parametric (par). Des types (ex : W) peuvent être créés pour typer les Variables d'un bloc et sont définis par des grandeurs physiques (ex : power) pour décrire des systèmes réels. et une unité (ex : Watts).

Chaque type de cellules peut être modélisé par héritage de la cellule générale `Cell` (cf. Fig. E.4). On peut imaginer spécifier la structure interne d'une batterie sous-définie *par omission*, en n'indiquant ni le type, ni les nombres de cellules en série et parallèle (cf. Fig. E.5). Celle-ci sera alors composée d'un nombre inconnu d'instances de cellule *cell* du bloc *Cell*.

Figure E.3 Modélisation d'une cellule (SysML).

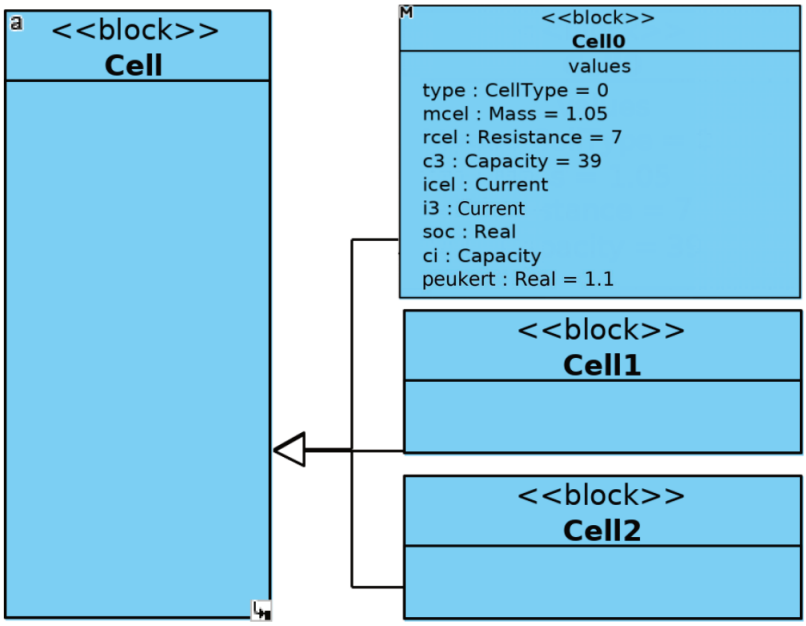


(a) Vue interne d'un bloc de cellule.



(b) Composition d'un bloc de cellule.

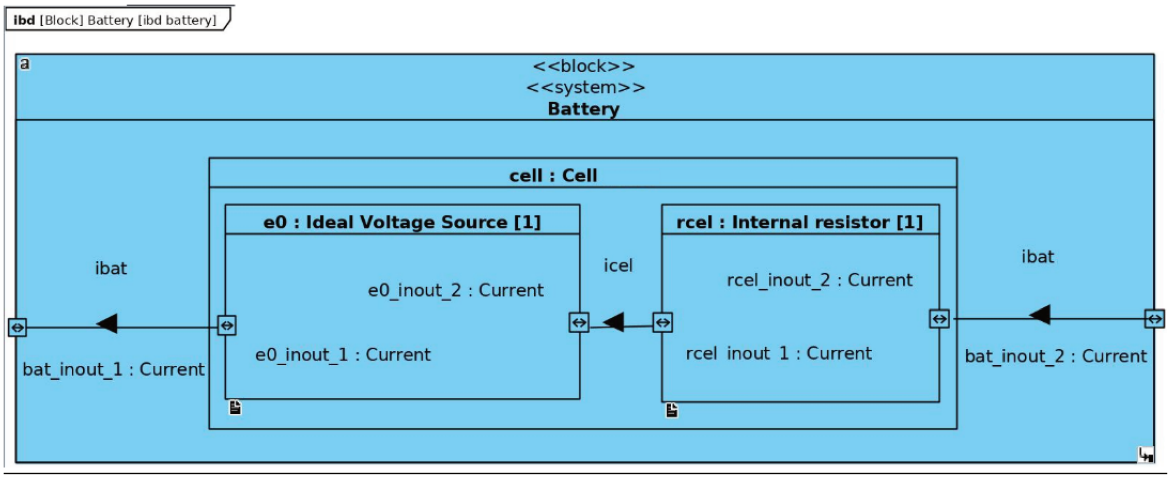
Figure E.4 Modélisation des 3 types de cellules sous forme de blocs (SysML).



Limites

Si de nombreux types de cellules sont à modéliser, un langage graphique sera rapidement saturé visuellement et aussi probablement au niveau des capacités de mémoire de l'outil de modélisation.

Figure E.5 Structure interne d'une batterie (ibd) (SysML - sans variabilité).



La multiplicité d'un lien de référence permet de spécifier le nombre d'instances auquel un bloc est relié (par exemple 450 cellules). On peut ainsi renseigner si un constituant du système est unique, optionnel ou multiple, ce qui traduit une forme de variabilité structurelle (la redondance d'un constituant, qui est une règle de configuration du système). Cependant, la multiplicité ne permet pas de définir le domaine d'une Variable. Cette remarque est tout de même à nuancer car la norme SysML v1.6 indique que le stéréotype "*interval*" peut être utilisé pour spécifier des domaines d'intervalles de réels sous forme de lois de probabilité.

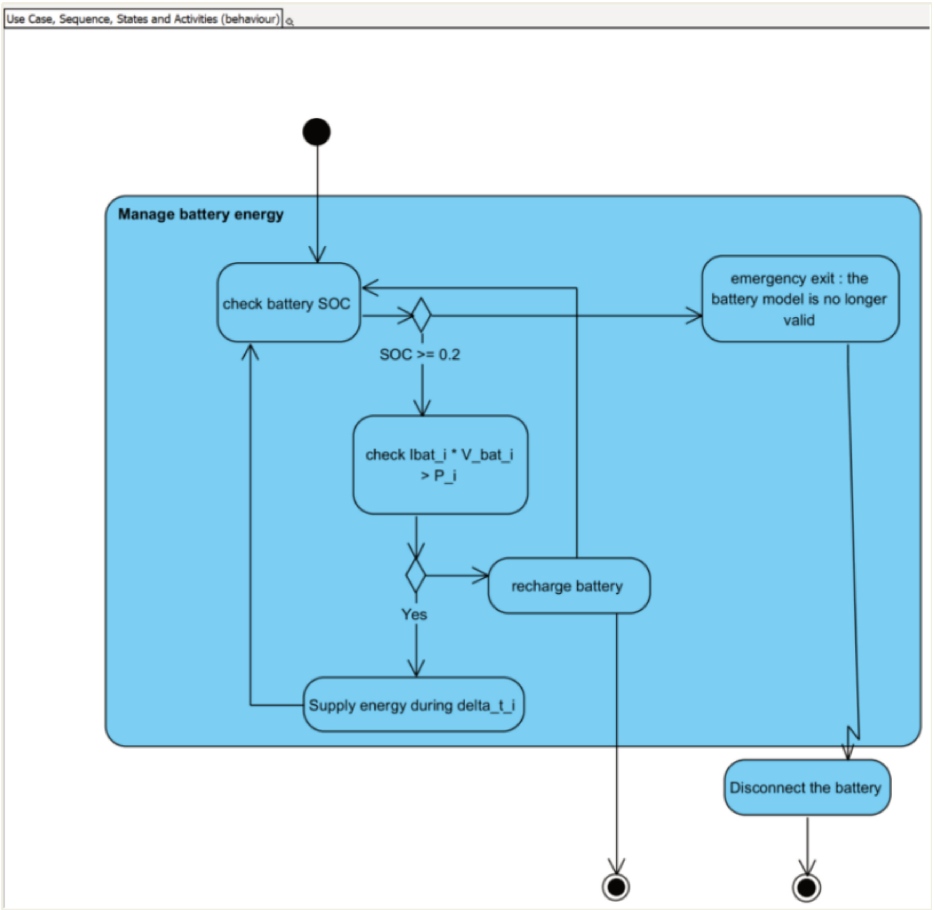
Remarque 1 : Pour l'analyse d'architectures, les modèles structurels d'un système défini sont à compléter par un ensemble de diagrammes de comportement décrivant son fonctionnement (cf. Fig. E.6 et E.7).

Remarque 2 : La Fig. E.8 montre que l'on peut modéliser les types de batterie (0, 1 et 2) à partir d'un modèle général pour une modélisation du problème comme un choix de batterie. Une batterie est alors modélisée comme une cellule équivalente (cf. Fig. E.9). Les variables déduites des instances *e0bat* et *rbat* doivent alors être contraintes explicitement par les relations du modèle électrique équivalent.

E.2.3 Modélisation de l'exigence sur l'énergie à fournir

La spécification des exigences peut être modélisée textuellement dans un diagramme req. On peut ainsi clairement séparer les éléments de l'espace du problème de ceux des solutions. Des stéréotypes spécifiques comme *functionalRequirement* permettent d'ajouter plus

Figure E.6 Diagramme d'activité (act) décrivant le fonctionnement séquentiel de la batterie.



de précision au type d'exigence. La modélisation de l'exigence E1 sur l'énergie à fournir est donnée Fig. E.10. Elle est construite en raffinant un bloc plus général d'exigence système. Le diagramme req peut aussi être représenté sous forme de table, ce qui permet de garder le diagramme lisible si de nombreuses exigences sont à spécifier. Cette possibilité dépend cependant de l'environnement de modélisation. Chaque exigence doit pouvoir être modélisable mathématiquement sous forme de contrainte.

Ce qui nous intéresse notamment est de pouvoir contraindre entre elles les valeurs des Variables déclarées dans les blocs de contraintes (ex : `vbat` et `velp` Fig. E.11). L'avantage du diagramme par comparaison à la simple définition de contrainte dans un bloc système est double : les contraintes sont réutilisables et on peut modéliser les intéractions entre différents systèmes. La traçabilité entre exigences et contraintes peut être assurée grâce aux relations *verify* et *satisfy* et entre exigences et blocs systèmes par allocation.

Figure E.7 Diagramme de séquence (seq) décrivant l'interaction de la batterie avec son environnement.

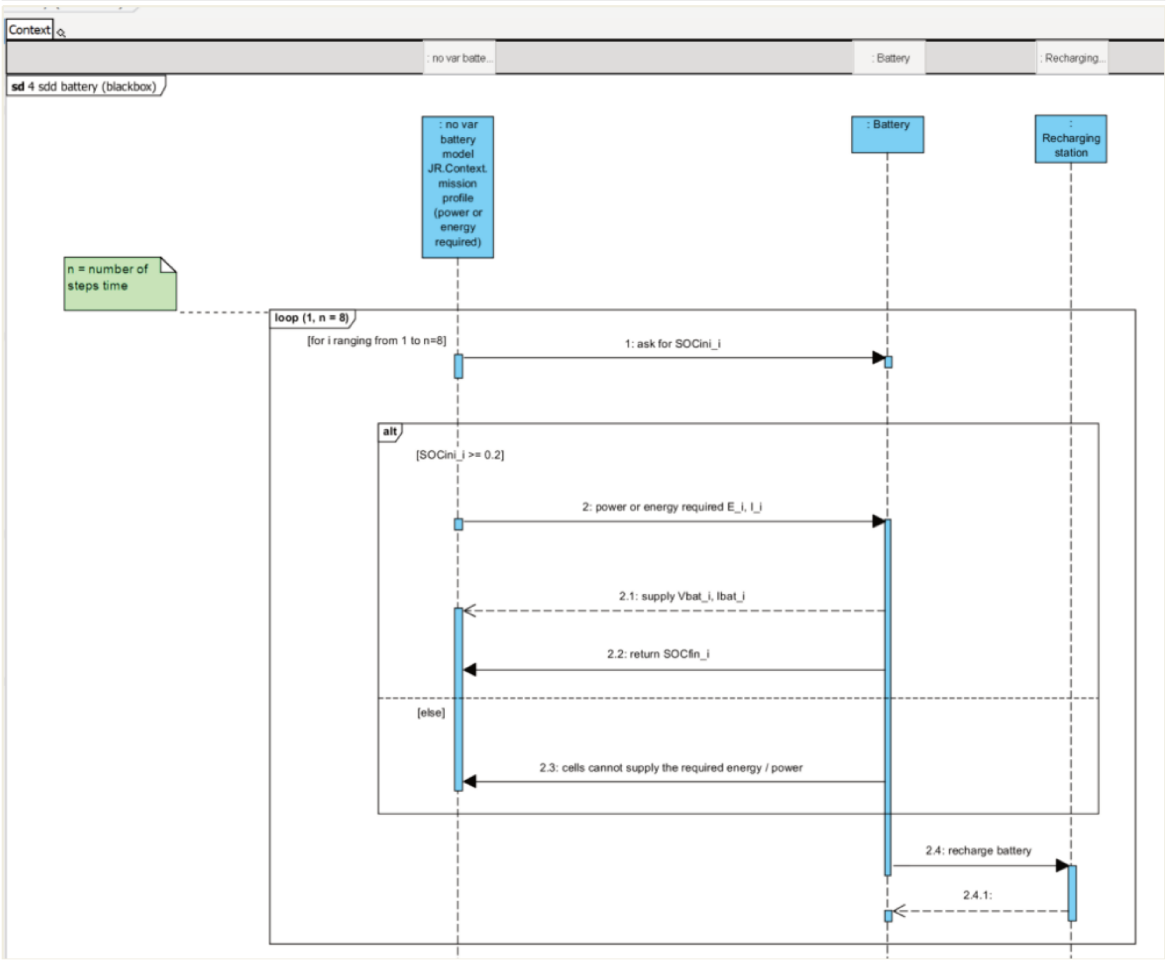
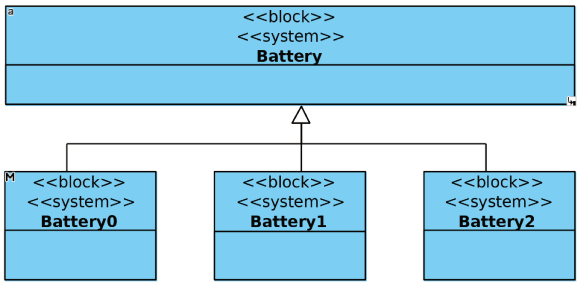


Figure E.8 Modélisation de 3 types de batteries (SysML).



E.2.4 Modélisation du choix de composants

SysML ne dispose pas de concepts pour représenter des catalogues de composants. Deux concepts SysML ont été dérivés pour cet exemple : la table req et le bloc de contraintes. La table est inadéquate car elle possède une disposition fixe qui rend les modifications diffi-

Figure E.9 Structure interne d'une batterie vue comme une cellule équivalente (SysML).

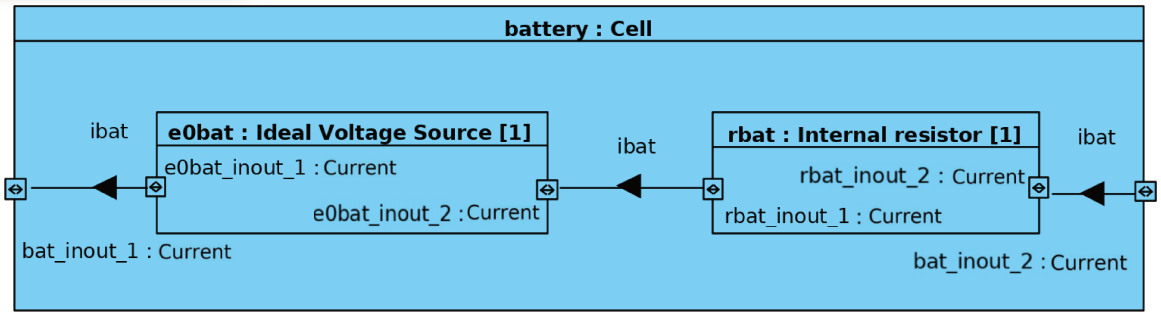


Figure E.10 Modélisation de l'exigence E1 (SysML - sans variabilité)

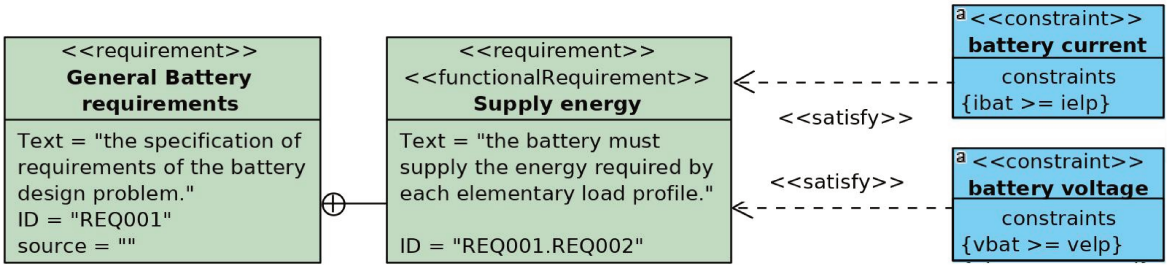
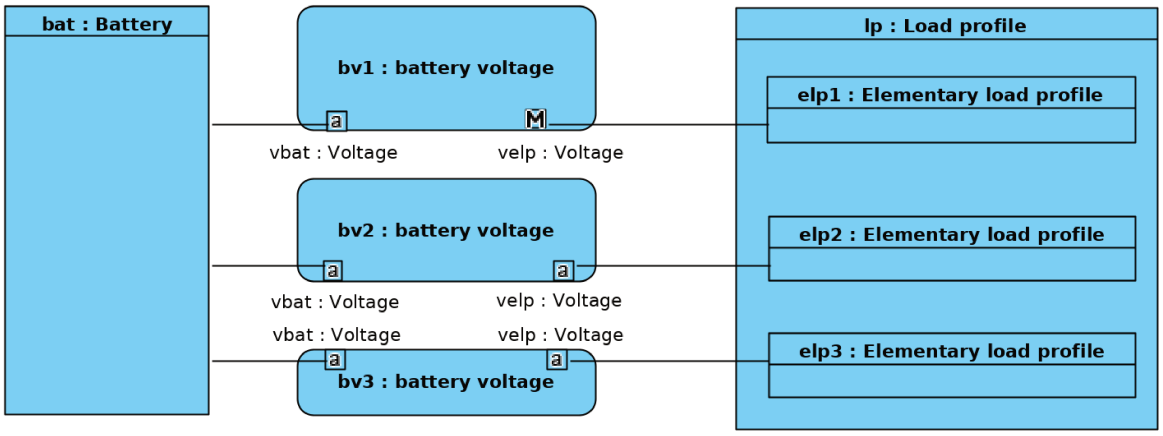
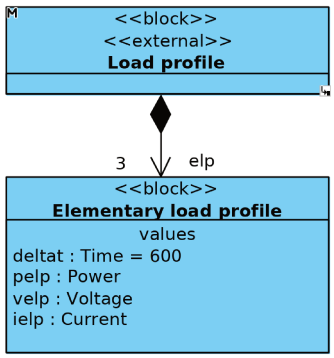


Figure E.11 Modélisation de la contrainte de puissance à fournir - tension de la batterie (SysML).

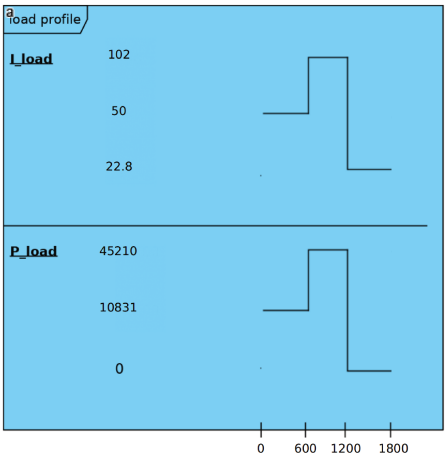


ciles. Pour modéliser le choix de cellules avec les blocs de contraintes, on peut modéliser chaque type de cellules puis les contraindre textuellement par des annotations pour tenter de modéliser le choix de composants dans un diagramme paramétrique.

Figure E.12 Modélisations d'un profil de mission (SysML).



(a) Comme bloc SysML.



(b) Comme diagramme temporel UML.

E.2.5 Modélisation d'un profil de mission

Un profil de mission peut être représenté comme un système dans un bdd par un bloc `Load profile` stéréotypé `external` pour préciser qu'il ne s'agit pas du système à concevoir (cf. Fig. E.12a). Cette précision permet de communiquer clairement sur le rôle des différents éléments du problème sans alourdir la modélisation. Un bloc `Elementary load profile` paramétré par une durée `deltat`, une puissance `pelp`, une tension `velp` et un courant `ielp` permet de représenter facilement tout profil de mission élémentaire. Comme SysML est un langage graphique, le plus pratique serait de pouvoir modéliser le profil de mission directement comme un graphe. Cependant, aucun diagramme temporel n'existe en SysML. La norme 1.6 de SysML indique que les diagrammes temporels ne sont pas encore assez matures pour SysML, probablement au regard des capacités des outils de simulation existants. SysML doit donc être complété par d'autres langages possédant des vues temporelles comme UML, MARTE ou AADL. Un diagramme temporel UML a été utilisé pour une modélisation alternative du profil de mission sous forme de graphe (cf. Fig. E.12b). Il faut également pouvoir assurer la consistence du modèle UML avec les modèles SysML. Comme le diagramme temporel ne fait pas partie de SysML, celui-ci ne peut pas réellement être maintenu à jour. L'utilisation de ce diagramme UML ne peut cependant pas être généralisée car elle dépend de l'outil de modélisation.

E.2.6 Premières remarques et conclusions

Cette première expérience de modélisation du problème avec SysML tend à considérer ce langage pour la spécification de systèmes plutôt que de problèmes en raison du manque de variabilité structurelle (et qui se confirme par l'utilisation d'outils d'analyse). Les approches MBSE actuelles ne s'appuient donc exclusivement que sur des langages de spécification de systèmes et non de problèmes de conception. En conséquence, celles-ci peuvent être considérées comme des approches orientées solutions.

Les nombreuses possibilités pour représenter un même concept (ex : il y a cinq façons de représenter une solution d'allocation) autorisent une liberté de modélisation selon les préférences des concepteurs mais n'aident pas à appréhender SysML de la meilleure façon. Un choix relativement mauvais peut rendre un modèle illisible (par exemple, modéliser une solution d'allocation par un lien *allocate to* sera défavorable face à une table si il y en a beaucoup à représenter). Pour les concepteurs inexpérimentés, le manque de lignes directrices pour la construction des diagrammes, ajouté aux nombreuses possibilités peut empêcher d'exploiter tout le potentiel du langage.

L'utilisation de SysML et des approches MBSE est un outil important dans l'ingénierie systèmes pour aiguiller la conception des systèmes complexes. Les limites en pré-conception se trouvent dans l'obtention des solutions à modéliser. Il s'avère très vite fastidieux de devoir représenter toutes les solutions possibles de systèmes (ici il n'y en avait que trois mais ce sont généralement des dizaines ou des centaines de composants à choisir dans les catalogues constructeurs). SysML reste un langage efficace pour les industriels qui ont su développer des environnement de modélisation permettant d'assurer la consistance des modèles lors de leurs modifications.

Une étude [216] menée sur la diffusion de SysML dans l'IS a révélé que le langage était de plus en plus présent dans les outils industriels. Les industriels souligneraient notamment la non-ambiguité et la précision avec lesquelles le système et ses caractéristiques principales peuvent être représentées. Cependant, tous les diagrammes ne seraient pas utilisés avec la même aisance : alors que les diagrammes de blocs externe et internes (structurels) et celui des exigences sont très utilisés, le diagramme paramétrique resterait le moins utilisé et le moins compris.

Utiliser un langage MBSE pour la synthèse nécessite que celui-ci possède une plus grande expressivité structurelle, qu'il soit formel et que des techniques et outils de résolution soient mis en place pour la résolution. Une approche qui reposerait sur un langage de spécification de systèmes étendus avec variabilité, peut être qualifiée d'approche orientée solutions étendue.

E.3 Modélisation du problème en SysML étendu

En IL, plusieurs approches de conception de systèmes logiciels orientées solutions étendues ont été développées pour ajouter de la variabilité à certains langages de spécification de systèmes tels que AUTOSAR, UML ou SysML [30, 46, 175, 217–219] ou encore encouragent de telles approches [29]. Pour gérer les problèmes de dimensionnement en IS, aussi bien du côté de la formalisation que de la résolution, SysML a notamment été utilisé conjointement avec le langage de programmation GAMS [77].

Pour tenter de pouvoir modéliser la variabilité en SysML, il faut faire du langage un formalisme. Quelques travaux ont été menés dans ce sens en IL. Des approches basées sur SysML mêlant conception et optimisation ont été développées pour les systèmes mécaniques (ModelCenter, ConsolOptcad) mais ne traitent pas les problèmes mixtes. De plus, celles-ci ne sont disponibles que dans des outils commerciaux et sont donc moins accessibles. D'un autre côté, un modèle SysML peut être complété par des annotations et des stéréotypes pour y ajouter des ddl [30, 46, 219]. Les annotations et stéréotypes permettent notamment d'étendre le langage SysML. Les récents travaux de thèse [46] montrent que la problématique liée à l'absence de variabilité en SysML est encore très actuelle. [46] propose des moyens d'implémenter la variabilité dans les modèles pour la gestion de problèmes de conception de systèmes embarqués. Ces travaux utilisent conjointement le langage de modélisation de systèmes embarqués MARTE pour modéliser la partie logicielle et SysML pour la partie physiques des systèmes embarqués.

La PPC est employée pour la conception par optimisation pour modéliser et résoudre des problèmes à variables mixtes. [46] souligne l'importance du choix de ddl et de leurs domaines.

E.3.1 Modélisation du choix de composants

La liste des cellules disponibles est désormais définie dans un diagramme de bloc (bdd) disctinct, avec un stereotype `ComponentInstance` et les variables discrètes du type et des nombres en série et parallèles de cellules déclarées avec leurs domaines respectifs. Les contraintes de configuration du système relatives au choix de composants peuvent être modélisées (ainsi que la redondance des composants, la présence conditionnelle, etc.). Un métamodèle décrivant le CSP/CSOP est ensuite défini pour une résolution par différents solveurs. [46] propose des solutions pour intégrer dans un même environnement formalisation et résolution des problèmes de conception. Pour la résolution, [46] a développé des modules pour l'environnement Eclipse permettant la transformation et la résolution des modèles SysML

avec des solveurs tiers : Choco (variables entières), PyOpt (variables continues) et Labix (variables entières). Cependant, il faut que les concepteurs choisissent le solveur adéquat au problème en fonction du type de variables. Il faut aussi une certaine maîtrise des trois solveurs et techniques de résolution associées. La résolution des problèmes mixtes ressort ainsi comme une nécessité et une difficulté [46]. L'utilisation d'une telle intégration est donc lourde et peu pratique. L'idéal serait de disposer d'un solveur unique capable de traiter directement les problèmes de conception mixtes.

E.3.2 Conclusion générale sur la pertinence de SysML étendu

Cette extension de SysML complétant les modèles existants et adaptant les diagrammes existants permet de réduire les coûts de modélisation. Ces travaux montrent également la difficulté pour la résolution lorsque l'on exprime un problème dans un langage haut-niveau et déclaratif de modélisation plutôt que de programmation puisqu'il devient nécessaire de transformer le modèle SysML en un modèle exécutable pour le solveur. Cette étape de transformation rend cependant plus difficile la traçabilité des erreurs (du modèle SysML, de l'algorithme ou du modèle transformé) car celles-ci ne peuvent pas être explicitement capturées lors de la formalisation.

En conclusion, nous pouvons voir que des travaux intéressants existent pour faire de SysML un formalisme adapté à la conception. Cependant de nombreuses limites demeurent (expression de la variabilité et résolution) et de nombreux développements sont encore nécessaires pour en faire un outil totalement pertinent et efficace en pré-conception des systèmes physiques complexes.

Bibliographie

[1] Exenberger, A., and Hartmann, S., 2007. “The Dark Side of Globalization. The Vicious Cycle of Exploitation from World Market Integration : Lesson from the Congo”. In Working Papers in Economics and Statistics, University of Innsbruck, Institüt für Finanzwiss.

[2] Hubert, A., Yvars, P.-A., Meyer, Y., and Zimmer, L., 2016. “Conception préliminaire optimale des systèmes électriques. Une approche par synthèse”. In Symposium de Génie Électrique 2016, France. Juin.

[3] Zimmer, L., and Zablit, P., 2001. “Global Aircraft Predesign based on Constraint Propagation and Interval Analysis”. In CEAS Conference on Multidisciplinary Aircraft Design and Optimization. Köln, Germany.

[4] Callon, M., 1986. “The Sociology of an Actor-Network : The Case of the Electric Vehicle”. In *Mapping the Dynamics of Science and Technology*. doi :10.1007/978-1-349-07408-22.

[5] Kempton, W., and Letendre, S. E., 1997. “Electric vehicles as a new power source for electric utilities”. In Transportation Research Part D : Transport and Environment, Elsevier, ed., Vol. 2, pp. 157–175. doi :10.1016/S1361-9209(97)00001-1, issn :13619209.

[6] Miller, J. M., 2017. “Hybrid electric vehicles”. In *Handbook of Automotive Power Electronics and Motor Drives*. doi :10.1201/9781420028157, isbn :9781420028157.

[7] Egbue, O., and Long, S., 2012. “Barriers to widespread adoption of electric vehicles : An analysis of consumer attitudes and perceptions”. In Energy Policy. doi :10.1016/j.enpol.2012.06.009, issn :03014215.

[8] Chan, C. C., 2007. “The state of the art of electric, hybrid, and fuel cell vehicles”. In Proceedings of the IEEE. doi :10.1109/JPROC.2007.892489, issn :00189219.

[9] Cairns, E. J., and Albertus, P., 2010. “Batteries for electric and hybrid-electric vehicles”. In Annual Review of Chemical and Biomolecular Engineering. doi :10.1146/annurev-chembioeng-073009-100942, issn :19475438.

[10] Burke, A. F., 2007. “Batteries and Ultracapacitors for Electric, Hybrid, and Fuel Cell Vehicles”. In Proceedings of the IEEE, Vol. 95, pp. 806–820. doi :10.1109/JPROC.2007.892490, issn :00189219.

[11] Ehsani, M., Gao, Y., Gay, S. E., and Emadi, A., 2004. *Modern electric, hybrid electric, and fuel cell vehicles : Fundamentals, theory, and design*. isbn :9781420037739.

[12] Roboam, X., 2012. *Conception systémique pour la conversion d’énergie électrique 1. Gestion, analyse et synthèse*. Traité RTA, série Génie électrique. Lavoisier, Edition Hermès.

[13] Fiorèse, S., and Meinadier, J., 2012. *Découvrir et comprendre l’ingénierie système*. AFIS. Cépaduès Éditions. isbn :9782364930056.

[14] Narin’yan, A., 1983. “Subdefiniteness and Basic Means of Knowledge Representation”. In Computers and Artificial Intelligence, Bratislawa, Vol. 5, pp. 443–452.

[15] Telerman, V., and Ushakov, D., 1996. “Subdefinite models as a variety of constraint programming”. In Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 1996), pp. 157–164. doi :10.1109/TAI.1996.560446, issn :10823409.

[16] Telerman, V., Sidorov, V., and Ushakov, D., 1997. “0bject-Oriented constraint programming environment NeMo+ and its applications”. In Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 1997), pp. 296–303. doi :10.1109/TAI.1997.632269, issn :10823409.

[17] Zimmer, L., Yvars, P.-A., and Lafaye, M., 2020. “Models of requirements for avionics architecture synthesis : safety, capacity and security”. In Complex Systems Design Management (CSDM) 2020.

[18] Davis, E., 1987. “Constraint propagation with interval labels”. In Artificial Intelligence, Vol. 32, Elsevier, pp. 281–331. doi :10.1016/0004-3702(87)90091-9, issn :00043702.

[19] Quine, W., 1969. *Set Theory and Its Logic*, 2nd ed. Harvard University Press. isbn :9780674802070.

[20] Yvars, P. A., and Zimmer, L., 2018. "System Sizing with a Model-Based Approach : Application to the Optimization of a Power Transmission System". In Mathematical Problems in Engineering. doi :10.1155/2018/6861429, issn :15635147.

[21] Chenouard, R., 2010. "Résolution par satisfaction de contraintes appliquée à l'aide à la décision en conception architecturale". PhD thesis, Ecole Nationale Supérieure d'Arts et Métiers. See :tel-00486662.

[22] Regnier, J., 2003. "Conception de systemes hétérogènes en génie electrique par optimisation évolutionnaire multicritère". PhD Thesis, Institut National Polytechnique de Toulouse, December.

[23] Fischer, X., 2000. "Strategie de conduite du calcul pour l'aide a la decision en conception mecanique intégrée. Application aux appareils à pression". PhD thesis, ENSAM Sciences et techniques Paris.

[24] Messine, F., Nogarede, B., and Lagouanelle, J.-L., 1998. "Optimal design of electromechanical actuators : a new method based on global optimization algorithm". In IEEE Transactions On Magnetics, Vol. 34, pp. 299–308.

[25] Fitan, E., Messine, F., and Nogarède, B., 2004. "The Electromagnetic Actuator Design Problem : A General and Rational Approach". In IEEE Transactions On Magnetics, Vol. 40, pp. 1579–1590.

[26] Mazhoud, I., Hadj-Hamou, K., Bigeon, J., and Remy, G., 2012. "The electromagnetic actuator design problem : an adapted interval global optimization algorithm". In IEEE Transactions On Magnetics, Vol. 48, pp. 387–390.

[27] Kang, K., Cohen, J., Hess, S., Novak, W., and Peterson, A., 1990. "Feature-Oriented Domain Analysis (FODA) Feasibility Study". CMU/SEI-90-TR-21, Carnegie Mellon University.

[28] Acher, M., Cleve, A., Collet, P., Merle, P., Duchien, L., and Lahire, P., 2014. "Extraction and evolution of architectural variability models in plugin-based systems". In Software and Systems Modeling, Vol. 13, pp. 1367–1394. doi :10.1007/s10270-013-0364-2, issn :16191374.

[29] Bąk, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., and Wąsowski, A., 2016. "Clafer : Unifying Class and Feature Modeling". In Journal of Software and Systems Modeling, Vol. 15, Springer Berlin Heidelberg, pp. 811–845. doi :10.1007/s10270-014-0441-1.

[30] Fontoura, M., Pree, W., and Rumpe, B., 2001. *The UML Profile for Framework Architectures*, 1st ed. Addison-Wesley. isbn :9780201675184.

[31] Czarnecki, K., 1998. "Generative programming : Principles and techniques of software engineering based on automated configuration and fragment-based component models". PhD Thesis, Technical University Of Ilmenau, October.

[32] Sinnema, M., and Deelstra, S., 2007. "Classifying variability modeling techniques". In Information and Software Technology. doi :10.1016/j.infsof.2006.08.001, issn :09505849.

[33] Schramm, J., Dohrmann, P., and Kuhrmann, M., 2015. "Development of flexible software process lines with variability operations : A longitudinal case study". In ACM International Conference Proceeding Series. doi :10.1145/2745802.2745814, isbn :9781450333504.

[34] JCGM, 2008. "Evaluation of measurement data - Guide to the expression of uncertainty in measurement". GUM, BIPM.

[35] Jackson, M., and Zave, P., 1995. "Deriving specifications from requirements : An example". In Proceedings 17th International Conference on Software Engineering (ICSE), pp. 15–24. doi : 10.1145/225014.225016.

[36] Jackson, M., 1999. "Problem analysis using small problem frames". In South African Computer Journal, Vol. 22 of *Special Issue of WOFACS '98*, pp. 47–60.

[37] Jackson, M., 2000. "Problem analysis and structure". In Keynote Talk at ITG/SEV Symposium, NATO Summer School, Marktoberdorf, Germany.

[38] Jackson, M., 1997. "The meaning of requirements". In Annals of Software Engineering, Vol. 3, pp. 5–21. doi :10.1023/A :1018990005598.

[39] Jackson, M., 2000. *Problem Frames : Analysing and Structuring Software Development Problems*. Addison-Wesley.

[40] Dobre, D., 2010. “Contribution à la modélisation d’un système interactif d’aide à la conduite d’un procédé industriel”. Phd thesis, Université Henri Poincaré Nancy I, November. See :tel-00553267v3.

[41] Bouffaron, F., 2016. “Co-spécification système exécutable basée sur des modèles. Application à la conduite interactive d’un procédé industriel critique”. PhD Thesis, Université de Lorraine, Nancy, January. See :tel-01754485v2.

[42] Roboam, X., 2012. *Conception systémique pour la conversion d’énergie électrique 2. Approche intégrée par optimisation*. Traité RTA, série Génie électrique. Lavoisier, Edition Hermès.

[43] Nguyen-Huu, H., Retière, F., Wurtz, F., Roboam, X., Sareni, B., and Aléjo, D., 2009. “Optimal Sizing of an embedded electrical system with an approach for limiting the search space”. In The International booktitle for Computation and Mathematics in Electrical and Electronic Engineering (COMPEL), Vol. 28, pp. 1141–1154.

[44] Li, B., Roche, R., and Miraoui, A., 2017. “Microgrid sizing with combined evolutionary algorithm and MILP unit commitment”. In Applied Energy. doi :10.1016/j.apenergy.2016.12.038, issn :03062619.

[45] Yvars, P. A., and Duhau, Q., 2012. “Managing system configuration through constraint propagation in function and value analyses of product families”. In Engineering Design, Vol. 23, pp. 361–377. doi :10.1080/09544828.2011.591774, issn :09544828.

[46] Leserf, P., 2017. “Optimisation de l’architecture de systemes embarqués par une approche basée modèle”. Phd thesis, Institut supérieur de l’aéronautique et de l’espace de Toulouse (ISAE-ONERA MOIS), May.

[47] Berger, N., 2010. “Modélisation et résolution en programmation par contraintes de problèmes mixtes continu / discret de satisfaction de contraintes et d’ optimisation”. PhD thesis, Université de Nantes, October.

[48] Yvars, P.-A., 2010. “A constraint-based approach to the composition relation management of a product class in design”. In Journal of Computing and Information Science in Engineering (JCISE), Vol. 10.

[49] Gil Herrera, J., and Botero, J. F., 2016. "Resource Allocation in NFV : A Comprehensive Survey". In IEEE Transactions on Network and Service Management. doi :10.1109/TNSM.2016.2598420, issn :19324537.

[50] Kaffa-Jackou, R., Brochado, M. R., Cruz, L. G., Batatia, H., and Mora-Camino, F., 2009. "A multilevel approach for airport security ressource allocation". In World Academy of Science, Engineering and Technology. issn :2010376X.

[51] Contreras-Moreno, G., and Toledo, G., 2018. "(Preliminary Concept) Design of a mechatronic system to relief search and rescue of victims after an earthquake disaster". PhD thesis, Fachhochschule Aachen, and Centro de Ingenieria y Desarrollo Industrial, September.

[52] Royce, W. W., 1987. "Managing the Development of Large Software Systems : Concepts and Techniques". In Proceedings of the 9th International Conference on Software Engineering, ICSE '87, IEEE Computer Society Press, p. 328–338. doi : 10.5555/41765.41801, isbn : 0897912160.

[53] Larman, C., and Basili, V., 2003. "Iterative and incremental developments. a brief history". In Computer, Vol. 36, pp. 47–56. doi : 10.1109/MC.2003.1204375, issn : 0018-9162.

[54] Martin, J., 1991. *Rapid Application Development*. Macmillan Coll. Div.

[55] Boehm, B., 2000. "Spiral Development : Experience, Principles, and Refinements". In Spiral Development Workshop, p. 49. isbn :CMU/SEI-00-SR-08, ESC-SR-00-08.

[56] Roques, P., and Vallée, F., 2002. *UML en action, 2e édition : De l'analyse des besoins à la conception en Java*, 2nd ed. Eyrolles. isbn : 9782212112139.

[57] Sohlenius, G., 1992. "Concurrent engineering". In CIRP Annals - Manufacturing Technology. doi :10.1016/S0007-8506(07)63251-X, issn :17260604.

[58] Brisset, S., 2007. "Démarches et outils pour la conception optimale des machines électriques". Habilitation à diriger des recherches, Université des sciences et techniques de Lille, December.

[59] Potoyan, D. A., Savelyev, A., and Papoian, G. A., 2013. "Recent successes in coarse-grained modeling of DNA". Vol. 3, pp. 69–83. doi : 10.1002/wcms.1114.

[60] Ingólfsson, H. I., Lopez, C. A., Uusitalo, J. J., de Jong, D. H., Gopal, S. M., Periole, X., and Marrink, S. J., 2014. "The power of coarse graining in biomolecular simulations". Vol. 4, pp. 225–248. doi : https ://doi.org/10.1002/wcms.1169.

[61] Kmiecik, S., Gront, D., Kolinski, M., Wieteska, L., Dawid, A. E., and Kolinski, A., 2016. "Coarse-Grained Protein Models and Their Applications". Vol. 116, pp. 7898–7936. doi : 10.1021/acs.chemrev.6b00163.

[62] Iwasaki, Y., and Simon, H. A., 1994. "Causality and model abstraction". In Artificial Intelligence. doi :10.1016/0004-3702(94)90014-0, issn :00043702.

[63] Nuseibeh, B., 2001. "Weaving together requirements and architectures". In IEEE Computer, Vol. 34, pp. 115–117.

[64] Eichelberger, H., and Schmid, K., 2013. "A Systematic Analysis of Textual Variability Modeling Languages". In Proceedings of the 17th International Software Product LineConference (SPLC'13), ACM, pp. 12–21. doi :10.1145/2491627.2491652.

[65] Gunter, C. A., Gunter, E. L., Jackson, M., and Zave, P., 2000. "A reference model for requirements and specifications". In IEEE Software, Vol. 17, pp. 37–43.

[66] Hall, J. G., and Rapanotti, L., 2005. "Problem frames for socio–technical systems. requirements engineering for socio–technical systems". Idea Publishing Group, pp. 318–339.

[67] Diampovesa, S., Hubert, A., and Yvars, P.-A., 2021. "Designing physical systems through a model-based synthesis approach. Example of a Li-ion battery for electrical vehicles". In Computers in Industry, Vol. 129, p. 103440. issn :01663615, doi :10.1016/j.compind.2021.103440.

[68] Duan, Y., and Ionel, D. M., 2013. "A Review of Recent Developments in Electrical Machine Design Optimization Methods With a Permanent-Magnet Synchronous Motor Benchmark Study". In IEEE Transactions on Industry Applications, Vol. 49, pp. 1268–1275. doi :10.1109/TIA.2013.2252597, isbn :9781457705403, issn :00939994.

[69] Giraud, X., 2014. "Méthodes et outils pour la conception optimale des réseaux de distribution d'électricité dans les aéronefs". PhD thesis, INSA de Toulouse, February.

[70] IEEE/ISO/IEC/S2ESC, 1999. IEEE 1220 :1999 - Standard for Application and Management of the Systems Engineering Process. Tech. rep., January. Software Systems Engineering Standards Committee. IEEE Computer Society.

[71] ANSI/EIA, 2003. American National Standards Institute (ANSI)/Electronic Industries Association (EIA). ANSI/EIA 632-2003. Processes for Engineering a System. Tech. rep.

[72] ISO/IEC, 2015. ISO/IEC 15288 : 2015 Systems and software engineering - System life cycle processes. Tech. rep. isbn :0738156655.

[73] Friedenthal, S., Moore, A., and Steiner, R., 2015. "Model-Based Systems Engineering". In *A Practical Guide to SysML*. doi :10.1016/b978-0-12-800202-5.00002-3.

[74] Douglass, B. P., 2016. "What Is Model-Based Systems Engineering ?". In *Agile Systems Engineering*. doi :10.1016/b978-0-12-802120-0.00001-1.

[75] Dai, Q., Kelly, J. C., Gaines, L., and Wang, M., 2019. "Life cycle analysis of lithium-ion batteries for automotive applications". In Batteries, Vol. 5. doi :10.3390/batteries5020048, issn :23130105.

[76] Garcia, J., Millet, D., Tonnelier, P., Richet, S., and Chenouard, R., 2017. "A novel approach for global environmental performance evaluation of electric batteries for hybrid vehicles". In Journal of Cleaner Production, Vol. 156, pp. 406–417. doi :10.1016/j.jclepro.2017.04.035, issn :09596526.

[77] Shah, A., Paredis, C., Burkhart, R., and Schaefer, D., 2012. "Combining Mathematical Programming and SysML for Automated Component Sizing of Hydraulic Systems". In JCISE - Journal of Computing and Information Science in Engineering, Vol. 12. doi : 10.1115/1.4007764.

[78] Martins, J. R. R. A., and Lambe, A. B., 2013. "Multidisciplinary design optimization : A survey of architectures". In AIAA Journal, Vol. 51, pp. 2049–2075. doi : 10.2514/1.J051895.

[79] Tran, T. V., 2009. "Problèmes combinatoires et modèles multi-niveaux pour la conception optimale des machines électriques". PhD thesis, Université des sciences et techniques de Lille, January.

[80] Lohninger, R., Grabner, H., Weidenholzer, G., Silber, S., and Amrhein, W., 2012. "Modelling , Simulation and Design of a novel Permanent Magnetic Reluctance Machine". In International Symposium on Power Electronics, Electrical Drives, Automation and Motion, pp. 620–624.

[81] Kobler, R., Andessner, D., Passenbrunner, J., and Amrhein, W., 2011. "Modeling, simulation and design of an axial flux machine using soft magnetic composite". In 2011 IEEE Vehicle Power and Propulsion Conference (VPPC), pp. 1–6. doi :10.1109/VPPC.2011.6043083, isbn :9781612842462.

[82] Amoiralis, E., Georgilakis, P., Tsili, M., and a.G. Kladas, 2009. "Global Transformer Optimization Method Using Evolutionary Design and Numerical Field Computation". In IEEE Transactions on Magnetics, Vol. 45, pp. 1720–1723. doi :10.1109/TMAG.2009.2012795, issn :0018-9464.

[83] Rao, L., Yan, W., and He, R., 1996. "Mean Field Annealing (MFA) and Optimal Design of Electromagnetic Devices". In IEEE Transactions on Magnetics, Vol. 32, pp. 1218–1221. doi :10.1109/20.497463.

[84] Nelder, J. A., and Mead, R., 1965. "A Simplex Method for Function Minimization". In The Computer booktitle. doi :10.1093/comjnl/7.4.308, issn :00104620.

[85] Dennis, J. E., and More, J. J., 1977. "Quasi-Newton Methods, Motivation and Theory". In SIAM Review. doi :10.1137/1019005, issn :00361445.

[86] Nazareth, J. L., 2009. "Conjugate gradient method". In Wiley Interdisciplinary Reviews : Computational Statistics. doi :10.1002/wics.13, issn :19395108.

[87] Audet, C., and Kokkolaras, M., 2016. "Blackbox and derivative-free optimization : theory, algorithms and applications". Vol. 17, pp. 1–2. doi : 10.1007/s11081-016-9307-4.

[88] Wurtz, F., Kuo-Peng, P., and De Carvalho, E., 2012. "The concept of Imaginary Machine for Design and setting of optimization problems : Application to synchronous generator". In 20th International Conference on Electric Machine (ICEM'2012), Marseille, France, Vol. 1, pp. 1461–1466. doi :10.1109/ICElMach.2012.6350071, isbn :9781467301428.

[89] Goldberg, D. E., and Holland, J. H., 1988. “Genetic Algorithms and Machine Learning”. In Machine Learning, Vol. 3, pp. 95–99. doi :10.1023/A :1022602019183, issn :15730565.

[90] Sareni, B., Régnier, J., and Roboam, X., 2006. “Conception simultanée de systèmes électriques hétérogènes par algorithmes évolutionnaires multicritères. Applications à l’optimisation de chaines de traction pour véhicules électriques”. In Techniques et sciences informatiques, Vol. 25, pp. 1103–1126. doi :10.3166/tsi.25.1103-1126, issn :07524072.

[91] Arjona, M. A., Hernandez, C., and Cisneros-Gonzalez, M., 2010. “Hybrid optimum design of a distribution transformer based on 2-D FE and a manufacturer design methodology”. In IEEE Transactions on Magnetics, Vol. 46, pp. 2864–2867. doi :10.1109/TMAG.2010.2044644, issn :00189464.

[92] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T., 2002. “A fast and elitist multiobjective genetic algorithm : NSGA-II”. In IEEE Transactions on Evolutionary Computation. doi :10.1109/4235.996017, issn :1089778X.

[93] Brisset, S., and Brochet, P., 2005. “Analytical model for the optimal design of a brushless dc wheel motor”. In International Journal for Computation and Mathematics in Electrical and Electronic Engineering (COMPEL), Vol. 24, pp. 829–848. doi :10.1108/03321640510612952.

[94] Forrester, A. I., Sbester, A., and Keane, A. J., 2008. “Engineering Design via Surrogate Modelling : A Practical Guide”. Wiley. isbn :9780470060681.

[95] Brisset, S., 2007. “Démarches et outils pour la conception optimale des machines électriques”. Habilitation à diriger des recherches, Université des sciences et techniques de Lille, Lille, France, 18 décembre.

[96] Trabelsi, H., 2014. “Contribution à la prise en compte d’exigences dynamiques en conception préliminaire de systèmes complexes”. Phd thesis, Ecole Centrale Paris. See :tel-01017111.

[97] Carlson, R., Wurtz, F., and Voltolini, H., 2012. “Sizing and Optimization models : design of a set of two permanent magnet generators”. In 20 th International Conference on Electric Machine (ICEM’2012), pp. 1356–1361.

[98] Wurtz, F., Delinchant, B., Brunotte, X., Pouget, J., and Dinh, V. B., 2014. "Les enjeux de la conception en phase d'esquisse pour les systèmes du génie électrique : illustration sur le cas des systèmes énergétiques pour les bâtiments". In Symposium de Génie Electrique (SGE'14), Cachan, France. See :hal-01024644.

[99] Roozenburg, N., and Eekels, J., 1995. *Product Design : Fundamentals and Methods*. Wiley. isbn :9780471954651.

[100] Galea, A., Borg, J., Grech, A., and Farrugia, P., 2010. "Intelligent Life-oriented Design Solution Space Selection". In International Design Conference (Design 2010), Dubrovnik, Croatia.

[101] Tsang, E., 1993. *Foundations of Constraint Satisfaction*, 1st ed. Academic Press. isbn :9781483220499.

[102] Benhamou, F., and Granvilliers, L., 2006. *Handbook of Constraint Programming*. Elsevier, ch. 16, Continuous and Interval Constraints, pp. 571–604.

[103] Russell, S. J., and Norvig, P., 2010. *Artificial Intelligence : A Modern Approach*, 3rd ed. Pearson. isbn :9780136042594.

[104] Montanari, U., 1974. "Networks of constraints : fundamental properties and applications to picture processing". In Information Science, Vol. 7, pp. 95–132.

[105] Waltz, D., 1972. Generating semantic descriptions from drawings of scenes with shadows. Tech. rep., Cambridge, Massachusetts.

[106] Lauriere, J. L., 1976. "Un langage et un programme pour résoudre et énoncer des problèmes combinatoires : Alice". PhD thesis, Paris 6 University.

[107] Mackworth, A., 1977. "Consistency in networks of relations". In Artificial Intelligence, Vol. 8, pp. 99–118.

[108] Jaffar, J., and Lassez, J.-L., 1987. "Constraint Logic Programming". In Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '87), Munich, West Germany, Association for Computing Machinery, pp. 111–119. doi :10.1145/41625.41635, isbn :0897912152.

[109] Wagner, H. M., 1969. *Principles of operations research, with applications to managerial decisions*. Prentice-Hall.

[110] Vargas, C., Saucier, A., Albert, P., and Yvars, P. A., 1994. "Knowledge Modelisation and Constraint Propagation in a Computer Aided Design System". In Workshop notes Constraint Processing in CAD of the 3rd International Conference on Artificial Intelligence in Design. Lausanne, Switzerland.

[111] Yannou, B., and Harmel, G., 2005. *Advances in Design*. Springer Verlag, ch. 12, Use of Constraint Programming for Design.

[112] Mazhoud, I., Hadj-Hamou, K., Bigeon, J., and Remy, G., 2012. "The Electromagnetic Actuator Design Problem : An Adapted Interval Global Optimization Algorithm". In IEEE Transactions on Magnetics, Vol. 48, pp. 387–390. doi :10.1109/TMAG.2011.2172922.

[113] Trabelsi, H., Yvars, P.-A., Louati, J., and Haddar, M., 2015. "Interval computation and constraint propagation for the optimal design of a compression spring for a linear vehicle suspension system". In Mechanism and Machine Theory, Vol. 84, Elsevier, pp. 67–89. doi : 10.1016/j.mechmachtheory.2014.09.013. See :hal-01587674.

[114] Meyer, Y., and Yvars, P.-A., 2012. "Optimization of a passive structure for active vibration isolation : an interval-computation- and constraint-propagation-based approach". In Engineering Optimization, Vol. 44, pp. 1463–1489. doi :10.1080/0305215X.2011.652102.

[115] Vareilles, E., Gaborit, P., Aldanondo, M., Carbonnel, S., and Steffan, L., 2012. "CoFiADe Constraints Filtering for Aiding Design". In Actes des 8th Journées Francophones de Programmation par Contraintes (JFPC 2012).

[116] Vincent, L., Yvars, P. A., and Millet, D., 2011. "Global optimization of environmental impact by a constraint satisfaction approach - Application to ship-ecodesign". In 18th International Conference on Engineering Design (ICED 11) - Impacting Society Through Engineering Design, Vol. 5, pp. 49–59. isbn :9781904670254.

[117] Tchertchian, N., Yvars, P. A., and Millet, D., 2013. "Benefits and limits of a Constraint Satisfaction Problem/Life Cycle Assessment approach for the ecodesign of complex systems : A case applied to a hybrid passenger ferry". In Journal of Cleaner Production, Vol. 42, Elsevier Ltd, pp. 1–18. doi :10.1016/j.jclepro.2012.10.048.

[118] Ansótegui, C., Béjar, R., Fernàndez, C., and Mateu, C., 2008. "Hard SAT and CSP instances with Expander Graphs". In Proceedings of the 23rd (AAAI) Conference on Artificial Intelligence.

[119] Gent, I. P., Miguel, I., and Nightingale, P., 2008. "Generalised arc consistency for the AllDifferent constraint : An empirical survey". In Artificial Intelligence, Vol. 172, pp. 1973–2000. Special Review Issue, doi : 10.1016/j.artint.2008.10.006, issn :00043702.

[120] Benhamou, F., McAllester, D., and Pascal Van Hentenryck, P., 1994. "CLP(Intervals) Revisited". In International Symposium on Logic Programming, Ithaca, New York, USA, M. Bruynooghe, ed., pp. 124–138.

[121] Gelle, E., and Faltings, B., 2003. "Solving mixed and conditional constraint satisfaction problems". In Constraints. doi :10.1023/A :1022394531132, issn :13837133.

[122] Faltings, B., 1994. "Arc-consistency for continuous variables". In Artificial Intelligence, Vol. 65, p. 363. doi :10.1016/0004-3702(94)90022-1. http ://infoscience.epfl.ch/record/97925.

[123] Haralick, R. M., and Elliott, G. L., 1980. "Increasing tree search efficiency for constraint satisfaction problems". In Artificial Intelligence. doi :10.1016/0004-3702(80)90051-X, issn :00043702.

[124] Brelaz, D., 1979. "New Methods to Color the Vertices of a Graph". In Comm. ACM 22, pp. 251–256.

[125] Appel, K., and Haken, W., 1977. "The Solution of the Four-Color-Map Problem". Vol. 237, Scientific American, a division of Nature America, Inc., pp. 108–121. doi : 10.2307/24953967, issn :00368733.

[126] IEEE/IEC/MSC, 2019. IEEE STD 754 :2019 - IEEE Standard for Floating-Point Arithmetic. Tech. rep., July. doi :10.1109/IEEESTD.2019.8766229, isbn :9781504459242.

[127] Lhomme, O., 1993. "Consistency Techniques for Numeric CSPs". In Proceedings of the 13th International Joint Conference on Artifical Intelligence (IJCAI), Chambéry, France, R. Bajcsy, ed., Vol. 1, Morgan Kaufmann, pp. 232–238. url :http ://ijcai.org/Proceedings/93-1/Papers/033.pdf.

[128] Megiddo, N., 1982. “Linear-Time Algorithms for Linear Programming in R3 and Related Problems”. In Proceedings of Annual Symposium on Foundations of Computer Science, Vol. 12, pp. 759–776. doi :10.1109/sfcs.1982.24, issn :02725428.

[129] Moore, R. E., 1966. *Interval analysis*. Automatic Computation. Englewood Cliffs, N.J., Prentice-Hall.

[130] Andrew, A., 2002. “Applied Interval Analysis : With Examples in Parameter and State Estimation, Robust Control and Robotics”. In Kybernetes, Vol. 31, Emerald Group Publishing Limited. doi :10.1108/k.2002.06731eae.002, issn : 0368492X.

[131] Kearfott, R. B., 1996. “Interval Computations : Introduction, Uses, and Resources”. In Euromath Bulletin, Vol. 2.

[132] Hansen, E., and Walster, G., 2004. *Global optimization using interval analysis*, 2nd ed. Marcel Dekker Edition.

[133] Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J.-F., 1993. “Revising hull and box consistency”. In 16th International Conference on Logic Programming (ICLP).

[134] Collavizza, H., Delobel, F., and Rueher, M., 1999. “Comparing Partial Consistencies”. In Reliable Computing. doi :10.1023/A :1009922003700, issn :13853139.

[135] Kleinrock, L., 1967. “Time-shared Systems : A theoretical treatment”. In Journal of the ACM (JACM). doi :10.1145/321386.321388, issn :1557735X.

[136] Gelle, E., 1998. “On the generation of locally consistent solution spaces in mixed dynamic constraint problems”. PhD Thesis, Ecole polytechnique fédérale de Lausanne (EPFL).

[137] Yvars, P.-A., and Zimmer, L., 2019. “DEPS Studio : Un environnement intégré de modélisation et de résolution de problèmes de conception de systèmes”. In Proceedings of the 8th French Conference on Software Engineering (CIEL’2019).

[138] Bowen, K. A., 1979. “Prolog”. In Proceedings of the 1979 Annual Conference (ACM 1979). doi :10.1145/800177.810020.

[139] Sneyers, J., Van Weert, P., Schrijvers, T., and De Koninck, L., 2010. "As time goes by : Constraint handling rules : A survey of chr research from 1998 to 2007". In Theory and Practice of Logic Programming, Vol. 10, Cambridge University Press, pp. 1–47. doi :10.1017/S1471068409990123.

[140] GUROBI Optimization, I., 2018. "GUROBI Optimizer Reference Manual, Version 5.0". url :https ://www.gurobi.com.

[141] Brook, A., Kendrick, D., and Meeraus, A., 1988. "GAMS, a user's guide". In ACM SIGNUM Newsletter. doi :10.1145/58859.58863, issn :0163-5778.

[142] Byrd, R. H., Nocedal, J., and Waltz, R. A., 2006. "Knitro : An Integrated Package for Nonlinear Optimization". In *Large-Scale Nonlinear Optimization. Nonconvex Optimization and Its Applications*, Vol. 83. Springer, Boston, MA, pp. 35–60. doi :10.1007/0-387-30065_14.

[143] Achterberg, T., 2009. "SCIP : Solving constraint integer programs". In Mathematical Programming Computation, Vol. 1. doi :10.1007/s12532-008-0001-1, issn :18672949.

[144] Drud, A. S., 1994. "CONOPT - A Large-Scale GRG Code". In ORSA Journal on Computing. doi :10.1287/ijoc.6.2.207, issn :0899-1499.

[145] Laborie, P., 2017. "Introduction to CP Optimizer for Scheduling". In International Conference on Automated Planning and Scheduling (ICAPS 2017), Pittsburgh, USA.

[146] https ://ibex-lib.readthedocs.io/en/latest/solver.html.

[147] Gent, I. P., Jefferson, C., and Miguel, I., 2006. "Minion : A fast, scalable, constraint solver". In Frontiers in Artificial Intelligence and Applications. isbn :9781586036423, issn :09226389.

[148] Granvilliers, L., and Benhamou, F., 2006. "Algorithm 852 : RealPaver : An interval solver using constraint satisfaction techniques". In ACM Transactions on Mathematical Software. doi :10.1145/1132973.1132980, issn :00983500.

[149] Zimmer, L., Anglada, A., Christie, M., and Grandvillers, L., 2004. "Constraint Explorer : a modelling and Sizing Tool for Engineering Design". In Invited session on Metamodelling and Constraint Based Problem Solving for Embodiment Design Support Systems in SCI, Orlando, USA.

[150] Kone, A., Nogarède, B., and Lajoie Mazenc, M., 1993. "Le dimensionnement des actionneurs électriques : un problème de programmation non linéaire". In Journal de Physique III, EDP Sciences, pp. 285–301.

[151] Bellman, R., 1956. "Dynamic Programming and Lagrange Multipliers". In Proceedings of the National Academy of Sciences of the United States of America, Vol. 42, National Academy of Sciences, pp. 767–769. doi :10.1073/pnas.42.10.767, issn :0027-8424.

[152] Bertsekas, D. P., 1982. "The Method of Multipliers for Equality Constrained Problems". In *Constrained Optimization and Lagrange Multiplier Methods*. doi :10.1016/b978-0-12-093480-5.50006-4.

[153] Diampovesa, S., Hubert, A., and Yvars, P. A., 2018. "Nouveaux outils de synthèse pour la conception préliminaire et le choix d'architecture : Application à un moteur brushless". In Symposium de Génie Électrique (SGE 2018), Nancy, France.

[154] Davisson, W. I., 1964. "Public Investment Criteria". In Land Economics, Allen and Unwin and MIT Press. doi :10.2307/3144347, issn :00237639.

[155] Eschenauer, H., Koski, J., and Osyczka, A., 1990. *Multicriteria Design Optimization. Procedures and Applications*, 1st ed. Springer-Verlag Berlin Heidelberg. doi :10.1007/978-3-642-48697-5, isbn :9783642486999.

[156] Tran, T. V., 2009. "Problèmes combinatoires et modèles multi-niveaux pour la conception optimale des machines électriques". PhD thesis, École Centrale de Lille, June.

[157] Diampovesa, S., Hubert, A., and Yvars, P. A., 2018. "Optimal Design for Electromagnetic Devices. A Synthesis Approach using Intervals and Constraint-based Methods". In 15th International Workshop on Optimization and Inverse Problems in Electromagnetism (OIPE 2018), Hall in Tirol, Austria.

[158] Diampovesa, S., Hubert, A., Yvars, P. A., Meyer, Y., and Zimmer, L., 2019. "Optimal design for electromagnetic devices : A synthesis approach using intervals and constraint-based methods". In International Journal of Applied Electromagnetics and Mechanics, Vol. 60, IOS Press, pp. 35–48. doi :10.3233/JAE-191104, issn :13835416.

[159] Ben Ayed, R., and Brisset, S., 2012. "Multidisciplinary optimization formulations benefits on space mapping techniques". In International Journal for Computation and Mathematics in Electrical and Electronic Engineering (COMPEL), Vol. 31, pp. 945–957.

[160] http ://www.constraintsolving.com/solvers/constraint-libraries.

[161] http ://labix.org/doc/constraint/.

[162] Jussien, N., Rochart, G., and Lorca, X., 2008. "CHOCO : an Open Source Java Constraint Programming Library". In CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08), Paris, France, pp. 1–10. See :hal-00483090.

[163] Schimpf, J., and Shen, K., 2010. "ECLiPSe - from LP to CLP". In Theory and Practice of Logic Programming (TPLP), Vol. 12 of *Special Issue 1-2 : Prolog Systems*, pp. 127–156. doi :10.1017/S1471068411000469.

[164] Schulte, C., Lagerkvist, M., and Tack, G., 2006. "Gecode : Generic Constraint Development Environment". In INFORMS Annual Meeting 2006. url :https ://www.gecode.org.

[165] Mackworth, A. K., 1992. "The logic of constraint satisfaction". In Artificial Intelligence, Vol. 58, pp. 3–20. doi :10.1016/0004-3702(92)90003-G, issn :00043702.

[166] Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G., 2007. "MiniZinc : Towards a standard CP modelling language". In Lecture Notes in Computer Science. doi :10.1007/978-3-540-749707_38, isbn :3540749691.

[167] Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G., 2007. "MiniZinc : Towards a Standard CP Modelling Language". In Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07), Berlin, Heidelberg, Springer-Verlag, pp. 529–543. isbn :9783540749691.

[168] Frühwirth, T., 1998. "Theory and practice of Constraint Handling Rules". In The Journal of Logic Programming, Vol. 37, pp. 95–138. doi :10.1016/S0743-1066(98)10005-5, issn :0743-1066.

[169] Frühwirth, T., 2015. “Constraint handling rules - what else ?”. In Rule Technologies : Foundations, Tools, and Applications RuleML 2015. Lecture Notes in Computer Science, Vol. 9202, Springer. doi :10.1007/978-3-319-21542-6_2, isbn :9783319215419, issn :16113349.

[170] Michel, L., and Van Hentenryck, P., 2005. “The Comet programming language and system”. In Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP’05). doi :10.1007/11564751_119, isbn :3540292381, issn :03029743.

[171] Barrett, C., and Tinelli, C., 2018. “Satisfiability modulo theories”. In *Handbook of Model Checking*, E. Clarke, T. Henzinger, H. Veith, and R. Bloem, eds. Springer. doi :10.1007/978-3-319-10575-8_11, isbn :9783319105758.

[172] Hnoteey, B., and O’Sullivan, B., 2016. “Introduction to Combinatorial Optimisation in Numberjack”. In Data Mining and Constraint Programming. Lecture Notes in Computer Science, C. Bessiere, L. De Raedt, L. Kotthoff, S. Nijssen, B. O’Sullivan, and D. Pedreschi, eds., Springer. doi :10.1007/978-3-319-50137-6_1.

[173] Akplogan, M., De Givry, S., Métivier, J. P., Quesnel, G., Joannon, A., and Garcia, F., 2013. “Solving the Crop Allocation Problem using Hard and Soft Constraints”. In RAIRO - Operations Research, EDP Sciences, Vol. 47, pp. 151–172. doi :10.1051/ro/2013032, issn :03990559. See :hal-01024641.

[174] Hebrard, E., 2008. “Mistral, a Constraint Satisfaction Library”. In 3rd International CSP Solvers Competition (CPAI08).

[175] Haugen, O., Møller-Pedersen, B., Oldevik, J., Olsen, G. K., and Svendsen, A., 2008. “Adding Standardized Variability to Domain Specific Languages”. In Proceedings of Software Product Line Conference, pp. 139–148.

[176] Eichelberger, H., El-Sharkawy, S., Kröher, C., and Schmid, K., 2015. Integrated variability modeling language : Language specification version 1.27, September.

[177] Jackson, D., Schechter, I., and Shlyahter, H., 2000. “ALCOA : The Alloy Constraint Analyzer”. In International Conference on Software Engineering (ICSE’00), pp. 730–733.

[178] Yvars, P.-A., and L.Zimmer, 2014. “DEPS : Un langage pour la spécification de problèmes de conception de systèmes”. In Proceeding of the 10th International Conference on Modeling, Optimization and SIMulation (MOSIM 2014).

[179] Yvars, P.-A., and Zimmer, L., 2021. “Integration of Constraint Programming and Model-Based Approach for System Synthesis”. In IEEE International Systems Conference (SYSCON 2021), Vancouver, Canada.

[180] Marty, R., 1985. “Object oriented programming”. In Computer Physics Communications. doi :10.1016/0010-4655(85)90084-0, issn :00104655.

[181] Diampovesa, S., Hubert, A., and Yvars, P. A., 2020. “Modélisation d’un problème de conception en vue de réutilisabilité. Illustration par l’exemple d’une batterie Li-ion”. In Symposium de Génie Électrique (SGE 2020-2021), Nantes, France.

[182] Wellstead, P. E., 1979. *Introduction to Physical System Modelling*. Academic Press Ltd. isbn :0127443800.

[183] Nelson, P. A., Gallagher, K. G., Bloom, I., and Dees, D. W., 2019. Modeling the performance and cost of lithium-ion batteries for electric-drive vehicles. Tech. rep., January.

[184] Hongwen, H., Xiong, R., and Fan, J., 2011. “Evaluation of Lithium-Ion Battery Equivalent Circuit Models for State of Charge Estimation by an Experimental Approach”. In Energies, Vol. 4, pp. 582–598. doi :10.3390/en4040582.

[185] Andrade, A., 2012. “Conception intégrée par optimisation multicritère d’un système d’actionnement électrique pour l’aéronautique”. PhD thesis, Institut National Polytechnique (INP) Toulouse.

[186] Andrade, A., Sareni, B., Roboam, X., and Couderc, M., 2013. “Conception intégrée par optimisation multicritère d’un système d’actionnement électrique pour l’aéronautique”. Vol. 16, pp. 221–242. doi :10.3166/ejee.16.221-242.

[187] Langlois, O., 2006. “Conception d’un réseau de secours électrique pour l’aéronautique”. Phd thesis, Institut National Polytechnique (INP) Toulouse.

[188] Notter, D. A., Gauch, M., Widmer, R., Wäger, P., Stamp, A., Zah, R., and Althaus, H. J., 2010. “Contribution of Li-ion batteries to the environmental impact of electric

vehicles". In Environmental Science and Technology, Vol. 44, pp. 6550–6556. doi :10.1021/es903729a, issn :0013936X.

[189] EPA-ORD, 2013. Application of Life- Cycle Assessment to Nanoscale Technology : Lithium-ion Batteries for Electric Vehicles. Tech. rep. National Risk Management Research Laboratory EPA's Office of Research and Development.

[190] Hans, B., and Melin, E., 2019. Analysis of the climate impact of lithium-ion batteries and how to measure it. Tech. rep., July. Commissioned by Transport Environment. 17 p.

[191] Menga, P., and Ceraolo, M., 2008. "An evaluation of global environmental and energy value of vehicle technologies". In Proceedings of the 3rd European Ele-Drive Transportation Conference, Geneva, Switzerland, Vol. 2. doi :10.1.1.623.3115.

[192] Amarakoon, S., Smith, J., and Segal, B., 2013. "Application of Life-Cycle Assessment to Nanoscale Technology : Lithium-ion Batteries for Electric Vehicles". Online. Report/Paper Numbers : EPA 744-R-12-001.

[193] Atkins, M. J., and Koch, C. R., 2003. "A well-to-wheel comparison of several powertrain technologies". In SAE Technical Papers 2003-01-0081. SAE 2003 World Congress Exhibition. doi :10.4271/2003-01-0081, eissn :01487191, issn :26883627. Also in : Advanced Hybrid Vehicle Powertrains 2003-SP-1750.

[194] Woo, J. R., Choi, H., and Ahn, J., 2017. "Well-to-wheel analysis of greenhouse gas emissions for electric vehicles based on electricity generation mix : A global perspective". In Transportation Research Part D : Transport and Environment. doi :10.1016/j.trd.2017.01.005, issn :13619209.

[195] Fritzson, P., 2011. *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. Wiley-IEEE Press, September. doi :10.1002/9781118094259.ch2, isbn :9781118010686, issn :9781118010686.

[196] DOE, 2009. GREET-Greenhouse gases, Regulated Emissions, and Energy use in Transportation. Tech. rep., Transportation Technology R&D Center. Argonne National Laboratory US DOE.

[197] Einhorn, M., Conte, F. V., Kral, C., Niklas, C., Popp, H., and Fleig., J., 2011. "A Modelica Library for Simulation of Elecric Energy Storages". In Proceedings of the 8th Modelica Conference, Dresden, Germany.

[198] Elmqvist, H., 1997. “Modelica - A unified object-oriented language for physical systems modeling”. In Simulation Practice and Theory. doi :10.1016/s0928-4869(97)84257-7, issn :09284869.

[199] Schamai, W., Fritzson, P., Paredis, C., and Pop., A., 2009. “Towards Unified System Modeling and Simulation with ModelicaML : Modeling of Executable Behavior Using Graphical Notations”. In Proceedings of the 7th International Modelica Conference.

[200] Schamai, W., Pohlmann, U., Fritzson, P., Paredis, C., Helle, P., and Strobel, C., 2010. “Execution of UML State Machines Using Modelica”. In 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT), Vol. 47, pp. 1–10.

[201] Paredis, C., Bernard, Y., Burkhart, R., de Koning, H. P., Friedenthal, S., Fritzson, P., Rouquette, N., and Schamai, W., 2010. “An Overview of the SysML-Modelica Transformation Specification”. In 2010 INCOSE International Symposium. doi :10.1002/j.2334-5837.2010.tb01099.x.

[202] Renier, R., and Chenouard, R., 2011. “De SysML à Modelica : Aide à la formalisation de modèles de simulation en conception préliminaire”. In 12ème Colloque National AIP PRIMECA, Le Mont Dore, France. See :hal-00585100.

[203] A. Saltelli, M. Ratto, T. A. F. C. J. C. D. G. M. S., and Tarantola, S., 2008. *Global Sensitivity Analysis : The Primer*. John Wiley Sons.

[204] ISO/IECJTC1/SC7, 2019. ISO/IEC/IEEE 42020 :2019 - Software, Systems and Enterprise - Architecture Processes, July.

[205] Martin, R. C., 2000. *Design principles and design patterns*. url :http ://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf.

[206] ISO/TC207/SC5, 2006. ISO/IEC/IEEE ISO-14040 :2006 Life cycle assessment - ICS : 13.020.10 - Environmental Management 13.020.60 Product life-cycles, July.

[207] Muralikrishna, I. V., and Manickam, V., 2017. “Chapter Five - Life Cycle Assessment”. In *Environmental Management*, I. V. Muralikrishna and V. Manickam, eds. Butterworth-Heinemann, pp. 57–75. isbn :978-0-12-811989-1, doi :10.1016/B978-0-12-811989-1.00005-1.

[208] Dunn, J. B., Gaines, L., Barnes, M., Wang, M., and Sullivan, J., 2012. "Material and energy flows in the materials production, assembly, and end-of-life stages of the automotive lithium-ion battery life cycle".
doi :10.2172/1044525,https ://www.osti.gov/biblio/1044525.

[209] Wang, Q., Xue, M., Lin, B.-L., Lei, Z., and Zhang, Z., 2020. "Well-to-wheel analysis of energy consumption, greenhouse gas and air pollutants emissions of hydrogen fuel cell vehicle in China". In Journal of Cleaner Production, Vol. 25.
doi :10.1016/j.jclepro.2020.123061, issn :09596526.

[210] Gaines, L., 2019. "Profitable Recycling of Low-Cobalt Lithium-Ion Batteries Will Depend on New Process Developments". In One Earth, Vol. 1, Elsevier, pp. 413–415. doi :10.1016/j.oneear.2019.12.001, issn :25903322.

[211] OMG, 2019. Systems Modeling Language (OMG SysML™) v.1.6.
url :http ://uml-sysml.org.

[212] Roques, P., 2016. "MBSE with the ARCADIA Method and the Capella Tool". In 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Toulouse, France. See :hal-01258014.

[213] Weilkiens, T., 2007. "The Pragmatic SYSMOD Approach". In *Systems Engineering with SysML/UML.* doi :10.1016/b978-0-12-374274-2.00002-x.

[214] Friedenthal, S., Izumi, L., and Meilich, A., 2007. "Object-Oriented Systems Engineering Method (OOSEM) applied to Joint Force Projection (JFP), a Lockheed Martin Integrating Concept (LMIC)". In INCOSE International Symposium, Vol. 17, pp. 1471–1491. doi :10.1002/j.2334-5837.2007.tb02961.x.

[215] Paredis, C., and Davies, K., 2011. System Analysis using SysML Parametrics : Current Tools and Best Practices. Online.
url :https ://www.openmodelica.org/images/docs/modprod2011-tutorial/modprod2011-tutorial4-Chris-Paredis-SysML-Parametrics.pdf.

[216] Desfray, P., 2012. Etat de l'art, SysML où en est-on. Online.
url :http ://www.miam.mips.uha.fr/sysml-uha/slides_desfray.pdf.

[217] Meacham, S., Gioulekas, F., and Phalp, K., 2015. "SysML based Design for Variability enabling the Reusability of Legacy Systems towards the support of

Diverse Standard Compliant Implementations or Standard Updates : The Case of IEEE-802.15.6 Standard for e-Health Applications". In SIMUTOOLS 2015. doi : 10.4108/eai.24-8-2015.2261108.

[218] Padilla Gaeta, J. A., 2014. "Modeling and implementing variability in aerospace systems product lines". Master thesis, University of Waterloo.

[219] Ziadi, T., Helouët, L., and Jézéquel, J.-M., 2003. "Towards a UML profile for software product lines in software product-family". In Proceedings of the 5th International Workshop on Product Familly Engineering (PFE-5).
See :hal-00794817.