CAPITALISATION ET TRAITEMENT DES MODELES POUR LA CONCEPTION EN GENIE ELECTRIQUE

Loig Allain

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

½__½__½__½__½__½__½__½__½__½__½__½

THESE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : « Génie électrique »

Préparée au sein du Laboratoire d'Electrotechnique de Grenoble

dans le cadre de l'Ecole Doctorale
« Electronique, Electrotechnique, Automatique, Télécommunication, Signal »

présentée et soutenue publiquement par

Loig ALLAIN

Ingénieur ENSIEG

Le 30 septembre 2003

CAPITALISATION ET TRAITEMENT DES MODELES POUR LA CONCEPTION EN GENIE ELECTRIQUE

Directeur de thèse : Laurent **GERBAUD**

Monsieur	J.P. ROGNON	Président
Messieurs	P. BROCHET	Rapporteur
	H. MOREL	Rapporteur
	J.P. ROGNON	Président
	J.L. SCHANEN	Examinateur
	L. GERBAUD	Directeur de thèse
	C. VAN DER SCHAEGHE	Examinateur

Vous avez à apprendre à rire. Pour atteindre l'humour supérieur, cessez d'abord de vous prendre trop au sérieux.

Hermann Hesse « Le loup des steppes »

Que tu deviennes professeur, savant, ou musicien, aie le respect du "sens", mais ne t'imagine pas qu'il s'enseigne.

Hermann Hesse « Le jeu des perles de verres »

à mes parents et à Delphine

Voici venu le moment de tourner une page, celle de trois années exceptionnelles en apprentissage. A cette occasion, je tiens à remercier toutes les personnes qui ont rendu possible cette expérience.
Epreuve angoissante s'il en est, tant on a peur d'oublier quelqu'un voici les remerciements.

Un seul mot, usé, mais qui brille comme une vieille pièce de monnaie : merci !
P. Neruda

Remerciements

Tout d'abord, je tiens à exprimer ici ma profonde gratitude envers l'ensemble des membres du jury de ma thèse. En particulier, je remercie MM BROCHET et MOREL de l'honneur qu'ils m'ont fait en acceptant d'être rapporteur sur mes travaux ainsi que de l'intérêt qu'ils ont manifesté pour eux-ci. Merci également à M ROGNON pour avoir présidé ce jury. Merci aussi à M SCHANEN d'avoir accepté de lire ce rapport avant d'assister à la soutenance, un acte héroïque.

Je tiens à remercier MM ROGNON et BRUNET qui ont assuré la direction du Laboratoire d'Electrotechnique de Grenoble pendant la durée de ma thèse.

Merci à MM DESTOBBELEIR, GARDIC et BEJEAN qui ont permis que ce travail de thèse se réalise dans le cadre d'une collaboration industrielle.

Merci à M. GERINIERE qui a su s'accommoder d'un sujet pour le moins original et qui en a compris les tenants et aboutissants avec une surprenante rapidité.

Que mes encadrants directs pour ce travail trouvent ici l'expression de toute la gratitude que je ressens envers ces deux hommes exceptionnels. Il m'est difficile de remercier l'un avant l'autre tant leur apport me semblent équivalent. Ces trois années furent pour moi extraordinaires et je les en remercie. J'ai en effet trouvé en chacun d'eux à la fois un maître et un ami. Christian, Laurent trouvez dans ces quelques mots l'expression de toute l'affection que je ressens pour vous.

Merci à toi Laurent pour avoir encadré ces travaux de thèse et pour avoir assuré la direction de l'équipe « Conception et Dimensionnement Intégré » pendant presque 2 ans. J'espère que nous nous retrouverons bientôt, peut être pour une virée d'escalade, pour une mousse au chocolat ou plus simplement pour parler de la pluie et du beau temps. J'ai trouvé en ta personne un encadrant de qualité et un « chef » (si, il te faut bien l'accepter un jour) exceptionnel d'humanité. Merci à toi pour la confiance que tu m'as accordée au cours de ces 3 années.

Merci aussi à Christian, avec qui je vais avoir l'occasion de continuer la route pendant quelques années encore. Toi aussi, je te remercie de la confiance que tu m'as témoignée et j'espère que tu ne le regretteras pas, je t'expliquerai bien sûr MAEL, les boules et les boites. Je pense que peu de personne ont cette chance de commencer leur vie professionnelle auprès d'une personne aussi sympathique et généreuse. Nous avons maintenant quelques années pour taquiner le tapis vert en espérant que les billes aient plus souvent envie de se rencontrer. Je te remercie de l'honneur que tu m'as fait en acceptant de travailler avec moi et en me faisant partager ton immense connaissance en électrotechnique, il me faudra sans doute une ou deux éternités pour acquérir ton sens de la pédagogie.

Ces trois années furent riches en évènements et pour moi en apprentissages de toutes sortes. Tous ne furent pas agréables mais tous m'ont enrichi. Je tiens à remercier ici MM BIGEON et ATIENZA pour ce qu'ils m'ont fait découvrir de la nature humaine. L'honnêteté intellectuelle et affective me commande de les remercier pour les discussions que nous avons eues et que nous continuerons, j'espère, à avoir. Qu'ils restent assurés de trouver en ma personne un ami, ou à défaut un professionnel de confiance : « L'amitié commande parfois de dire à l'ami qu'il se trompe ». Certaines leçons sont plus dures à apprendre quand elles viennent d'amis, je n'aurais pas voulu qu'elles viennent d'ailleurs.

Je serais injuste si je ne remerciais pas l'ensemble de l'équipe CDI pour les discussions que nous avons pu avoir. Le débat « syntaxe » et « sémantique » restera longtemps gravé dans ma mémoire ainsi que les « réunions composants » de Frédéric.

Merci à Frédéric WURTZ pour ses éclairages sur les épistémologies positiviste et constructiviste. Un exposé brillant qui remet à leur place les scientifiques de l'absolu. Je te souhaite de continuer dans ta recherche et de t'y épanouir encore longtemps. Attention quand même à ne pas toujours retomber sur l'épistémologie. L'humain doit rester au cœur de l'acte de conception, mais aussi au cœur des relations.

Merci à Jaime FANDINO, petit bonhomme humain, parfois trop. Je te souhaite de t'en sortir avec ces petites machines « pourries » au fonctionnement complètement « tordu ». Bon courage pour cette seconde thèse et contacte moi pour ton HDR, je tiens vraiment à la voir. Trouve dans ces quelques mots l'expression de ma reconnaissance pour les discussions que nous avons eues et pour ce sens de l'humain que tu m'as fait partager, sache aussi que tu trouveras toujours en moi un ami et une oreille si tu en as besoin.

Merci également à Alain BOLOPION qui nous a réjoui du récit de séjours à l'autre bout du monde et a apporté un peu de rêve sur la plate-forme. Merci à toi pour ta générosité naturelle et bon courage continue de voyager.

N'oublions pas l'ensemble du personnel du laboratoire pour l'ambiance chaleureuse qu'ils y font régner. Un grand merci à Patrick pour son dévouement total à une cause noble : l'administration système du réseau. Sans lui et son équipe de courageux Vincent et Corinne, personne ne pourrait faire avancer la science, le papier est certes pratique mais souvent insuffisant.

Que l'ensemble des espèces vivant sur la plate forme trouvent ici mes remerciements les plus sincères et mes vœux de réussite dans leurs entreprises. David ami de longue date et que j'ai eu l'occasion d'apprécier plus encore au cours de ces 2 années passées cote à cote. Je te souhaite bon courage pour la fin de cette thèse qui avait si brillamment commencé. Nous continuerons à nous voir et j'espère que la solution trouvée pourra être menée à son terme. Franck, dit « Le Franck » ou encore « Francky » ou encore « Kiki » ou encore « Franck », bon courage à toi, avec de la persévérance tu pourras parler sans que Laule ne te coupe la parole et tu feras émerger la problématique de ta thèse avant la fin de cette année. Salut à toi Laurent dit « Laule » ou « Le Laule », merci pour ta bonne humeur, ta candeur et ta naïveté qui nous procurent toujours autant de sujets de conversation et de fous rires qu'au jour de ton arrivée. Ne change rien, tu es génial comme tu es. Merci aussi à Vincent, dit « Le Vince » ou « Duke » ou « Oh la et ho coquin » ou encore « prends de l'angle » ou « met les gaz » ou …, merci pour ton aide inestimable dans le développement de RAMA, pour tes idées efficaces et

originales dans le développement de code. Apprends à parler moins fort et tu iras loin (si tu prends le départ maintenant sans hésiter à mettre un peu de gaz). Bon courage aussi au petit nouveau qui a hérité de la maintenance et de l'extension des outils développés au cours de cette thèse, bon courage Bertrand, tu en auras bien besoin ;). Bon courage aussi à Hichem et merci pour son travail, bienvenue dans l'équipe.
Un grand merci aux anciens pour leur générosité. Merci à toi Max, je m'entraîne à « balistic » avant notre prochaine rencontre et je te battrai un jour. Merci au Grand Ben, un pilier de la salle PC, 1m90 (ou plus, vu d'en bas j'ai jamais su) de bonne humeur de poilade et de déconne, un personnage génial et généreux, je t'attends sur Annecy ou ailleurs où tu seras toujours le bienvenu.
Que Jean-Mich trouve ici toute ma gratitude et l'expression de la plus sincère amitié. Elle est née de nombreuses discussions allant de la guerre en Irak (« ils font n'importe quoi ces américains ») à des sujets plus proches et aux retombées plus immédiates pour nous deux (« à ton avis, je fais un héritage ou je fais un tableau ? Tu fais comment pour faire une sphère dans 3DS ? Fais moi voir les screen de Trainz ! ») et issues de passions communes que j'ignorais. D'autant qu'il avait le bon goût d'avoir les mêmes opinions que moi ce qui est la preuve d'une grande perspicacité ;).
8h00, c'est l'heure du café et de l'arrivée de l'infatigable Bertrand, merci pour toutes les discussions que nous avons pu avoir autour de café, merci pour les jeux de mots et les casse-tête.

Enfin, « last but not least », Ben, dit « Le Ben » ou « P'tit Ben », un ami d'une valeur exceptionnelle. Je ne regrette qu'une seule chose de ces trois années avec toi : la fin. Nos chemins professionnels divergent et nous ne nous verrons plus aussi souvent (au boulot). Mais ma maison restera toujours ouverte pour toi, Astrid et le reste de ta famille. Je n'ai toujours pas pu apprendre ton calme, ta souplesse de caractère et ta concision (tu vois même dans les remerciements je dépasse les trois pages), j'en aurais pourtant bien besoin. Voici les quelques maigres mots insuffisants pour parler de cette amitié.

Merci aussi à tous les thésards du LEG qui en font le plus grand Laboratoire de l'univers connu et inconnu ! Tous les thésards trouvent ici mes remerciements qui vont aussi bien au « p'tits jeunes » Christian, Franck, Jean-Paul, Manuella, Malik et les autres qu'aux anciens Guillaume (bon courage et lâche pas l'affaire) Bertrand, Yvan, Guillaume, Raphael, Sébastien G, Damien et tous les autres.

Merci aussi à toutes celles et tous ceux que j'ai pu fréquenter au cours de ces années et dont les discussions m'ont beaucoup apporté ainsi « Le Doc' » de Cluses, Serge B, David, Marc, Alain, Stéphane, Anne-Sophie et tous les autres devraient se reconnaître ici.

Pour finir, ces remerciements n'auraient pas de valeur sans une pensée pour mes parents et toute ma famille qui ont su me soutenir, me remonter le moral et me supporter pendant ces longues années. Qu'ils trouvent ici ma reconnaissance éternelle, je commence aujourd'hui seulement à comprendre ce que je leur dois.

Merci enfin et surtout à celle qui partage ma vie depuis un certain nombre d'année. Delphine, « sache que je », et n'oublie pas. Merci de m'avoir supporté encouragé, cajolé, fait descendre de mon arbre. Merci pour tout et pour le reste aussi.

Merci à Henri le « Tonton Tomate » pour toutes les discussions passionnantes que nous avons pu avoir et à sa femme la si gentille Lucette qui se mêle de tout, un jour peut être je suivrai vos conseils. Merci à Alba pour son honnêteté et sa candide franchise.

En conclusion, merci à toi lecteur d'être arrivé ici et bon courage pour la suite. Tu verras c'est passionnant il y est question de boules et de boites ainsi que manipulations formelles autant de mots pour décrire des choses dont je ne suis pas l'unique auteur, Laurent en est également coupable.

Sommaire

Tables des Figures

CHAPITRE 1 :

CHAPITRE 2 :

CHAPITRE 3 :

CHAPITRE 4 :

CHAPITRE 5 :

Glossaire

Glossaire

A.M.C.C. Association Machine Convertisseur statique Commande

Algorithme Jeu d'instructions informatiques préprogrammées. Par exemple une loi de commande ou des tests sur des variables.

Bloc Elément de modélisation de type boite noire. Utilisable dans un contexte logiciel spécifique avec d'autres blocs pour construire une description plus complexe

Boite Modèle de type boite blanche dont les différentes grandeurs sont typées (Paramètres physiques, Entrées, Sorties, …) dont le contenu est accessible

Boite blanche Format de capitalisation pour lequel l'ensemble de la connaissance est accessible et modifiable par l'utilisateur

Boite grise Format de capitalisation des modèles pour lequel la connaissance est dans un format peu accessible ou programmé. Par exemple, le code source d'une S-Function

Boite noire Format de capitalisation des modèles pour lequel la connaissance n'est ni accessible ni modifiable. Par exemple, S-Function compilées de MATLAB/SIMULINK

Boule Modèle de type boite blanche dont les différentes grandeurs ne sont pas typées dont le contenu est accessible

Capitalisation Conservation de la connaissance, ici la capitalisation est la conservation des modéles pour les réutiliser, les modifier ou les contextualiser.

COB Acronyme pour Computationnal OBject. Composant informatique de calcul pour le dimensionnement utilisé dans la suite Pro@Design

Composant Objet informatique décrit indépendamment d'un contexte logiciel. Autonome, il est défini par un certain nombre d'interfaces qui permettent de le piloter

Contextualisation Ensemble des actions réalisées pour transformer une boule en boite.

Entrée En simulation temporelle, grandeur supposée variable en fonction du temps. Elle fournit une information à une boite de simulation. Par exemple, une source ou une commande sont des entrées

Excitation Dans le cas de la description d'un système d'état, l'excitation est le vecteur des entrées du système d'état (vecteur U)

Grandeur d'état Grandeur variable et continue dans le temps. Ce sont des grandeurs qui apparaissent dérivées par rapport au temps dans la description d'un modèle, par exemple, la tension aux bornes d'un condensateur, ou la position d'un mobile.

HAGEL Acronyme pour Hybrid Automaton for Generation of Equation and Logic : générateur de code basé sur la définition de patrons

IHM Interface Homme Machine, ensemble graphique qui permet à un utilisateur de piloter un logiciel

MAEL Acronyme pour Module d'Analyse des Equations et de la Logique : environnement logiciel pour l'édition, la capitalisation et traitement des modèles boules et boites blanches.

MOO Modélisation Orientée Objet

Netbeans Ref : http://www.netbeans.org
Environnement de développement modulaire donc le socle graphique est exploité dans MAEL

Paramètre numérique Grandeur utilisée pour paramétrer des algorithmes de calcul numériques. Cela peut être, par exemple, une précision à atteindre pour un calcul d'intégrale

Paramètre physique En simulation temporelle, grandeur considérée comme indépendante du temps. C'est une caractéristique de la structure. Ainsi la longueur du paquet de fer d'une machine asynchrone est un paramètre physique.

Patron Description schématique d'un code à écrire ou à générer pour un outil spécifique. Les squelettes sont construits à partir des spécifications logicielles de l'outil de calcul.

Projection Ensemble des activités qui permettent de générer un code de calcul informatique pour un environnement choisi.

RAMA Acronyme pour Rule Applicator for Mathematic Analysis : outil léger utilisé pour le calcul formel et fondé sur l'utilisation de règles

Simulation mixte Simulation temporelle qui mixte les systèmes discrets et les systèmes continus

Sortie En simulation temporelle, grandeur supposée variable en fonction du temps. Elle est évaluée dans une boite et sortie afin d'être utilisée dans d'autres boites en tant qu'entrée.

Squelette Voir patron.

Système hybride Les systèmes hybrides sont constitués d'une liste de « sous-systèmes » qui décrivent chacun des états continus qui s'enchainent les uns les autres selon certaines conditions

XML eXtended Markup Language. Langage à balises extensibles, utilisé entre

autre pour la sauvegarde structurée des données

Introduction

Introduction

Les acteurs de la conception en Génie Electrique manipulent des modèles macroscopiques, que ce soit pour la simulation de structures complexes associant des Machines avec des Convertisseurs Statiques et des Algorithmes de Commande, ou pour le dimensionnement des actionneurs. La capitalisation de ces modèles est donc un enjeu important pour les outils d'aide à la conception.

Outre la multiplication des modèles et la complexification des applications qu'il doit traiter, le concepteur se trouve confronté à la multiplication des outils informatiques pour la résolution de ses problèmes. Celle-ci se justifie par la complémentarité des analyses accessibles grâce à ces outils, mais s'accompagne de l'obligation de produire des modèles spécifiques pour ces outils ; ce qui pose d'évidents problèmes de gestion de la cohérence entre ces modèles et de perte de temps.

De récents progrès dans les sciences informatiques, et en particulier l'utilisation de plus en plus courante des composants informatiques, ont permis le développement d'environnements d'optimisation modulaires. Aujourd'hui, l'augmentation de la puissance du matériel informatique rend possible le dimensionnement contraint pour des structures complexes en intégrant notamment des notions temporelles au sein des critères dimensionnants de la structure.

Pour répondre à ces problèmes, nous nous proposons de fournir une aide au concepteur tant dans la mise au point de ses modèles pour des outils différents que dans la capitalisation de ses modèles. Afin de le soulager des activités fastidieuses et productrices d'erreurs de la manipulation formelle des modèles et d'écriture de code, nous lui proposons une démarche structurée et automatisée de transformation de ses représentations mathématiques.

Dans le chapitre 1, après avoir succinctement présenté notre vision de l'activité de conception fondée sur l'utilisation de modèles macroscopiques, nous présentons les outils actuels de la modélisation de structure pour le dimensionnement et pour la simulation en insistant sur leurs spécificités du point de vue du concepteur. Nous présentons rapidement la Modélisation Orientée Objet, une approche récente de la modélisation de structure multi-physique utilisée essentiellement pour la simulation.

Dans le chapitre 2, nous proposons une méthodologie de travail pour la modélisation et le dimensionnement des structures multi-physique. Cette méthode de travail est proposée pour guider le concepteur dans son activité de modélisation pour la simulation puis le dimensionnement contraint d'Association Machine Convertisseur Commande (AMCC) prenant en compte des contraintes issues de l'analyse de simulation temporelle (valeur maximale de courant, démarrage, …). Cette approche s'appuie fortement sur l'automatisation des traitements de modèles décrits au chapitre 3 et sur les outils développés au chapitre 4.

Dans le chapitre 3, nous proposons un double formalisme pour la capitalisation de la connaissance : les boules, représentation hors contexte des modèles, et les boites, représentation orientée des modèles. Nous proposons également un processus automatisé de transformation des représentations qui permet de passer des boules aux boites. Ce chapitre se termine en présentant la problématique de la génération automatique de code comme solution à la multiplication des environnements de calcul dédiés.

Dans le chapitre 4, nous présentons MAEL (Modul for Analysis of Equations and Logic), une implantation informatique des concepts développés dans les deux chapitres précédents. Après avoir fait un bilan des besoins de l'outil, nous proposons une architecture informatique qui présente deux niveaux d'utilisation en fonction des compétences de l'intervenant : concepteur utilisateur de l'outil informatique ou bien expert de l'outil informatique. Deux modules de l'outil sont présentés au travers de l'utilitaire RAMA (Rule Applicator for Mathematic Analysis) pour les traitements formels et du module HAGEL (Hybrid Automaton for Generation of Equations and Logic) pour la génération de code.

Le chapitre 5 présente deux exemples d'utilisation des outils et de mise en œuvre des méthodologies présentées pour le dimensionnement et la simulation d'actionneurs électromécaniques. Ce qui nous permet d'effectuer un choix circonstancié de structure.

Enfin nous concluons par un bilan et des perspectives de notre travail.

Chapitre 1

Modélisation, simulation et dimensionnement

Chapitre 1 Modélisation, simulation et dimensionnement

A - Contexte : La conception d'actionneurs électromécaniques

A - 1 Contexte Applicatif

Le contexte applicatif de l'étude est celui de la conception (simulation, dimensionnement) d'application du type Association Machine – Convertisseur – Commande (A.M.C.C.) destinée à des applications de vitesse variable. La démarche suivie dans cette optique sera également applicable au dimensionnement d'actionneur électromagnétique (par exemple, des électro-aimants). Ce besoin apparaît dans le contexte des automatismes de l'habitat, en réponse à une demande du marché qui désire des mouvements visuellement plus « souples » pour les actionneurs.

Ainsi, l'objectif de l'étude est de fournir les moyens méthodologiques et les outils pour la description et le traitement des modèles du concepteur afin de lui permettre de construire son argumentation pour le choix des actionneurs permettant d'obtenir une fonctionnalité particulière, par exemple, la vitesse variable. La structure choisie doit remplir un certain nombre de contraintes de performances et d'encombrement. Dans le cas qui nous intéresse – l'entraînement de volets roulants – les contraintes que l'actionneur doit remplir sont issues de l'environnement très particulier dans lequel il est appelé à évoluer. Il est, en effet, intégré dans un environnement confiné, le tube autour duquel le volet vient s'enrouler (Figure 1. 1). De cette particularité viennent non seulement des contraintes de volume et d'encombrement mais également des contraintes d'échauffement limité et de rendement à maximiser.

Figure 1. 1 : Entrainement tubulaire à dimensionner

Dans le cas général du dimensionnement de structure, la majorité des critères sont construits à l'aide de modèles macroscopiques ou approchés. La construction de ces modèles n'est pas toujours possible, c'est par exemple le cas de l'évaluation des pertes dans certains convertisseurs statiques de puissance [LAR]. Dans ces cas particuliers, il s'avère nécessaire de simuler le fonctionnement dynamique de la structure pour accéder au critère dimensionnant voulu.

Afin de répondre aux questions qui se posent à nous dans ce contexte, nous devrons donc appréhender des représentations a priori différentes qui permettent de construire une information différente : la simulation temporelle et des représentations de régimes permanents ou statiques.

Dans le même temps, l'étude que nous menons doit pouvoir permettre une ré-exploitation de la méthodologie proposée à d'autres problèmes qui pourraient apparaître.

A - 2 Contexte méthodologique

A - 2 - 1 L'activité de conception

Nous n'étudions pas, ici, la conception sous son aspect d'innovation. Nous posons comme postulat de base que le concepteur dispose d'une certaine technologie déjà plus ou moins éprouvée lorsqu'il se pose un nouveau problème. Sa démarche a pour objectif de choisir l'« assemblage » de « briques techniques » le plus pertinent pour répondre à son désir. Ainsi, il existe de nombreuses solutions techniques pour créer la fonction de vitesse variable (Figure 1. 2)

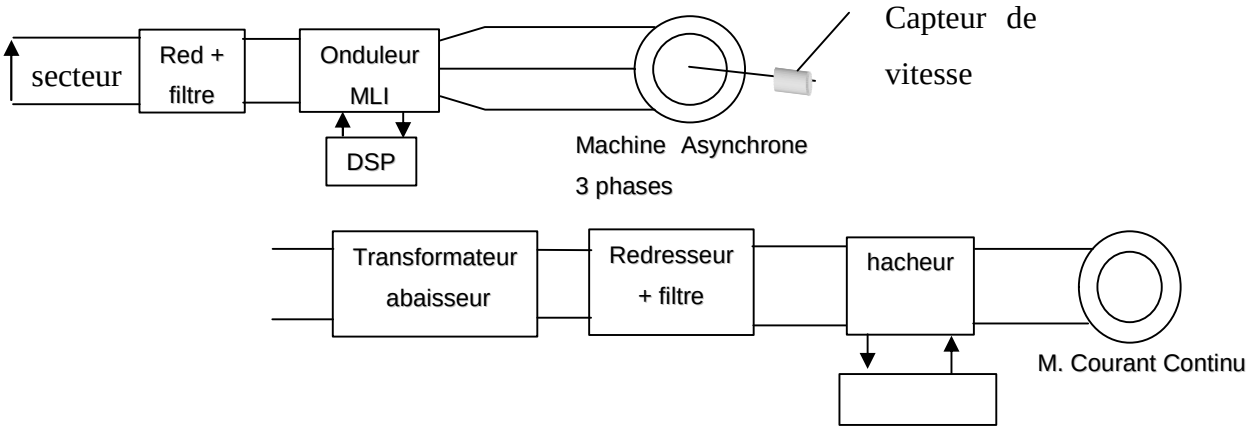


Figure 1. 2 : Exemples d'association permettant de réaliser la vitesse variable

Sur la Figure 1. 3, nous présentons le cycle de conception tel que nous le percevons dans cette étude. On remarquera que cette représentation pourrait être acceptable pour la conception de nombreux systèmes, autres que les systèmes du génie électrique :

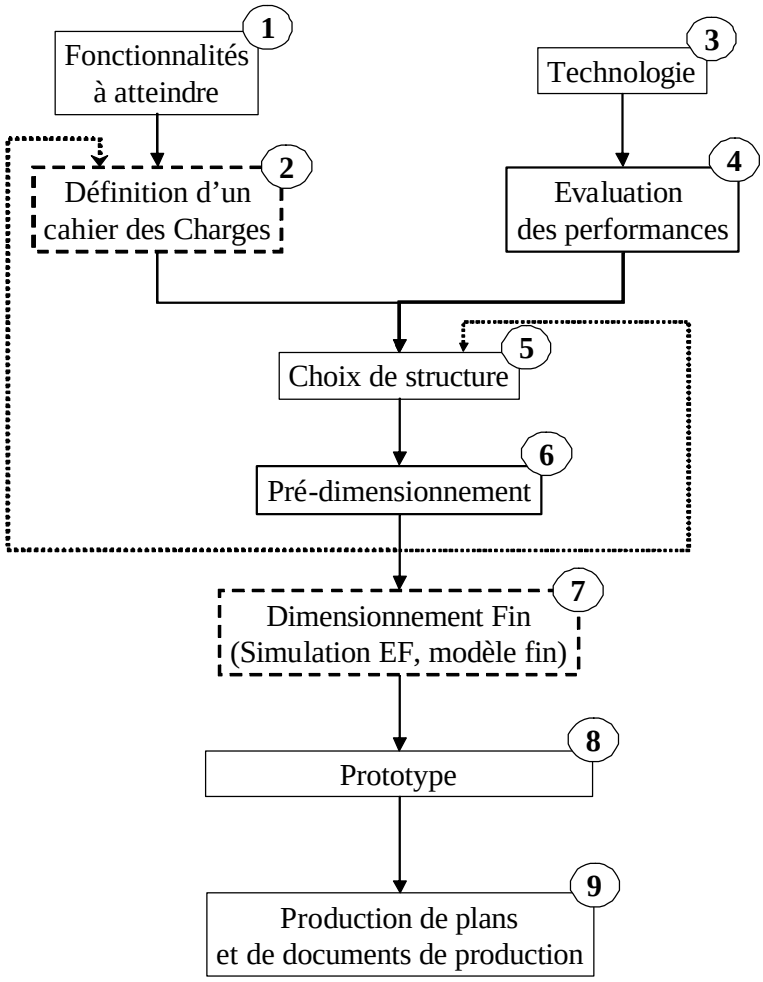


Figure 1. 3 : Le processus de conception

Lors d'une action de conception de structure, le concepteur dispose d'un point de départ constitué de deux éléments :

- une fonction à remplir (**1**), c'est-à-dire un objectif que son dispositif doit atteindre lors de son fonctionnement (dans le cas qui nous intéresse la vitesse variable)
- la technologie (**3**) : c'est-à-dire un ensemble d'éléments techniques qui, assemblés, réalisent la fonction désirée. La technologie est constituée par l'ensemble des solutions techniques qui existent à un moment donné.

Partant d'un problème de conception : l'ajout ou la création d'une fonctionnalité, le concepteur évalue les possibilités que la technologie lui apporte (**4**) par le biais des performances des solutions étudiées. Cette évaluation a un double impact, à la fois sur la fonctionnalité elle-même mais aussi sur la définition d'un cahier des charges. En effet, l'évaluation de la technologie peut le pousser à remettre en cause la fonctionnalité qu'il désire ajouter, ou au contraire lui ouvrir de nouveaux horizons. Par ailleurs cette étude, guidée par les fonctionnalités à atteindre, lui permet de définir un cahier des charges de l'applicatif à

créer (**2**). Le cahier des charges constitue, ici, la traduction en terme d'encombrement, de performances, de structures d'une fonction à remplir.

Par exemple, « faire de la vitesse variable dans un tube d'enroulement » peut être traduit par :

- une limitation de l'encombrement dans un tube de diamètre 0.08m.
- la possibilité d'avoir deux régimes de vitesse.
- exercer un contrôle sur la vitesse de l'actionneur à 10%.

La mise en regard du « cahier des charges » et des performances de chacune des solutions techniques lui permet alors de faire un choix de structure plus précis (**5**). Le résultat de ce choix est parfois constitué d'un ensemble de solutions techniques susceptibles de remplir le cahier des charges initial.

L'étape suivante est le pré dimensionnement de la, ou des, structures (**6**). Lors de cette phase du processus de conception, le concepteur réalise une étude grossière de son dispositif, il détermine les dimensions de son système. Parfois il remet en cause le cahier des charges initial. L'issu de cette étape doit être un choix d'une structure prédimensionnée, dont les performances seront ensuite analysées plus finement.

Cette analyse fine est l'objet de la phase suivante, l'utilisation de modèles lourds de simulation et de calcul est alors courante pour permettre d'affiner le dimensionnement de la structure (**7**), ou de valider des points litigieux de son fonctionnement, par exemple les courants maximums dans les composants de puissance.

Les étapes suivantes, qui ne sont pas l'objet de notre étude, mènent à la mise en production du dispositif : réalisation de prototypes (**8**), production des plans, et mises au point des méthodes de production ainsi que de l'outil qui sert cette dernière (**9**).

La représentation du processus de conception est volontairement simplifiée et ne montre pas toutes les remises en cause possibles lors des phases finales de la conception. Dans un cas idéal de conception, plus le concepteur est avancé dans le processus moins les remises en causes agissent sur le cahier des charges initial, notamment en raison des coûts que ces remises en cause induisent.

Enfin, signalons que, dans ce projet, nous ne travaillons que sur les aspects de l'évaluation des performances et du pré-dimensionnement. Par ailleurs, nous verrons également que certains aspects de notre travail peuvent être exploités pour le dimensionnement fin des structures.

A - 3 Une activité « multi »

A l'image de la conception, le processus de conception que nous détaillons est multiple :

- par la diversité des domaines abordés
- par le nombre d'acteurs concernés
- par la diversité des outils et des langages utilisés

A - 3 - 1 Multi-physique

L'activité de conception d'un ensemble tel que celui qui nous intéresse, nécessite des compétences multiples : mécanique, thermique, électronique de puissance, électromagnétisme et économie. Ainsi, les outils et méthodes mis à disposition du concepteur doivent permettre l'appréhension de ces domaines différents.

A - 3 - 2 Multi-acteurs

Dans le cas idéal, bien que l'activité de conception soit multi-physique, l'ensemble de ces compétences est détenu par un individu unique qui est alors à même d'évaluer la globalité du système. Malheureusement, les domaines couverts sont tellement variés que cela n'est que rarement le cas, ce qui complexifie l'activité de conception. Par ailleurs, la conception d'un produit met de plus en plus souvent en présence les différents acteurs de sa réalisation. Les sous-traitants deviennent des acteurs de la conception d'un produit, apportant leurs connaissances des capacités de leur appareil de production ; ils participent activement à l'élaboration d'une solution technique. Il découle de cela que l'activité de conception est une activité multi-acteurs et que, outre la différence des domaines abordés, il faut prendre en compte le fait que cette activité est le fruit du dialogue entre des personnes d'horizons divers, de cultures différentes.

A - 3 - 3 Multi-outils

La principale conséquence de cette diversité d'horizons, de formations et de moyens est une multiplication des outils utilisés pour les calculs. Suivant les points qui intéressent le concepteur, il utilise des outils différents, informatiques ou non. Ainsi, par exemple, pour le dimensionnement d'une machine asynchrone, il est possible d'utiliser

- soit un outil de calcul simple (TK-SOLVER [TKS], ou MATH CAD [MATH]) qui permet de connaître les paramètres du schéma équivalent de la machine en fonction de ses caractéristiques géométriques

- soit un outil de simulation du type MATLAB/SIMULINK [M/S] ou SABER [SAB] pour évaluer le comportement temporel de l'actionneur.

Par ailleurs, les habitudes et les formations diverses poussent les concepteurs à utiliser des outils différents pour un objectif d'étude commun.
L'activité de la conception est donc une activité multi-outils.

A - 3 - 4 Multi-langage

Conséquence directe de cette multiplication des acteurs et des outils, la multiplication des modes de représentation d'un problème de conception en fait une activité multi-langage. Ainsi, bien que travaillant autour d'un produit commun, les acteurs de la conception se trouvent confrontés à un problème de cohérence de la représentation de leur connaissance.
Pour un même objectif d'étude,par exemple, la caractérisation statique des structures, les logiciels MATH CAD[MATH], Pro@Design[PRO] et TK-Solver[TKS] utilisent des modes de description et de stockage de l'information différents. Chacun de ces outils ayant des capacités différentes, il est nécessaire de disposer du modèle du dispositif dans chaque environnement : les passerelles étant généralement inexistantes, le choix d'un outil se fait souvent à l'exclusion des autres.
Le même conflit apparaît lorsque l'on s'intéresse à la simulation temporelle des dispositifs, MATLAB/SIMULINK[M/S], SABER [SAB] ou GENTIANE/ORO [NOR] utilisent des modes de représentation du modèle différents.

A - 3 - 5 La capitalisation

Par ailleurs, l'ensemble des entreprises et des communautés scientifiques se trouve aujourd'hui confronté à un problème de stabilité de la connaissance. En effet, la pyramide des âges d'une part, et d'autre part l'accélération du « turn-over » dans les entreprises posent des problème de pérennité de l'expertise.
Il ressort l'existence d'un réel besoin de capitalisation de la connaissance. Un système de **gestion de ressources de modèles** dans un bureau d'étude est nécessaire. Or, comment répondre à cet enjeu alors qu'au sein d'un même bureau d'étude, le savoir-faire est éclaté et représenté sous des formes diverses, plus ou moins incompatibles entre elles ?

A - 4 Les entités de modélisation

A - 4 - 1 Contenu des modèles

L'activité du concepteur, telle que nous la présentons ici, s'appuie sur une donnée essentielle : les modèles des structures. Ils sont au cœur du travail du concepteur, nous détaillons donc leur constitution.

La problématique de dimensionnement d'actionneurs, telles que nous l'abordons dans ce travail, fait intervenir deux aspects de la modélisation :

- la modélisation statique, par exemple, pour les caractéristiques géométriques des actionneurs ou pour l'évaluation des performances statiques des actionneurs.
- la modélisation dynamique, par exemple, pour la modélisation des pertes en commutation des convertisseurs statiques, pour l'évaluation des valeurs crêtes de courant

Pour plus de clarté, dans la suite de ce rapport nous distinguerons le **calcul** de la **simulation**. Le terme **calcul** est utilisé pour désigner les activités de calculs qui ne font pas intervenir de simulation temporelle. Par exemple, les évaluations des paramètres du schéma équivalent d'une machine asynchrone sont des calculs.

Par opposition, la **simulation** est une activité qui fait intervenir le temps dans la résolution des équations. Par exemple, l'évaluation de l'évolution temporelle du courant en ligne d'un convertisseur ressort de la simulation.

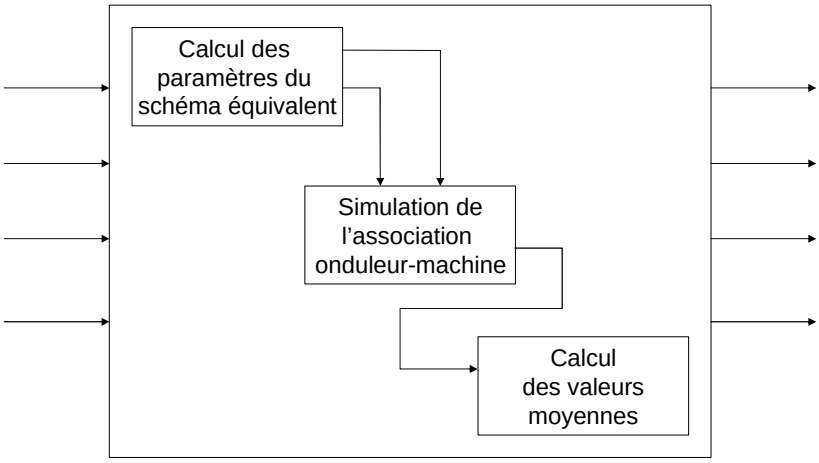


Figure 1. 4: Exemple de problème de dimensionnement incluant la simulation

Les modèles utilisés sont composés :

- d'équations (ou relations) qui traduisent une réalité physique sous une forme mathématique

- d'algorithmes, qui représentent une réalité physique plus complexe à traduire. Ils sont, par exemple, utilisés pour donner le modèle d'une commande dans le cas de la simulation temporelle, ou pour représenter un modèle statique complexe (calculs de pertes [LAR])
- de grandeurs, qui sont les éléments descriptifs des systèmes modélisés et qui sont mises en relation par le biais des équations ou des algorithmes pour représenter le comportement du système
- de tableaux de données, utilisés pour traduire des courbes de saturation, des équivalences entre les diamètres des fils de cuivre nus ou isolés
- de fonctions, qui sont des artifices mathématiques utilisés dans la description des modèles
- de tests, pour décrire des modèles mixtes en simulation, ou des modèles multi-structures en optimisation.

***Remarque** : l'approche que nous présentons ici considère que les simulations temporelles, et en particulier les analyses menées sur leurs résultats sont des éléments de la modélisation pour le dimensionnement.*

A - 4 - 2 La nature des modèles utilisés

Les modèles que nous manipulons ne prennent pas en compte toute la complexité de la physique : ce sont des modèles macroscopiques. Nous les utilisons pour représenter des relations qualitatives plutôt que quantitatives entre les grandeurs. Ces modèles sont construits à partir d'une connaissance du fonctionnement du dispositif et utilisent un jeu d'hypothèses pour alléger leur représentation. L'utilisation de ces hypothèses implique la construction d'un domaine de validité du modèle.

2 - a) Les modèles analytiques pour le dimensionnement

Ces modèles sont utilisés dans une activité de dimensionnement de structure, où le concepteur détermine les valeurs des différentes grandeurs physiques de son dispositif. Cette activité peut se faire :

- soit « à la main », l'expert connaissant le comportement de son dispositif saura trouver un jeu de valeurs qui fera que son dispositif répondra à un cahier des charges défini au début de l'activité globale de conception, cette méthode est aussi connue sous le nom de « essai – erreur ».

- soit automatiquement en couplant un algorithme d'optimisation sous contrainte et un modèle [ATI].

Le développement des outils de dimensionnement de structure n'est pas un problème nouveau en soit. Il a déjà été traité de multiples manières, que ce soit par les études universitaires [WUR], [ATI] [JAN] ou par les éditeurs de logiciels [ISI] [EPO]. Des solutions diverses ont déjà été proposées au nombre desquelles se trouvent les techniques d'optimisation sous contraintes. Ces dernières utilisent des algorithmes d'optimisation :

- soit déterministes (par exemple, du type SQP)
- soit stochastiques (par exemple génétique).

Tous utilisent le même type d'information pour la recherche de la solution :

- un modèle du dispositif qui définit l'espace de recherche en reliant les différentes grandeurs du dimensionnement
- des contraintes posées sur ces grandeurs pour définir les bornes de l'espace de recherche.

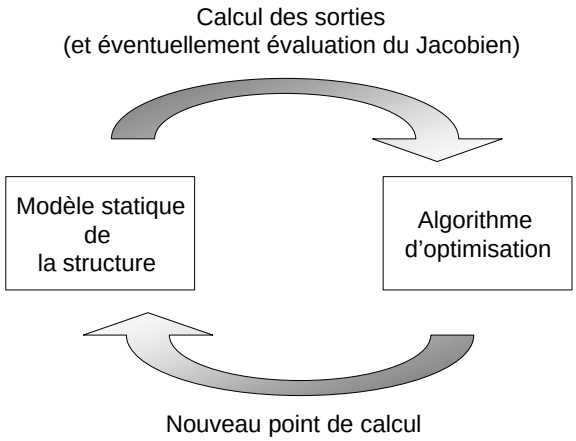


Figure 1. 5: Environnement d'optimisation

La Figure 1. 5 illustre l'approche utilisée dans ces cas. Le modèle de dimensionnement est chargé de faire un calcul de ses grandeurs de sorties pour un jeu de grandeurs d'entrées imposées par l'environnement réalisant l'optimisation. Le logiciel d'optimisation détermine alors un nouveau jeu pour les valeurs d'entrées et ainsi de suite dans un processus itératif qui permet de trouver un jeu de paramètres d'entrées tels que le dispositif réponde au cahier des charges.

Dans ce cas d'utilisation, étant donné le grand nombre d'évaluations nécessaires, les modèles manipulés doivent être rapides et souples d'utilisation. Ainsi, notons que les modèles sont essentiellement statiques pour des raisons de rapidité de calcul essentiellement.

Remarque : *Il existe aujourd'hui des outils de calculs intégrant ces fonctionnalités. Dans certains, le concepteur décrit son modèle sous la forme d'équations symboliques [PRO], [MATH]. Pour d'autres logiciels, le concepteur doit créer un code de calcul qui respecte un certain nombre d'interface pour la communication avec le logiciel [ISI], [EPO] ou [M/S].*

2 - b) Les modèles de simulation

Il est parfois nécessaire d'introduire dans la description macroscopique du dispositif une simulation temporelle. Celle-ci permet, par exemple, de construire une image des pertes en commutation du convertisseur. Une telle modélisation est souvent mixte, c'est-à-dire qu'elle mixe des phases continues et discrètes. C'est par exemple le cas des AMCC.

Parfois, pour les systèmes d'état, une modélisation hybride est utilisée, les descriptions en « topologie variable » de convertisseurs d électronique de puissance [GER-1] en sont un exemple. Nous donnons sur la Figure 1. 6 l'exemple de la description d'un hacheur série. La description de ce convertisseur fait apparaître 3 états continus, le choix du système actif à un instant t est effectué par le biais d'un algorithme (ici une séquence de tests) qui évalue les états des différents interrupteurs et en déduit le schéma équivalent à résoudre tout en gérant les transitions au niveau des valeurs des grandeurs (par exemple, la remise à zéro de certains courants).

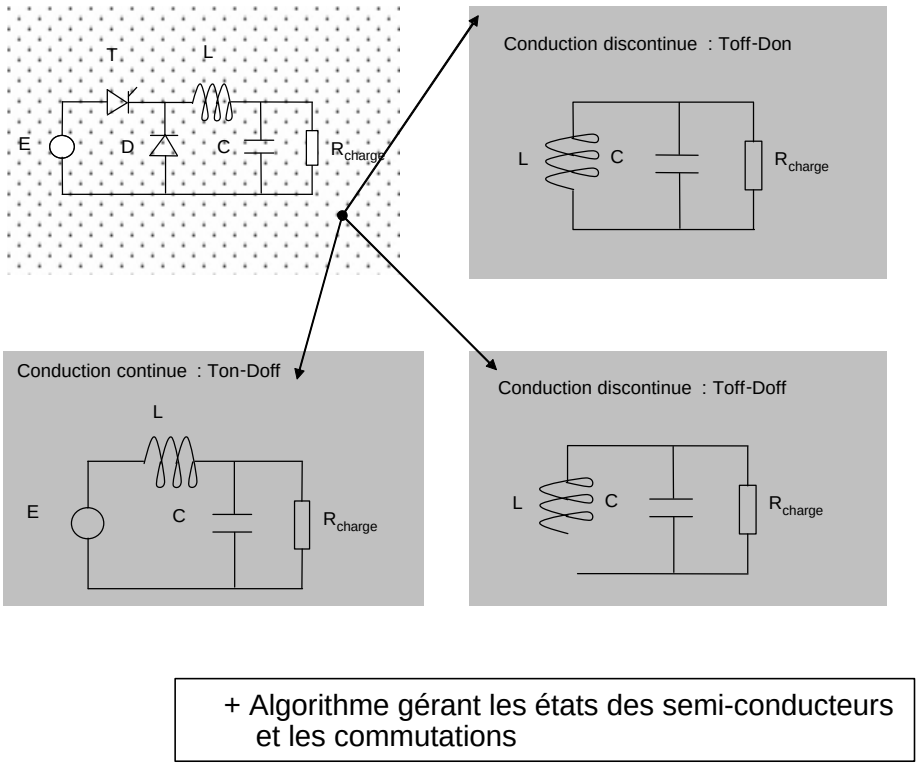


Figure 1. 6: Réprésentation en topologie variable d'un hâcheur série

A - 4 - 3 La création et la mise au point des modèles macroscopiques

Il est parfois possible de trouver des modèles macroscopiques de dispositifs utilisables pour le dimensionnement ou la simulation. Mais l'expert ne dispose pas toujours de l'information nécessaire à leur création et leur mise au point. Nous proposons ici (Figure 1. 7) un rappel de la méthodologie proposée dans [WUR] pour la mise au point des modèles macroscopiques utilisés dans une optique de dimensionnement par optimisation.

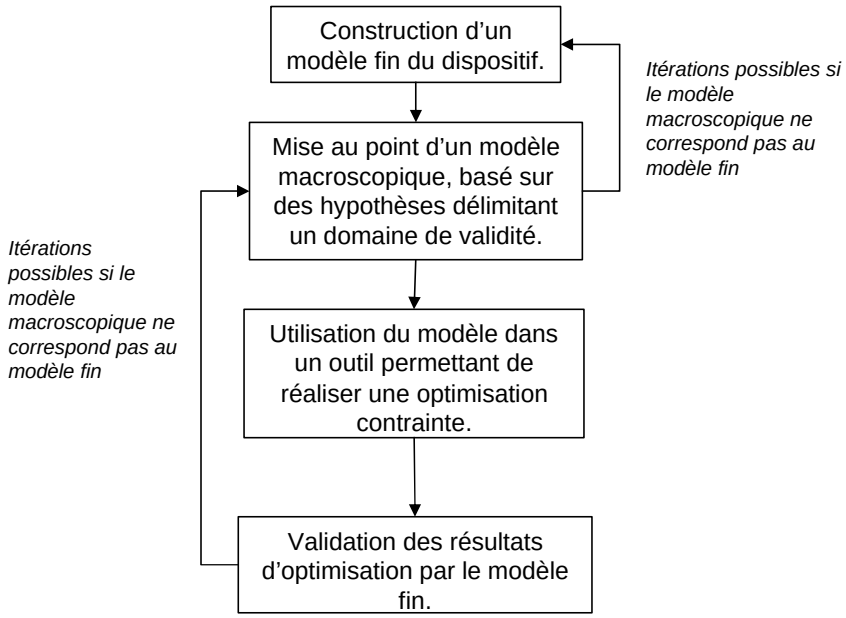


Figure 1. 7: Processus de création d'un modèle macroscopique pour le dimensionnement

Cette approche envisagée pour la modélisation de structures en régime statique est également valable pour la modélisation en régime transitoire.

B - Modélisation, simulation et dimensionnement de structures.

B - 1 Modélisation et simulation de structures.

L'accroissement de la puissance de calcul au cours des dernières années et dans les années à venir rend accessible l'encapsulation de simulations temporelles dans une optique de dimensionnement. Il n'en reste pas moins que ces simulations doivent être rapides et utilisées avec parcimonie. Il est donc nécessaire de prévoir le développement de modèles de simulations dédiés au dimensionnement.

Dans ce cadre, les questions suivantes se posent naturellement : Comment intégrer une simulation dans une optimisation ? Quel outil utiliser ?

B - 1 - 1 <u>Les outils de simulation temporelle</u>

1 - a) Architecture des outils de simulation

La grande majorité des outils de simulation peut être classifiée en fonction des possibilités d'interaction qu'ils laissent à l'utilisateur. Ces possibilités sont à relier à l'architecture de tels outils (Figure 1. 8).

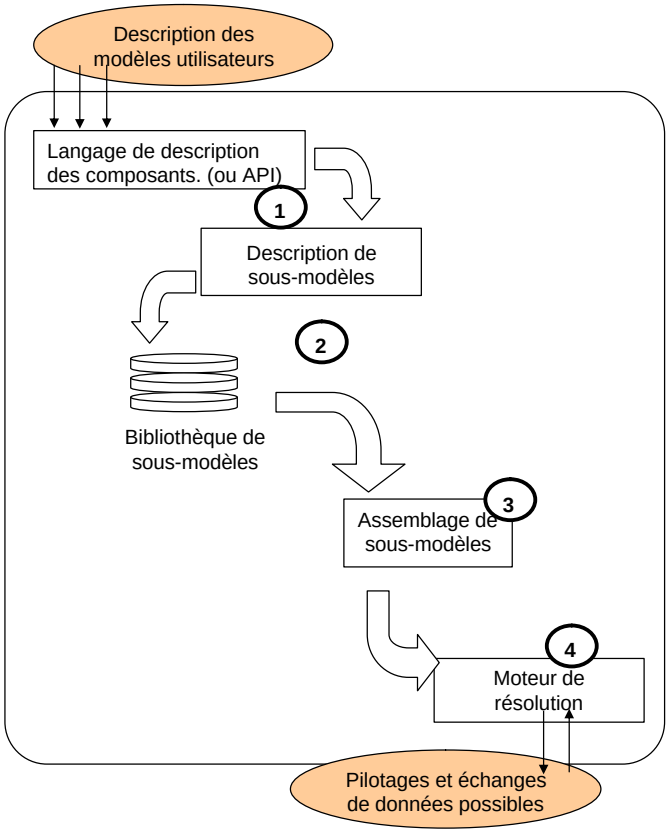


Figure 1. 8 : Architecture des outils de simulation temporelle

En effet, les outils de simulation temporelle sont construits autour d'un moteur de résolution **(4)**, il s'agit de l'élément central de la simulation qui permet de définir les enchaînements des calculs entre les différents éléments de la simulation. La plupart de ces méthodes d'intégration numériques utilisent des méthodes « pas à pas ».

Suivant le degré d'ouverture, le concepteur peut choisir ou non le moteur de résolution du problème différentiel, il lui est également possible de le « piloter ». C'est-à-dire qu'il est possible, par le biais de fichiers de commande ou d'interfaces logicielles, de lui donner des instructions depuis un environnement externe. Il s'agit aujourd'hui d'une pratique de plus en plus courante, pour des raisons commerciales la plupart du temps. Ainsi, MATLAB, SABER, P-Spice ou GENTIANE/ORO [NOR], [GER-2] offrent la possibilité d'un pilotage externe.

Les outils de simulation, pour des raisons de convivialité et de réutilisation des modèles développés, utilisent des interfaces graphiques pour décrire une structure à simuler comme un assemblage des sous-modèles. Cet assemblage des sous-modèles **(3)** peut se faire de différentes façons :

- par le biais d'un équivalent de « Net-List »
- textuellement (GENTIANE-ORO)

- graphiquement (MATLAB/SIMULINK), SABER, SIMPLORER[SIM], DYMOLA [DYM](Figure 1. 9).

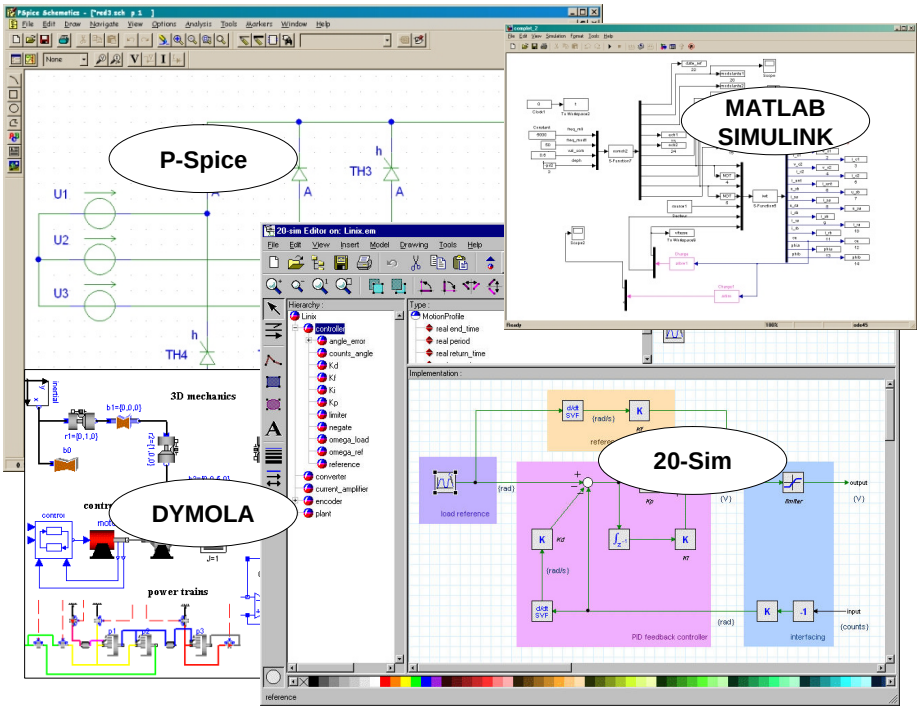


Figure 1. 9: Exemples de description visuelle d'assemblage de composants

Les sous-modèles sont choisis par le concepteur dans une bibliothèque **(2)** de sous-modèles. Cette dernière peut être générale (MATLAB/SIMULINK) ou dédiée à un métier spécifique (SABER, P-Spice, SIMPLORER, PowerSim de MATLAB).

Enfin, un utilisateur, qui ne trouve pas satisfaction dans les bibliothèques de sous-modèles, a la possibilité de créer ses propres modèles **(1).** Cette description peut se faire de trois manières :

- soit en respectant un « contrat » (des interfaces), l'utilisateur doit alors produire un code informatique parfois ardu à réaliser. Par exemple, il est possible de créer des blocs de simulation pour MATLAB/SIMULINK en langage C, ADA ou FORTRAN.
- soit en utilisant un langage de description proposé par les éditeurs. On peut citer VHDL-AMS [VHDL] pour SIMPLORER ou MAST pour SABER, MEX pour MATLAB. Ces langages sont généralement des simplifications de langages informatiques.
- soit par assemblage de composants pour faire des macro-composants.

1 - b) Langages et formalismes de modélisation pour la simulation

Parallèlement au développement des outils de simulations tels que nous venons de le présenter, les dernières décennies ont connu une véritable explosion de langage et de formalisme pour la description des problèmes de simulation. Ces langages et formalismes répondent à des besoins particuliers pour des domaines spécifiques.

Par exemple, VHDL-AMS, utilisé dans les domaines de la microélectronique et des microsystèmes pour la conception de processeurs ou d'architectures électroniques, répond à un réel besoin de ce domaine en introduisant dans la création des modèles, la possibilité de décrire l'architecture de la structure modélisée. Ce langage est suffisamment générique pour permettre la description de systèmes multi-physiques, mais son inconvénient est la complexité d'un langage multi-représentation (comportement, architecture, …).

Les langages de Modélisation Orientée Objet, présentée au paragraphe C de ce chapitre ont connu une réelle explosion avec l'émergence des langages tels que Modelica [ELM-1][ELM-2][MOD], le ?-Language[VAN][FAB] ou encore YASHM [THE].

Mais des formalismes de description multi-physiques ont également connu un développement important et suscite un intérêt croissant. Ainsi, le formalisme de modélisation par Bond-Graph [KAR][ALL] permet de représenter des structures multi-physiques et complexes en s'intéressant à la nature des transferts énergétiques entre les éléments de cette structure.

B - 2 Le dimensionnement de structures intégrant de la simulation

B - 2 - 1 Simulation et dimensionnement des structures

Parmi les outils de simulation existants, certains permettent le dimensionnement de structures:

- soit, en interne, c'est-à-dire qu'une fonctionnalité de dimensionnement est intégrée dans l'outil
- soit, en externe, c'est-à-dire que l'optimisation est réalisée par un outil extérieur à l'outil de simulation lui-même. Cet outil extérieur pilote le logiciel de simulation pour en récupérer les résultats de simulation.

Dans les deux cas, la philosophie du dimensionnement est la même, l'optimisation de la structure est un processus consommateur des résultats d'une simulation temporelle.

B - 2 - 2 Dimensionnement en interne

Pour la majorité de ces outils, le dimensionnement n'est qu'une possibilité rajoutée « après coup » aux fonctionnalités du logiciel. Ainsi, par exemple, l'outil de simulation d'électronique

P-Spice propose un module de dimensionnement, qui permet de définir les caractéristiques des composants du circuit afin que le fonctionnement de ce dernier corresponde à un cahier des charges. Le principe de cette approche est illustré sur la Figure 1. 10.

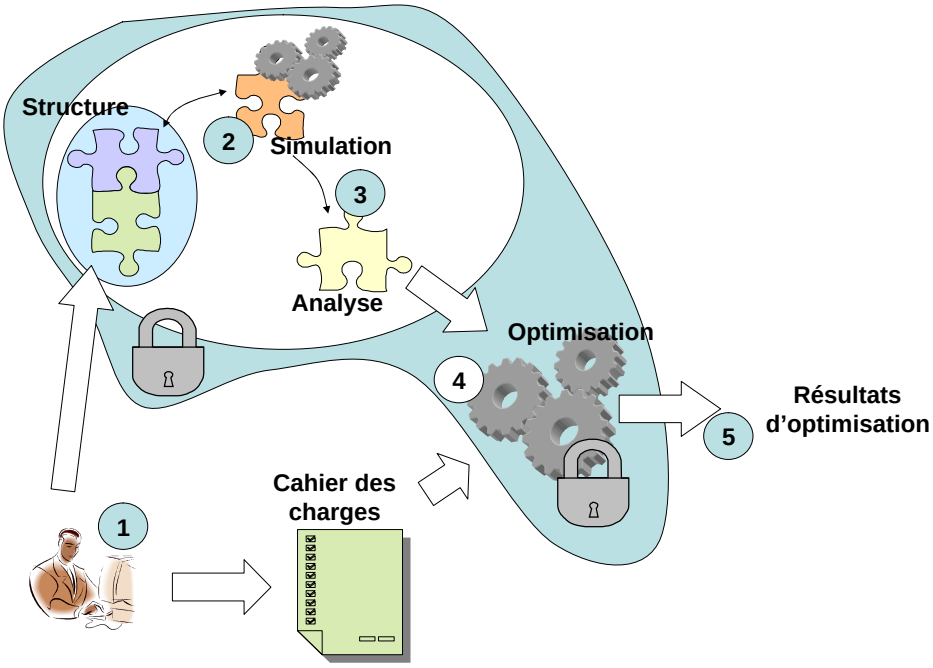


Figure 1. 10:Simulation encapsulée dans l'optimisation

Le concepteur décrit la structure à dimensionner et un cahier des charges pour l'activité d'optimisation **(1)**. L'exécution de la procédure d'optimisation s'appuie sur les résultats donnés par le moteur de simulation et filtrés par l'analyse. Dans ce cas, l'algorithme d'optimisation est imposé de même que le moteur de simulation. Dans cette solution d'optimisation, le modèle définissant l'espace de recherche pour l'optimisation est constitué par :

- la structure à optimiser
- le moteur de simulation de l'outil **(2)**
- les analyses offertes par l'outil, évaluation des valeurs moyennes, efficaces, maximales, d'ondulation…**(3)**

Cette approche n'est nullement satisfaisante dans le cadre du dimensionnement des actionneurs électromécaniques, en effet, il existe d'autres caractéristiques du dispositif qui ne sont pas atteignables par cette approche. De plus le nombre de paramètres optimisables est limité (8 pour P-Spice) ce qui n'est pas suffisant.

B - 2 - 3 Dimensionnement en externe

Dans cette approche, l'outil de simulation et l'outil d'optimisation sont indépendants. Le second pilote le premier pour récupérer les résultats de la simulation qui sont analysés pour en extraire les grandeurs dimensionnantes [KRA].

Cette autre solution pour l'optimisation de structures est illustrée en Figure 1. 11.

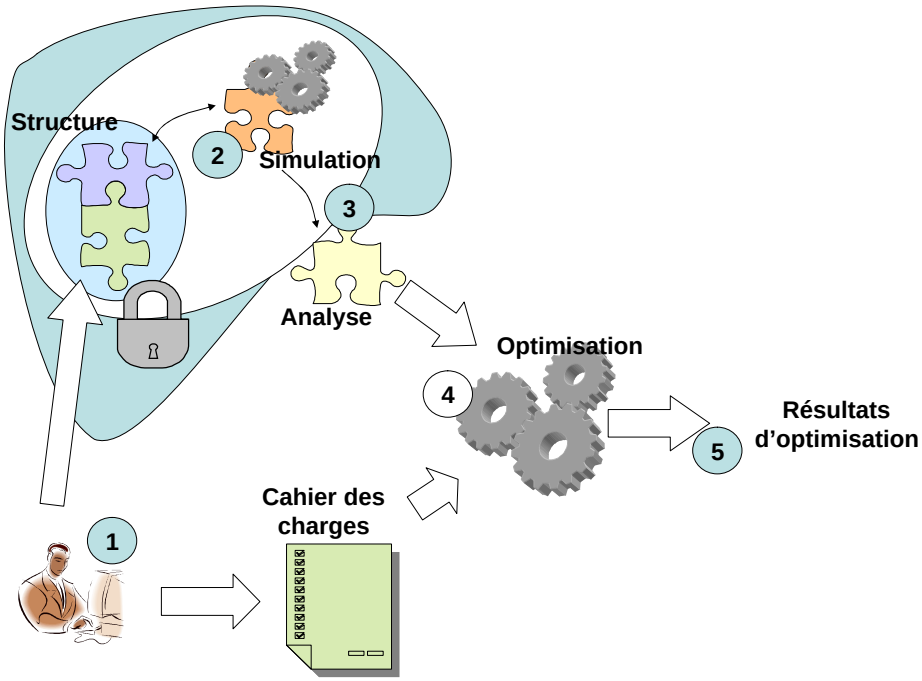


Figure 1. 11: Simulation et optimisation dans des outils différents

Dans ce cas, le modèle du dispositif est constitué de la simulation temporelle de ce dernier, que le concepteur n'a pas la possibilité de modifier.

L'avantage de ces 2 approches réside essentiellement dans la facilité de description des structures à dimensionner. De plus, pour le dimensionnement en interne, le concepteur dispose d'une interface commune pour la description des structures en simulation et en dimensionnement.

B - 3 Bilan des outils de simulation

Le recours à la simulation dans l'optimisation de système est souvent long et décrit un espace exploratoire restreint en raison de problèmes de convergence numérique importants, nous éviterons donc autant que possible les simulations temporelles dans un contexte d'optimisation. Nous leur préférons un modèle plus grossier, mais aussi plus rapide et dont les capacités exploratoires sont plus importantes.

Nous présentons, un comparatif non exhaustif des outils de simulation les plus usités dans notre domaine dans le tableau 1.

Tableau 1: Comparatif de quelques outils de simulation temporel

Outil	Moteur ouvert	Pilotable (API ou fichier)	Langage de description des composants	Contrat d'écriture des composants	Bibliothèques de composants	Interface de description des assemblages	Optimisation
MATLAB	**V** Choix résolution	**V** API et Fichier	**V** langage outil	**V**			**V** Interne
MATLAB/ SIMULINK	**V**	**V** API et Fichier, ActiveX	**V** langage outil	**V** S-Function	**V** génériques et métiers	**V** Graphique	via MATLAB ou externe
SABER	**V**	**V** API et fichier	**V** MAST		**V** Métier	**V** Graphique	**V** Interne/ Externe
P-Spice	**V** S-Function	**V** API	**V**	**V**	**V** Métier	**V** Graphique et texte (Net-List)	**V** Interne
GENTIANE/ ORO	**V**	**V** API et fichier		**V**	Par application dédiée	**V** Textuel	**V** Externe
CADENCE	**V**	**V**	**V**	**V**	**V**	**V**	
ECOSIM-PRO		**V** ActiveX	**V**	**V**	**V** génériques et métiers	**V** Graphique et texte	
20-Sim	**V** générations multiples	**V** API	**V** Bond Graph		**V** génériques et métiers	**V** Graphique	
Simplorer	**V**	**V** API	**V** VHDL-AMS	**V**	**V** métiers	**V** Graphique	**V** Interne

***Remarque** : En grisé nous marquons les informations que nous n'avons pas pu établir de manière certaine.*

C - La modélisation Orientée Objet (MOO)

C - 1 Qu'est ce que la Modélisation Orientée Objet ?

Apparue au début des années 1990, la Modélisation Orientée Objet (MOO)[ELM-1] est le pendant de la Programmation Orientée Objet. La MOO permet de structurer la description des modèles physiques de la même façon que la POO a permis d'assurer une structuration au développement informatique. Par la structuration en « objets » permettant de créer des entités de bases de la modélisation, l'information est capitalisée sous une forme intelligible. De plus, la prise en compte des notions d'héritage et d'agrégation permet d'enrichir la base de connaissance par itérations successives.

C - 1 - 1 L'héritage

Comme pour le développement Objet, il est possible de créer un modèle nouveau à partir d'un modèle existant. En effet, il est possible d'hériter d'un modèle pour rajouter ou surcharger une information nouvelle.

Ceci se traduit par l'exemple suivant, qui montre le modèle de la résistance d'un conducteur, illustré sur la Figure 1. 12 :

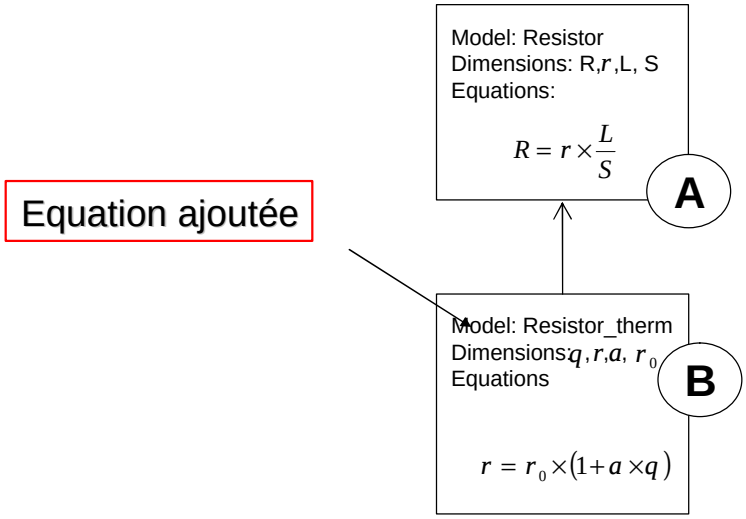


Figure 1. 12: Exemple d'héritage - Description de résistances

Le modèle A existe dans la base de connaissance et décrit la résistance d'un conducteur en fonction de sa longueur, sa section et sa résistivité. Si le concepteur a besoin d'une résistance dont la valeur est une fonction de la température, il décrit le modèle B, qui héritant du modèle A, n'a pas besoin de redéfinir la résistance d'un fil. En revanche, il faut rajouter une relation qui permet de prendre en compte la variation de la température.

Dans l'exemple précédent, nous avons enrichi la description d'un modèle par l'ajout d'une équation. Il est également possible de modifier une description en remplaçant une équation par une autre, comme ceci est illustré sur la Figure 1. 13. Ceci s'apparente à la surcharge d'opérations de la Programmation Orientée Objet.

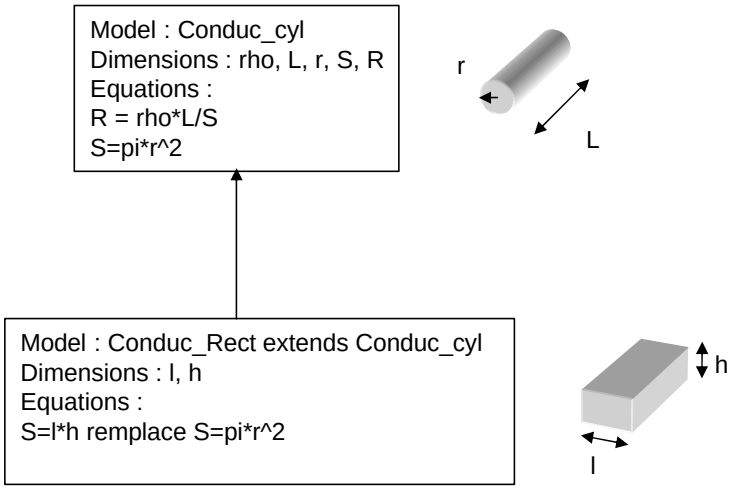


Figure 1. 13: Exemple de modification d'un modèle par surcharge de la relation permettant de connaitre la section d'un conducteur

Dans cet exemple, la base de données des modèles permet de modifier la résistance d'un conducteur en modifiant la relation permettant de lui imposer une section.

Outre la structuration naturelle de la connaissance imposée par une utilisation pertinente des ces technologies, cela permet au concepteur de construire rapidement un nouveau modèle par modifications sur un modèle existant.

C - 1 - 2 L'agrégation

L'agrégation est un moyen de construire un modèle complexe par assemblage de modèles élémentaires existants dans la base de données des modèles. Le concepteur dispose d'un ensemble de « briques » élémentaires de modélisation dont l'utilisation lui permet de construire un modèle plus complexe sans se préoccuper de la modélisation des briques élémentaires.

La Figure 1. 14 montre un exemple de composition dans le cas de la construction du modèle d'une association Machine à courant continu – Hacheur.

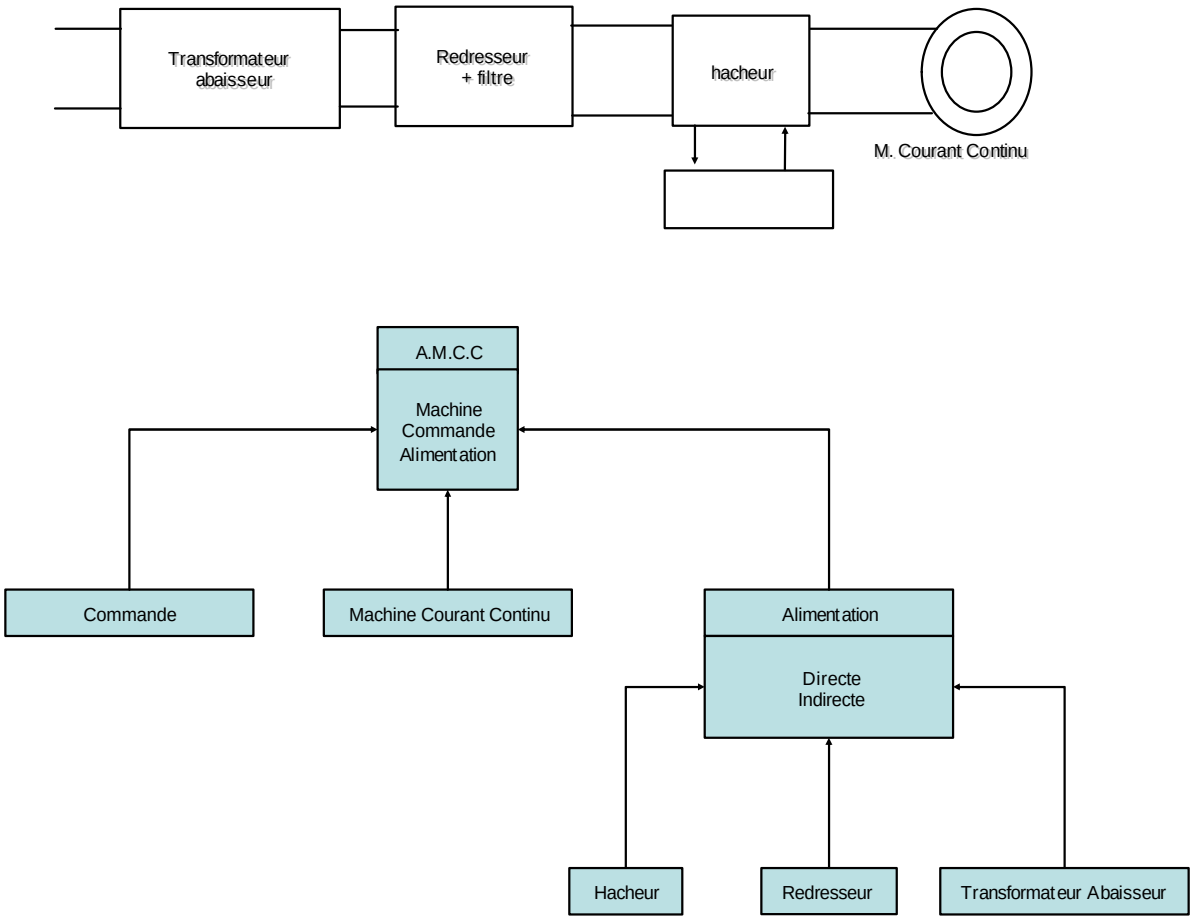


Figure 1. 14: Exemple d'AMCC décrite dans un langage MOO

A la fin de son activité de modélisation, le concepteur a fait croître la base de données des modèles. De plus, il dispose de briques élémentaires auxquelles il peut aisément accéder pour les modifier. Il n'est pas nécessaire d'appréhender la complexité de la représentation de l'ensemble de l'A.M.C.C. pour modifier, par exemple, une relation caractéristique de la machine.

L'utilisation actuelle de ces techniques de modélisation se développe dans les domaines de la simulation temporelle [ELM-1], [FAB], [THE]. Mais, à notre connaissance, ces techniques ne sont pas utilisées dans un domaine autre que celui de la simulation temporelle et en particulier pas dans le domaine du dimensionnement.
Il est tout à fait possible d'étendre ces modes de représentation et de structuration à un contexte de dimensionnement de structures.

C - 2 Les outils et langages de la MOO

Les concepts de la Modélisation Orientée Objet disposent maintenant d'une histoire qui a donné naissance à un certain nombre d'implémentations informatiques.
Au nombre de ces outils, Dymola [DYM] qui repose sur le langage Modelica [MOD] permet la simulation de dispositifs multi-physiques [ELM-2]. L'outil de modélisation permet l'édition des modèles élémentaires par le biais d'un langage de description. Il est également possible de décrire les agrégations de modèles en utilisant un module de description des compositions sous forme graphique.

D'autres outils et langages ont une finalité de simulation de processus industriel, comme c'est le cas pour le langage YAHSM développé au Laboratoire d'Automatique de Grenoble. Il reprend les principes de bases de la modélisation orientée objet et permet la construction de la description d'un procédé batch [THE].

Enfin, des langages tels que le ?-language permettent la simulation de dispositifs multi-physique pour la mise en œuvre de procédés, par exemple [VAN][FAB].

D - La génération de code – Une passerelle entre les descriptions

Que ce soit dans une optique de dimensionnement ou pour la simulation de structure, il est nécessaire de disposer d'un code informatique qui permette d'intégrer les calculs numériques des modèles dans un environnement.
Certaines approches, telles que celle proposée par MATLAB (les S-Functions) ou par EPOGY pour le dimensionnement de structure, imposent au concepteur de produire le code informatique en C ou dans un langage de programmation avancé (C++, ADA ou FORTRAN) pour profiter de l'environnement. En revanche, d'autres suites logicielles proposent au concepteur de saisir leurs modèles dans un langage qui n'est pas celui qui sera utilisé pour les évaluations numériques. Dans ce dernier cas, il est nécessaire de disposer d'un système permettant la création du code effectivement utilisé : un générateur de code.
Des outils tels que Dymola, Pro@Design, ECOSIM-PRO [ECO], GENTIANE-ARMOISE [GER-3], ou 20-Sim [20S] (outils de modélisation des « graphes de liens » ou Bond-Graph) disposent de leurs propres structures de génération de code. Ces systèmes sont dédiés à un objectif de création de code et ne permettent pas la création de code pour d'autres outils à l'exception de GENTIANE et 20-Sim qui par leur architecture rendent possible la génération de code vers d'autres langages, actuellement les S-Functions de MATLAB/SIMULINK.

Il existe par ailleurs des outils permettant de contribuer à la génération de code dans un langage dédié, par exemple MACSYMA ou encore MAPLE permettent de supporter la génération d'un code C.

Malgré quelques passerelles développées de manières occasionnelles entre les différents environnements, il n'existe pas à l'heure actuelle de standard de description. De sorte que le concepteur doit souvent développer plusieurs fois le même modèle pour les différents outils de simulation ou de calcul qu'il utilise au cours du processus de conception. Il s'ensuit des problèmes de :

- cohérence entre les différents modèles
- temps perdu dans la mise au point des modèles
- répétition d'une tâche non productive
- pérennité des modèles décrits par rapport aux outils

Le concepteur se trouve confronté au choix « définitif » d'un outil à l'exclusion des autres alors que son activité nécessite l'utilisation d'autres environnements (Figure 1. 15).

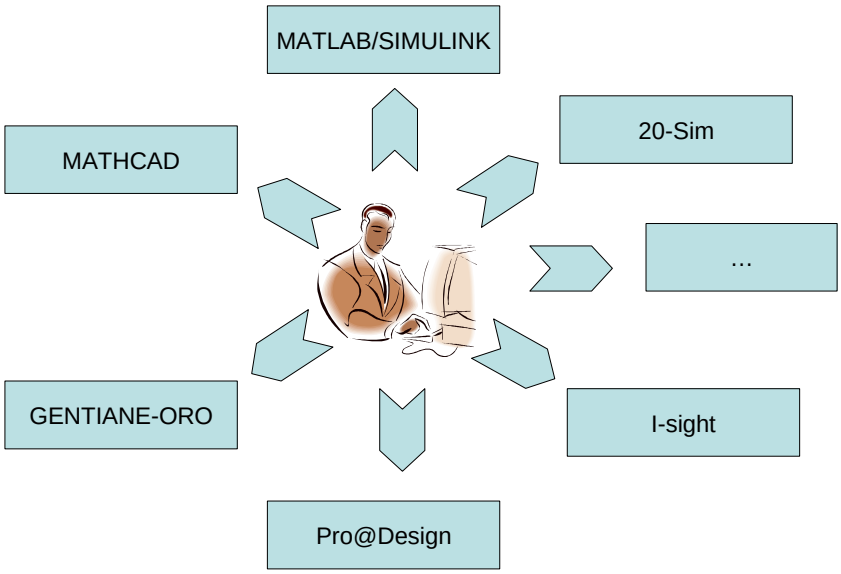


Figure 1. 15: Nécessité de décrire n fois une structure unique en fonction de l'application utilisée

E - La capitalisation des modèles

E - 1 - 1 Enjeu de la capitalisation

Un enjeu des outils d'aide à la conception est, aujourd'hui, la capitalisation de la connaissance et de l'expertise de l'ingénieur.

La capitalisation est le moyen de conserver, dans un but de réutilisation et de transmission, la connaissance du concepteur ou de l'entreprise. Elle doit se faire en utilisant plusieurs supports qui garantissent une certaine pérennité à la représentation. Nous déclinons la capitalisation de la connaissance selon deux axes principaux :

- le moyen de conserver, de façon structurée et, idéalement, documentée le savoir d'un concepteur
- le moyen de réutiliser cette connaissance dans des environnements différents selon les objectifs de l'étude.

E - 2 Les moyens de la conservation de la connaissance

Nous distinguons deux possibilités de réaliser cette capitalisation de la connaissance :

- une capitalisation « boite noire » : la connaissance portée par le modèle n'est pas accessible au concepteur.
- une capitalisation « boite blanche » : la connaissance est accessible au concepteur, il peut non seulement la visualiser mais également la modifier.

Un intermédiaire peut être considéré entre ces deux moyens, dans lequel l'utilisateur peut visualiser la connaissance portée par le modèle mais pas la modifier ou bien faire une modification dans un langage de programmation avancé. Nous nommons cette représentation « boite grise »

E - 2 - 1 Les « boites noires »

1 - a) Caractérisation

La capitalisation « boite noire » est une forme de capitalisation dans laquelle le concepteur n'a pas de moyen d'agir directement sur la description du modèle qu'il utilise. Cette impossibilité se décline de deux façons :

- soit il n'a aucun moyen de connaître les lois comportementales traduites dans le modèle

- soit l'accès se révèle tellement lourd (lecture et compréhension d'un code informatique) pour un non initié que la connaissance lui reste effectivement inaccessible et non modifiable.

1 - b) Avantages et inconvénients

Cette forme de capitalisation de la connaissance présente des avantages et des inconvénients. Les avantages les plus évidents sont :

- la disponibilité immédiate du modèle dans l'environnement de calcul grâce aux librairies
- la dissimulation de l'expertise introduite dans le modèle pour une diffusion, par exemple, vers les sous-traitants afin de créer une synergie de conception entre client et fournisseur

Ceux-ci se heurtent aux inconvénients suivants :

- la communicabilité limitée entre des logiciels différents, le modèle boite noire n'est pas portable hors de l'environnement qui l'a créé
- l'inaccessibilité de l'expertise réelle
- l'impossibilité de modifier l'expertise décrite dans le modèle
- l'absence de documentation sur l'utilisation du modèle, notamment des hypothèses utilisées pour le modèle

Il existe aujourd'hui de nombreuses utilisations des boites noires :

- Pro@Design avec les COB (Computational OBject) décrit dans [ATI] et utilisés dans un contexte de dimensionnement de structures
- SABER, ou SIMPLORER et P-SPICE avec le langage VHDL avec des composants prédéfinis dans des bibliothèques

De plus, un intermédiaire existe, les boites grises ;

- ECOSIM qui définit un langage spécifique, SABER avec MAST : dans ces cas, nous considérons que la connaissance est au moins partiellement dissimulé car retranscrite dans un formalisme rigide et obscur pour le néophyte.
- MATLAB/SIMULINK avec les S-Functions pour la simulation temporelle des systèmes

E - 2 - 2 Les boites blanches

2 - a) Caractéristiques

Par opposition aux « boites noires », la capitalisation en « boite blanche » est une voie qui permet de créer des modèles dont la connaissance est accessible à un utilisateur tiers non seulement en lecture mais également en écriture. C'est-à-dire que le concepteur peut ainsi profiter de l'expertise d'un autre, la modifier et l'enrichir pour que le résultat soit un modèle plus complet ou plus adapté à ses besoins.

2 - b) Avantages et inconvénients

Cette conservation de la connaissance du concepteur présente des avantages :

- une certaine garantie pour l'utilisateur qui maîtrise ainsi le modèle utilisé
- la diffusion de la connaissance
- la réutilisation possible dans des contextes différents
- la documentation naturelle des modèles (hypothèses de la modélisation, moyens d'utilisation, …)

et des inconvénients :

- nécessité de comprendre le modèle pour l'utiliser
- l'absence du support de calcul efficace, donc d'outil et des librairies adjointes
- une trop grande lisibilité de l'expertise, qui peut être génante lors de la diffusion de modèles auprès de sous traitants

Néanmoins, cette possibilité de conservation est utilisée dans de nombreux outils :

- DYMOLA avec le langage de Modélisation Orientée Objet : Modelica ®[MOD]
- MathCAD avec des feuilles de calculs
- …

E - 2 - 3 Bilan : boites noires contre boites blanches

Nous présentons sur la Figure 1. 16 un bilan des avantages et inconvénients des deux approches.

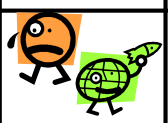
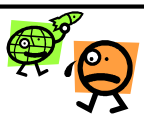
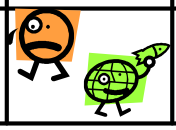
	Exemples	Portabilité	Librairies outils	Documentation (hypothèses, nature des modèles)	Diffusion de connaissance	Modification
Boite Noire	P-Spice					
Boite Blanche	DYMOLA					

Figure 1. 16:Bilan Boites-Noires / Boites Blanches

E - 2 - 4 Les boites « grises »

Il existe un intermédiaires entre les boites blanches et les boites noires. Il s'agit de la description sous forme de code informatique du modèle. Dans la plupart des cas, cette représentation n'est que difficilement compréhensible et modifiable pour un concepteur non averti (modèle, langage de description). Toutefois, dans le cas de la description de loi de commande, par exemple, sa mise au point peut nécessiter des interventions sur cette représentation intermédiaire.

Nous considérons donc le code source informatique comme un intermédiaire de capitalisation acceptable dans ce contexte.

E - 2 - 5 Un besoin : la documentation des modèles

Conserver une connaissance sans lui donner un sens, « une explication de texte », n'apporte pas de réelle plus value, ni pour le concepteur ni pour l'utilisateur. En effet, dans bien des cas la simple écriture d'un jeu de relations entre des variables n'apporte pas d'informations exploitables sur les hypothèses faites, sur la nature des grandeurs mises en relations ou sur la loi physique représentée.

Afin de garantir une certaine intelligibilité au modèle, le concepteur modélisateur se doit de documenter sa représentation mathématique. C'est-à-dire qu'il doit apporter une information supplémentaire au modèle sous la forme de schéma, de nomenclatures des variables ou d'explicitation textuelle des hypothèses faites.

Le soin apporté par le concepteur à la documentation de la représentation mathématique au cours de sa mise au point a un impact direct sur les possibilités de réutilisation et de correction de ce même modèle.

Offrir au concepteur la possibilité de décrire un modèle finement documenté apparaît donc comme une nécessité pour tout environnement de modélisation scientifique. C'est en effet par le biais de cette information, non mathématique ou algorithmique, que l'expertise prend toute sa valeur.

E - 3 Les limites de la capitalisation

Aujourd'hui, la capitalisation de la connaissance des concepteurs se trouve confrontée aux limitations,

- de composition des sous-modèles entre eux, par exemple les problèmes de causalité dans la modélisation ne sont pas encore résolus d'une manière systèmatique
- d'une description orientée de la connaissance, c'est-à-dire que les modèles sont construits pour une utilisation donnée sans considération pour la connaissance qu'ils portent
- liées aux outils supports de leur description

E - 3 - 1 La composition

Certains cas de réutilisation de l'existant peuvent causer des problèmes. C'est par exemple le cas de la composition de modèles de simulation entre eux, qui peut faire apparaître des problèmes de causalité. Ce type de problèmes nécessite souvent pour être résolu de créer un couplage fort entre les deux éléments de la représentation.

La Figure 1. 17 illustre ce propos dans la description des problèmes de causalité qui apparaissent lors de la description d'une association entre une alimentation de type onduleur et une machine asynchrone.

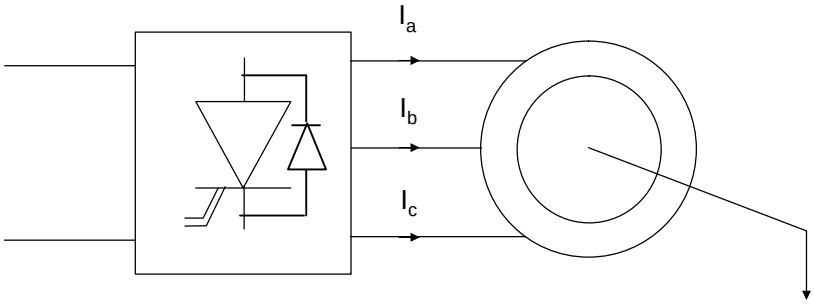


Figure 1. 17: Association Onduleur Machine qui pose des problèmes de causalité

Dans cette association, le fonctionnement est tel que dans la majorité des cas, les courants I_a, I_b et I_c sont imposées par la charge et donc par la machine alors que les tensions sont imposées par l'alimentation. En revanche dans les cas de conduction discontinue, la machine impose les tensions aux bornes des interrupteurs. Dans ce cas de modélisation, il y a donc clairement un problème de causalité [HAU], [ALL], [TAN].

E - 3 - 2 L'orientation

Lors de la description de son dispositif, le concepteur est souvent astreint à un certain nombre de choix qui peuvent le limiter dans l'utilisation ultérieure de ce modèle. Ainsi, dans le cas du dimensionnement de structure, par exemple, le logiciel Pro@Design impose au concepteur le choix des entrées et des sorties de son modèle. L'écriture des équations du modèle impose donc de sélectionner les entrées et les sorties de son modèle. Il en est de même pour les outils de calculs tels que MATH CAD ou Matlab.

Lors de la construction du modèle dynamique de son dispositif, le concepteur est obligé de choisir au début les entrées et les sorties de son modèle mais aussi les grandeurs qui resteront constantes au cours du temps (par exemple imposer qu'une valeur de résistance soit constante). Ces choix a priori sont souvent préjudiciables à une réutilisation des modèles dans des contextes différents.

Par exemple, le modèle dynamique d'une machine synchrone est le même pour les fonctionnements en générateur et en moteur. Pourtant les entrées ne sont pas les mêmes dans les deux cas, cette simple différence impose de créer deux modèles représentant une connaissance identique sous un « formatage » différent. (Figure 1. 18).

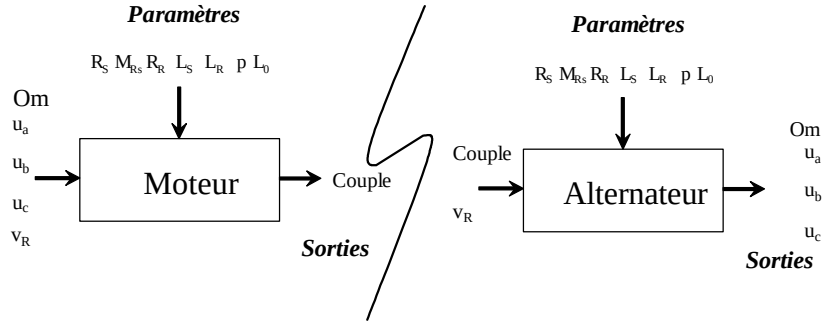


Figure 1. 18: Plusieurs utilisations d'une machine synchrone

F - Solution proposée

Aujourd'hui, le concepteur se trouve confronté à un problème crucial : celui de la gestion de sa ressource modèle. Etant donnée l'utilisation courante des modèles de structures, tant pour

le dimensionnement que pour la simulation des structures, il est primordial pour lui d'avoir une politique de gestion de sa connaissance adaptée à son contexte de travail.
Cette gestion doit se faire sur deux axes :

- la capitalisation de sa connaissance, c'est-à-dire la conservation de ses modèles dans un but de réutilisation
- le travail sur cette ressource, il est nécessaire, pour l'exploiter de fournir au concepteur les moyens de cette exploitation. Ceci se décline sur deux axes, les méthodologies de travail et les outils informatiques.

F - 1 Dimensionnement et simulation

Le dimensionnement automatisé de structures à partir de la description d'un modèle macroscopique est très fréquent dans les bureaux d'étude. Aujourd'hui, l'accroissement des capacités de calculs rend envisageable de réaliser l'optimisation de structures en prenant en compte des contraintes issues de simulations temporelles des structures.
Des travaux précédents [GER-2], [TOU] ont montré la faisabilité d'une telle approche. Dans ce travail, nous nous concentrons sur l'intégration des simulations temporelles complexes dans une optique de dimensionnement.
Cette volonté fait inévitablement cohabiter deux aspects de la modélisation de structure :

- la modélisation pour le calcul
- la modélisation pour la simulation

Il n'existe pas aujourd'hui, à notre connaissance, d'approche unifiée pour ces deux modélisations. L'objectif de notre travail est donc de proposer une approche unifiée de la modélisation et du traitement des modèles, et dans ce cadre nous proposons de nous intéresser à la gestion de la ressource des modèles des concepteurs de dispositif.

F - 2 Gestion de la ressource modèle

La capitalisation de la connaissance du concepteur, et plus particulièrement de ses modèles, ne prend de sens que si elle permet d'envisager plusieurs représentations, plusieurs visualisations. La solution que nous proposons, dans ce contexte, est une aide et une incitation forte à la documentation des modèles.
De plus, gérer la ressource des modèles, cela suppose essentiellement d'éviter un travail redondant. En effet, la multiplication des outils de calculs et de simulation aux finalités différentes impose au concepteur de répéter son travail de modélisation pour chaque outil utilisé.

Les travaux que nous présentons ici s'articulent autour des points suivants :

- une modélisation a-contextuelle, c'est-à-dire la création de modèles qui décrivent la physique d'une structure sans a priori sur l'utilisation de la structure ou du modèle
- une structuration des modèles fortement inspirée des concepts de la Modélisation Orientée Objet
- la capitalisation de la connaissance des experts sous deux formats contextualisés et hors contexte
- un système de génération de code réalisant des passerelles vers différents environnements de calculs scientifique.

F - 3 Différents niveaux de représentation des modèles

Pour répondre à la multiplication des outils, nous proposons une approche de la modélisation qui fait intervenir plusieurs niveaux de représentations. Chacun de ces niveaux est adapté à un besoin spécifique du concepteur. Le concepteur travaille indifféremment avec chacun de ces niveaux de représentation, mais il ne décrit son modèle que pour le niveau le plus générique : celui de la description de la connaissance. Nous lui fournissons les outils et les méthodes réalisant le passage du niveau le plus générique aux niveaux les plus contextualisés, c'est-à-dire dédiés à une utilisation spécifique.

Afin de permettre une meilleure compréhension de ce rapport, nous définissons maintenant des termes que nous utiliserons par la suite afin de nous éviter les périphrases inutiles :

- une **boite** est une description formelle d'une structure, il s'agit d'un ensemble de relations (équations, algorithmes) qui représentent la physique de la structure et constitue un modèle orienté
- un **bloc** est la projection de la boite pour un environnement dédié. Il s'agit d'une représentation informatique du modèle : le code ; ce format est souvent capitalisable et réutilisable dans les bibliothèques personnalisées des outils. Le bloc est dédié à un outil, c'est, par exemple, le cas des blocs de SIMULINK.
- un **composant** informatique de calcul est une entité informatique hors environnement qui contient une description du modèle sous forme de code. Il possède les API nécessaires pour l'exploitation du code par d'autres outils, c'est, par exemple, le cas des COB pour Pro@Design.

Chapitre 2

De la modélisation aux calculs

Chapitre 2 De la modélisation aux calculs

A - Introduction

Nous avons vu que, au cours de son activité, le concepteur crée et utilise des modèles. Ce travail sert deux objectifs :

- la simulation temporelle des structures
- le dimensionnement (ou pré-dimensionnement) des structures.

Pour cela, et étant donnés les impératifs de temps, il est nécessaire de disposer de modèles dont les temps de calculs sont faibles. En effet, la démarche de conception n'est pas la même en fonction du temps de calcul nécessité par les modèles utilisés par le concepteur. Dans une optique de dimensionnement, l'importance des temps de calculs est encore augmentée par le grand nombre de calculs effectués. Il est donc nécessaire de disposer de modèles rapides, la plupart du temps macroscopiques.

Lors du développement de tels modèles, le concepteur doit prendre en compte le contexte d'utilisation de son modèle, c'est-à-dire l'environnement informatique dans lequel il sera utilisé. Comme nous l'avons vu dans le premier chapitre, un certain nombre d'environnements de simulation ou de dimensionnement permettent au concepteur d'ajouter des modèles développés pour ses propres besoins. Cette activité n'est malheureusement ni simple ni rapide.

Nous proposons dans ce second chapitre une méthode de travail qui permet au concepteur de développer ses propres modèles dans l'optique de les intégrer :

- soit dans un environnement de simulation (ou de calcul)
- soit dans un composant informatique [DEL] utilisable en conception.

En particulier, nous nous intéressons à la prise en compte de modèles de simulations dans une optique de dimensionnement.

Après une rapide présentation de la méthodologie de travail proposée et de ses enjeux, nous approfondissons les différentes étapes abordées par le concepteur.

B - Présentation générale de la méthodologie

Sur la Figure 2. 1, nous présentons une vision globale de la méthode de travail proposée dans ce chapitre.

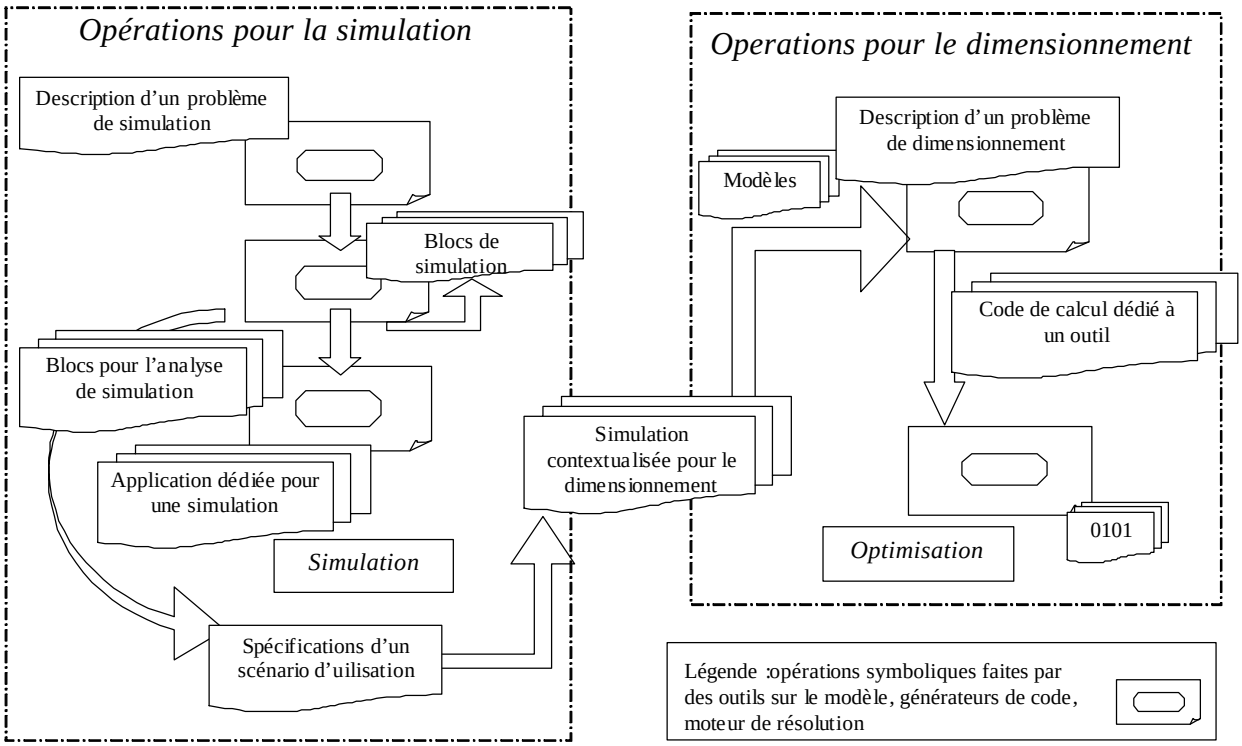


Figure 2. 1: Méthodologie de travail pour la modélisation en conception

Nous rappelons qu'un des objectifs de cette étude est l'intégration de la description de simulations temporelles complexes dans un processus d'optimisation. En effet, des travaux précédents [WUR], [SAU], [COU], [BEL], [DEL], [ATI] ont permis d'appréhender différents éléments de la complexité de la modélisation pour le dimensionnement. D'autre part, il est également possible, dans une certaine mesure, de définir un problème incluant la simulation d'un dispositif continu pour accéder à des grandeurs dimensionnantes du dispositif.

Nous présentons sur la Figure 2. 1 l'ensemble du cheminement suivi par le concepteur pour intégrer des éléments de simulation temporelle complexe dans un composant informatique utilisable pour le dimensionnement. Il s'agit d'une méthode de travail que nous proposons afin de pouvoir automatiser la majorité des actions du concepteur pour simuler et/ou dimensionner son dispositif. Cette méthode fait clairement apparaître deux axes directeurs en fonction de l'objectif visé par le concepteur :

- la simulation d'une structure
- le dimensionnement d'une structure

Ces deux axes sont similaires par l'enchaînement général des actions. Dans la suite de ce chapitre nous décrivons tout d'abord la démarche commune aux deux axes de la description du modèle, nous affinons la description de l'axe de la simulation d'une structure avant de l'étendre au problème de dimensionnement.

Remarque : *Nous présentons ici une séparation des deux objectifs de modélisation. Il est important de noter que l'enchainement « simulation – dimensionnement » n'est pas nécessaire. En effet, il est fréquent de développer un bloc de simulation sans envisager d'utilisation pour le dimensionnement. De même, la construction d'un modèle de dimensionnement ne présuppose pas de l'existence d'un modèle de simulation.*

B - 1 Un fil conducteur commun pour la simulation et le dimensionnement

B - 1 - 1 De la description d'un modèle à l'écriture d'un code

Les modèles sont des moyens utilisés pour le dimensionnement ou la simulation des structures, ils sont donc conçus dans cet objectif. Il est en effet primordial de ne pas perdre de vue les objectifs de calculs et d'évaluation, qui ne peuvent être effectués que dans le contexte d'un outil de calcul.

La Figure 2. 2 présente un enchaînement classique d'actions élémentaires dont la finalité est la production d'un bloc de calcul utilisable dans un environnement de calcul choisi par le concepteur.

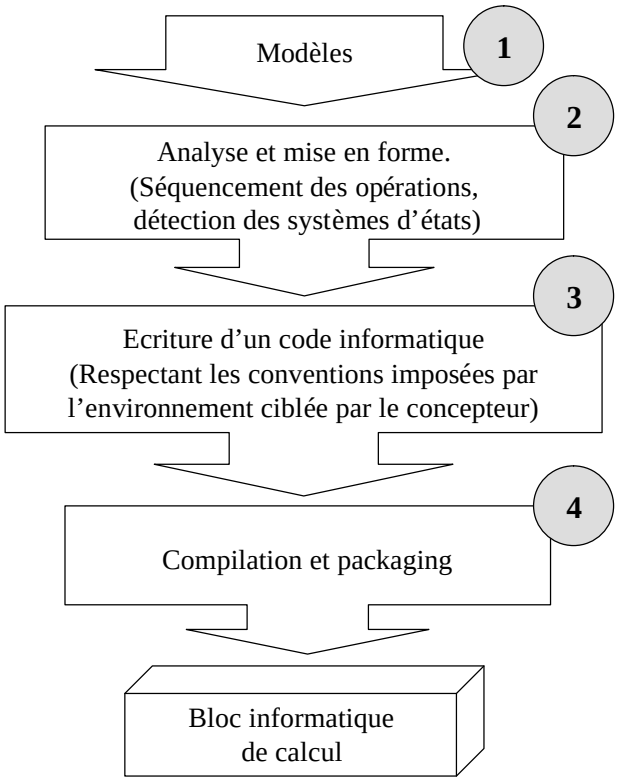


Figure 2. 2: Mise en oeuvre d'un modèle de calcul

Le concepteur définit le modèle utilisé au cours de son étude **(1)**, il s'agit d'un ensemble d'informations, de relations entre des grandeurs qui traduisent le comportement ou les caractéristiques de son dispositif. Puis, il analyse et prépare **(2)** son modèle afin d'obtenir une représentation de cette information qui lui offre le moyen de définir rapidement le code de calcul de son modèle **(3)**. Pour finir, il utilise un compilateur de code et un éditeur de lien **(4)** afin d'obtenir un bloc de calcul ; c'est-à-dire, une description de son modèle dédiée à un environnement. Celle-ci est utilisable par le biais des bibliothèques de l'outil qu'elle vient compléter.

B - 1 - 2 La description d'un modèle (1)

La première étape du travail du concepteur est la mise au point d'un modèle macroscopique de son dispositif. Il écrit un jeu d'équations et de relations algorithmiques qui représentent le comportement dynamique ou statique de son dispositif.
La description de ces équations peut se faire par le biais d'un éditeur de texte simple, et la forme élémentaire de capitalisation de la connaissance du modèle est alors le fichier texte contenant la description du modèle.

B - 1 - 3 Analyse et mise en forme du modèle (2)

Puis, le concepteur prépare l'écriture du code informatique correspondant à ces besoins. Ceux-ci sont caractérisés par :

- un « contexte » d'utilisation : c'est-à-dire le choix d'un ensemble de grandeurs qu'il connaît (des caractéristiques physiques accessibles par la mesure ou l'identification) pour permettre l'évaluation des inconnues.
- un environnement logiciel : l'outil informatique supportant les calculs nécessaires.

La définition du « contexte » d'utilisation de son modèle impose une « orientation » du modèle. La résolution numérique du problème se traduit alors par la résolution d'un problème tel que :

$$f(S,E)=0$$

où S, respectivement E, est le vecteur des grandeurs de sorties, respectivement d'entrées, du modèle.

Les contraintes de temps, issues du contexte d'utilisation des blocs de calculs, restent aujourd'hui très fortes, malgré l'augmentation des capacités de calculs des ordinateurs.

Rappelons, par exemple, que les temps de calculs pour l'inversion d'une matrice sont en n^3, où n est la dimension de la matrice [NOU]. Il est donc important d'éviter les manipulations portant sur des systèmes de tailles importantes. Ainsi, afin de produire un code de calcul rapide, le concepteur doit déterminer, si possible, une séquence de calculs élémentaires conduisant à la résolution de son problème numérique. Pour réaliser cette opération, il dispose d'outils de calcul formel tels que MAPLE [MAP], MATHEMATICA [MATH-2], etc…. L'usage de ces outils lui facilite la recherche d'une séquence optimale de calcul et l'inversion d'équations élémentaires en lui évitant les erreurs de manipulation symbolique.

B - 1 - 4 Ecriture du code de calcul (3)

Dès lors qu'il dispose de cette séquence d'opérations numériques élémentaires, il doit encore écrire le code informatique correspondant au bloc informatique de calcul utilisé par l'environnement logiciel cible. Cette étape est particulièrement fastidieuse pour le concepteur. Il doit en effet se plonger dans un monde qui n'est pas le sien : celui du développement informatique, pour :

- comprendre les spécifications du code à écrire
- produire le code
- dépenser une énergie considérable dans la mise au point de son bloc de calcul (déboguage, respect des obligations de l'environnement cible…)

Il dispose alors d'un code qui traduit son modèle en une séquence d'opérations compréhensibles par l'outil utilisé. Au passage, il dispose d'une seconde forme de capitalisation de son modèle : le code source du bloc de calcul.

B - 1 - 5 Compilation et packaging (4)

Le code informatique décrivant les opérations de calculs n'est pas une entité utilisable telle quelle dans les environnements de calculs : il est encore nécessaire de le compiler et de faire un édition de lien. Cette opération se fait en utilisant les outils classiques pour cette tache qui sont pour la grande majorité des environnements de développement informatique.

A la fin de ces actions, le concepteur dispose d'un fichier informatique qui contient le bloc de calcul ou de simulation correspondant à son modèle.

Il est par ailleurs fréquent que la mise au point du modèle de calcul requière de nombreuses itérations autour du processus présenté en Figure 2. 2.

B - 2 Les enjeux de l'approche proposée ici

Le travail que nous présentons ici a pour objectif de produire une aide substantielle au concepteur pour la mise au point de ses modèles dans l'optique d'une utilisation dans plusieurs environnements logiciels en suivant les deux axes suivants :

- une méthodologie de travail pour la modélisation en conception
- des utilitaires permettant de libérer le concepteur des tâches fastidieuses auxquelles il est confronté au quotidien.

Lors de ces activités de modélisation, le concepteur se trouve face à un certain nombre d'actions autour desquelles il doit souvent itérer manuellement. Il est donc important, pour lui, d'éviter les pertes de temps induites dans les répétitions de ces activités de modélisation, de mise en forme et de documentation des différents modèles produits lors de ce travail. Ainsi, nous proposons de répondre aux enjeux suivants :

- la capitalisation du travail produit par le concepteur
- la libération du concepteur par rapport aux outils externes (de calcul formel, d'édition de code, …) utilisés dans la démarche suivie
- l'automatisation de certaines tâches productrices d'erreurs : les actions d'analyse du modèle **(2)**, d'écriture du code informatique **(3)** et de compilation **(4)**

C - Méthodologie de travail proposée pour la modélisation en conception

Notre objectif est de structurer la démarche de modélisation afin d'en automatiser certaines étapes. Notons que l'ensemble de cette démarche est actuellement suivie, dans les grandes lignes, de manière implicite par le concepteur qui désire produire un code de calcul. Notre travail consiste donc à extraire et structurer les actions implicites du concepteur au cours de son travail pour produire un support d'aide à la formulation de ses modèles.

C - 1 Traitement pour la simulation de structure

C - 1 - 1 Un constat sur la pratique de la modélisation

Lors de la modélisation du comportement dynamique d'une structure, il est fréquent d'adopter une approche « par blocs ». C'est-à-dire, une approche au cours de laquelle, le concepteur découpe son dispositif en briques de bases qu'il réutilise ensuite dans un modèle global obtenu par description d'un « réseau » de briques élémentaires (Figure 2. 3).

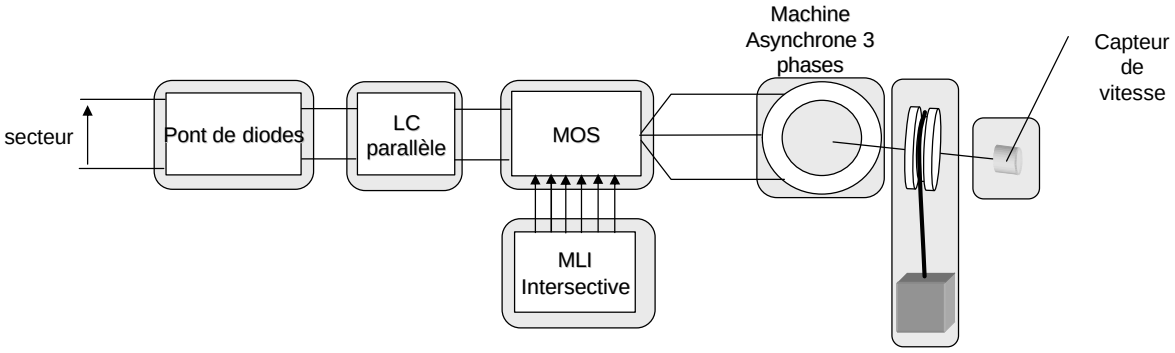


Figure 2. 3: Exemple de découpage en blocs de modélisation élémentaires

Nous pensons que cette approche doit être conservée, et ceci malgré les limitations intrinsèques qu'elle contient et qui sont détaillées en C-1-4. En effet, ces limitations peuvent être aisément contournées par une approche légèrement différente de la modélisation alors que les avantages sont indéniables :

- la focalisation de l'expert sur son métier. Un découpage fonctionnel permet au concepteur de se concentrer sur son expertise (mécanique, électronique, commande, électromagnétisme…) et de profiter de l'expertise des autres
- la capitalisation des expertises dans des bibliothèques métiers
- la capitalisation de l'expertise du concepteur dans des bibliothèques dédiées soit à un individu soit à une entreprise ou un laboratoire.
- la lisibilité de la structure, le découpage fonctionnel permet une meilleure lecture de la structure globale

Une fois ce découpage défini, le concepteur décrit chaque élément de manière indépendante. Cette approche lui permet de s'appuyer sur un travail déjà fourni et dont le résultat est conservé dans une librairie. Une tendance du concepteur modélisateur de structures est de réaliser un découpage de sa structure en blocs fonctionnels. Cette approche peut, dans certains cas faire apparaître des problèmes de causalité en cours de simulation.

Il s'agit d'une problématique déjà largement traitée par de nombreux auteurs [HAU], [ALL]. La modélisation par Bond-Graph, par exemple, apporte une excellente réponse aux problèmes de causalité [ALL], [TAN].

Nous proposons une approche qui permet de traiter, au moins partiellement, cet aspect problématique de la modélisation par blocs (cf § C - 1 - 4) par l'agrégation des modèles.

Il existe une logique de capitalisation de la connaissance qui considère que ces blocs de calculs doivent être l'élément central de la capitalisation. Nous proposons dans le chapitre 3 une capitalisation différente, orientée vers la connaissance du concepteur.

C - 1 - 2 Méthode de travail pour la modélisation en simulation

Sur la Figure 2. 4, nous présentons la méthodologie de travail développée dans ce travail pour permettre la modélisation des structures en régime dynamique.

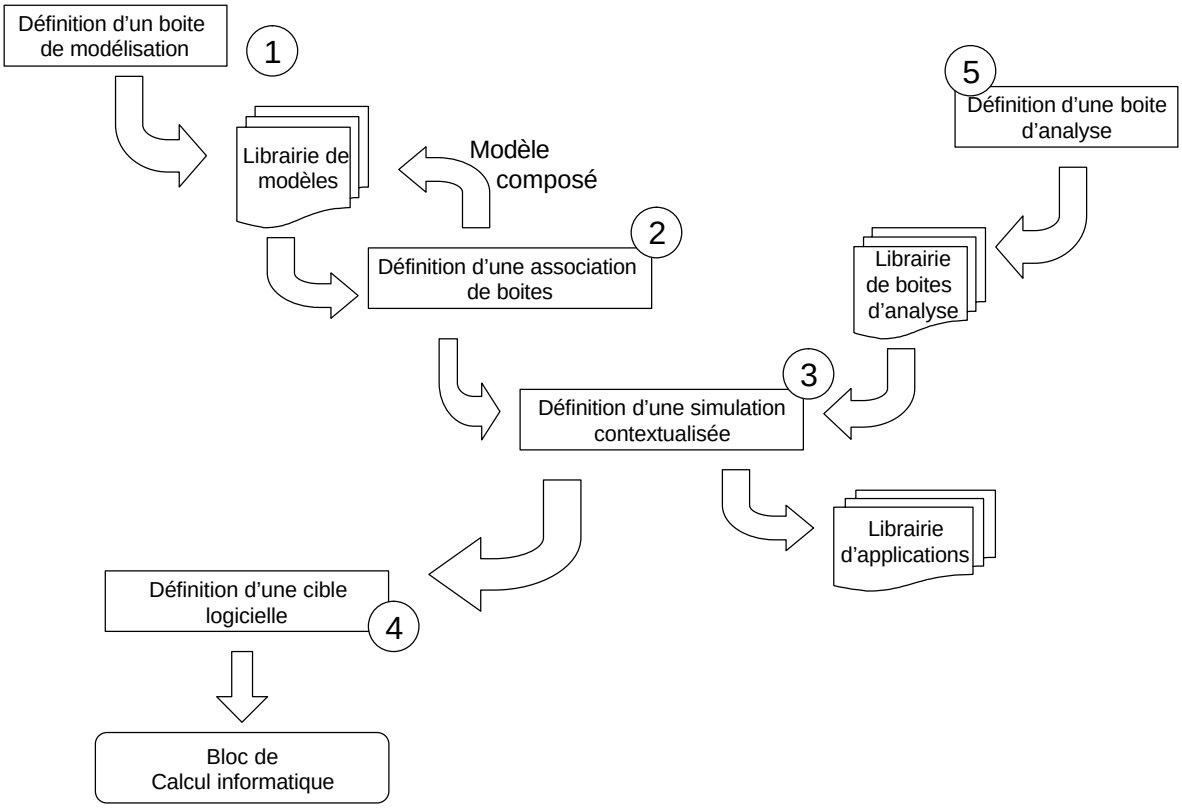


Figure 2. 4: Processus d'obtention d'un composant informatique de calcul

Cette figure présente les principales étapes du processus de création des blocs informatiques de calcul pour la simulation. Dans ce cadre, le travail du concepteur consiste essentiellement en choix :

- de modélisation : par la définition des boites élémentaires de modélisation **(1)**
- d'analyse à effectuer : calcul de valeurs moyennes, de transformées de Fourier **(5)**.
- de structure : lors de la description des associations (choix d'une A.M.C.C. ou d'une autre) **(2)**
- de scénarii d'analyse : récupération d'une valeur maximale, évaluation d'une valeur moyenne **(5)** et **(3)**

- d'environnement logiciel pour le calcul (MATLAB, P-Spice, …) **(4)**

La Figure 2. 4 présente une vue globale des activités du concepteur pour la modélisation d'une structure. Il faut toutefois noter que, en fonction de l'objectif qu'il vise, le concepteur enchaînera les étapes présentées selon deux séquences possibles :

- la description d'un problème de simulation : **(1), (2), (3), (4).**
- la description d'une simulation à intégrer dans un processus de dimensionnement : **(1), (2), (5), (3), (4)**. Dans ce cas, le concepteur définit des analyses à réaliser sur les résultats de la simulation.

Nous nous focalisons maintenant sur chaque étape de ce processus afin de préciser le travail fourni par le concepteur ; notons que certaines de ces tâches peuvent être automatisées. Cette automatisation est proposée dans l'implémentation informatique décrite au chapitre 4 de ce rapport.

C - 1 - 3 Construction de modèles (1)

Comme nous l'avons précisé plus haut, une approche classique en modélisation de structures est une approche par définition de sous-modèles. La première action du concepteur-modélisateur est donc de définir une fragmentation de son dispositif en sous-modèles. Ceci lui permet alors de se concentrer successivement sur chacun des problèmes de modélisation posés par les modèles élémentaires.

Sur la Figure 2. 5, nous détaillons l'enchaînement des actions que le concepteur doit effectuer pour construire un sous-modèle. Nous signalons de même les actions que nous avons pu automatiser.

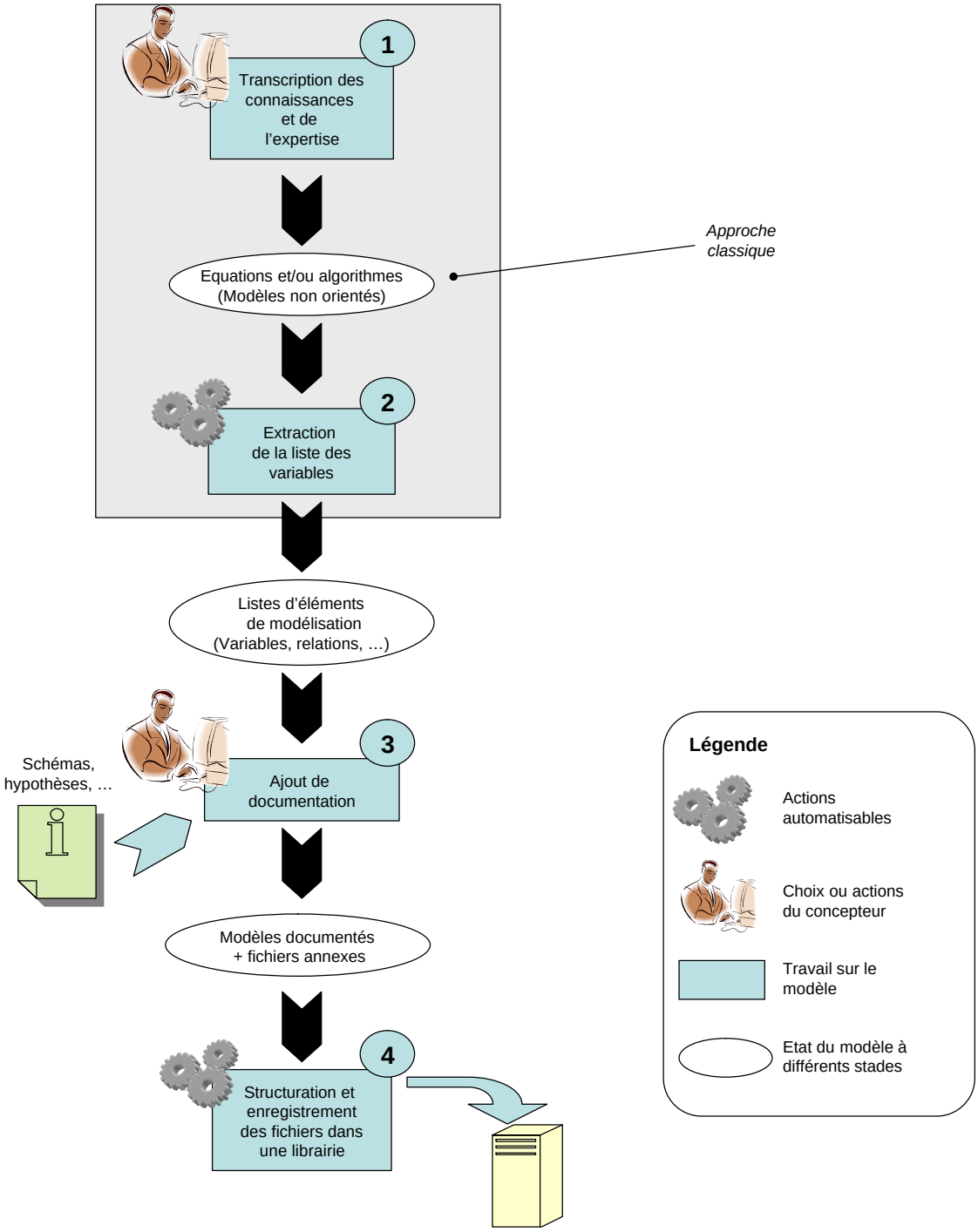


Figure 2. 5: Processus de création d'un sous modèle

Ce processus est celui qui est, aujourd'hui, naturellement réalisé par le concepteur. Nos travaux l'explicite afin de pouvoir en automatiser les étapes systématisables.

Les données d'entrée, fournies par le concepteur, sont composées de la connaissance et de l'expertise de ce dernier **(1)**. Elle est traduite sous la forme de relations mathématiques ou algorithmiques.

Lors de ce processus, le concepteur doit tout d'abord décrire son expertise dans le langage, supposé universel, des mathématiques ou de l'algorithmique **(1)**. Lors de cette description, nous prévoyons que le modèle de description doit pouvoir être réutilisé, ce qui veut dire compris par d'autres personnes. Dans ce contexte, le concepteur doit attacher une grande importance à la documentation de son modèle **(3)**, pour cela nous lui proposons une infrastructure logicielle facilitant cette activité.
Pour chaque élément de modélisation (équations ou algorithmes), nous extrayons automatiquement la liste des variables intervenants dans la description **(2)**. La liste de ces variables est proposée au concepteur qui les documente **(3)** en leur attachant un ensemble d'informations qui enrichissent la sémantique des variables. Cet enrichissement se fait par l'ajout de schémas, de graphes, de commentaires. Le concepteur peut également spécifier des unités pour chaque variable, ce qui permet d'automatiser le contrôle de l'homogénéité des équations.
La gestion des fichiers et de la structuration des données représentant le modèle est gérée de manière automatique, puis le résultat de ce travail **(4)** est ajouté dans une librairie de modèles. Ces fichiers sont une représentation structurée et organisée de la connaissance de l'expert. Idéalement, cette connaissance est enrichie de documentation (schéma, hypothèses de modélisation) lors du processus de structuration de l'information.

Ces modèles, qui sont des boites élémentaires de modélisation, sont ensuite réutilisés par le concepteur qui décrit le modèle d'un dispositif par description d'une association de modèles élémentaires.

C - 1 - 4 La description d'une association (2)

La complexité de modélisation d'une structure peut être diminuée lorsque l'on considère cette structure comme une association de sous-systèmes élémentaires.
Mais, lors d'une telle approche de la modélisation, le concepteur est confronté à un problème récurrent de la modélisation par sous-modèles : le problème de causalité en un même modèle.
Dans un cas tel que celui-ci, nous proposons au concepteur de construire une représentation unique regroupant les sous-modèles incriminés dans le problème de causalité.
Sur la Figure 2. 6, nous présentons le découpage en sous-modèles imposé par la résolution du problème de causalité.

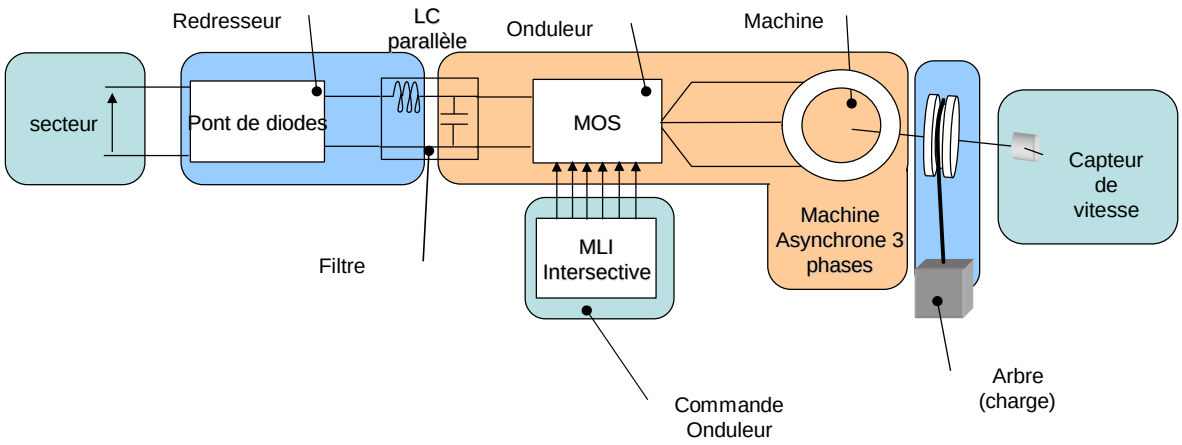


Figure 2. 6: Exemple de découpage en blocs de modélisation élémentaires

Ainsi, afin de résoudre le problème de causalité qui apparaît entre la machine et l'onduleur, nous construisons un modèle unique regroupant ces deux éléments.

Ainsi, dans le cas de l'environnement MATLAB/SIMULINK, cela correspond à réécrire une S-Function qui globalise les modèles des blocs incriminés. Lorsque l'on adopte cette solution, il apparaît alors d'autres problèmes :

- celui des noms de variables, nous renommons l'ensemble des variables de ses blocs élémentaires pour maintenir une cohérence interne dans le modèle résultant.
- la globalisation des modèles, c'est-à-dire qu'il est nécessaire de reconstruire les jeux de relations en prenant en compte l'ensemble des blocs de modélisation.

L'association décrite par le concepteur est elle-même considérée comme un « bloc de modélisation » et, à ce titre, bénéficie des avantages de capitalisation et de réutilisation présentés plus haut.

C - 1 - 5 Définition d'une simulation contextualisée

La description d'un modèle pour la simulation est dite contextualisée lorsque le concepteur a choisi :

- un modèle de structure
- des paramètres physiques, qui sont les caractéristiques statiques de la structure
- les entrées du dispositif, également appelées « excitations » dans la dénomination des systèmes d'états faite ici
- les sorties qui sont les grandeurs internes du système auxquelles il désire avoir accès

Lorsque le concepteur a fait ses choix, nous préparons un code de calcul pour l'environnement cible. La Figure 2. 7 présente les différentes étapes d'une telle préparation du code d'un calcul pour un environnement.

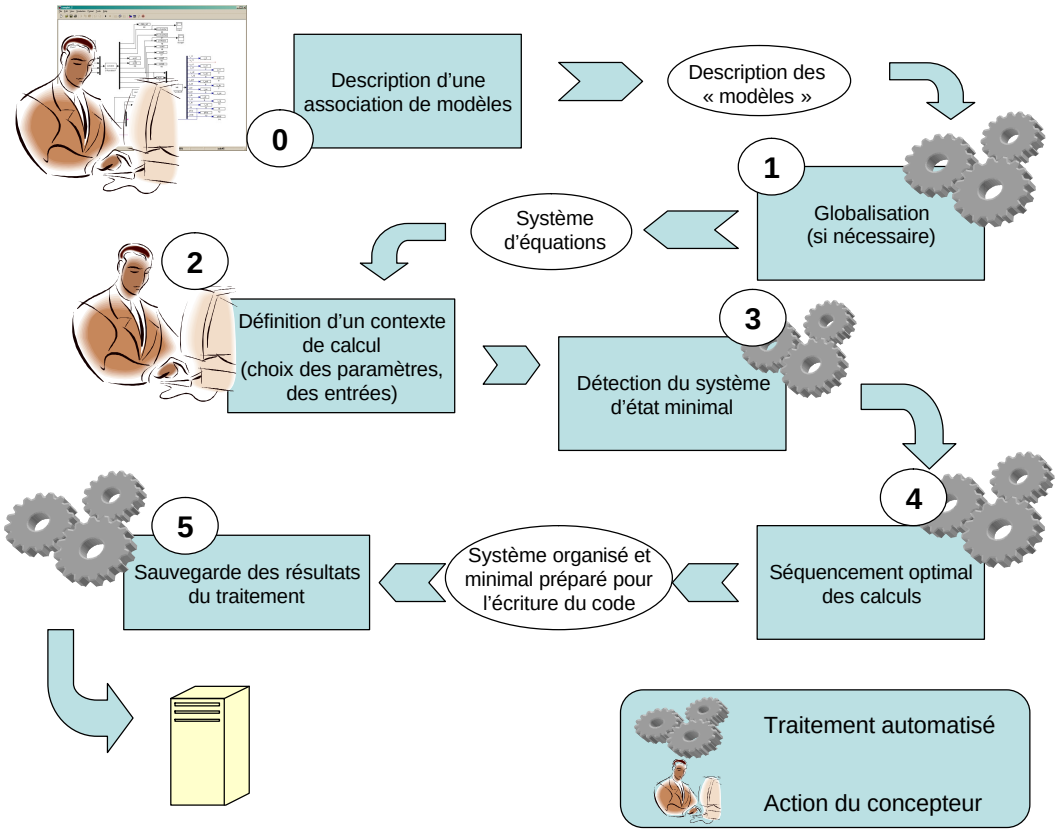


Figure 2. 7: Traitement sur le modèle préparant l'écriture du code

Nous présentons ici les objectifs des différentes étapes de traitement du modèle.

L'étape initiale est la description d'une association de blocs **(0)**. Elle correspond à la description de la structure dont on désire faire la simulation.

Dans ce chapitre, nous détaillons l'enchaînement des traitements à faire sur les modèles pour préparer la création du code informatique, nous ne regardons que les aspects méthodologique, le chapitre 3 de ce rapport détaille les traitements.

5 - a) La globalisation (1)

La globalisation est la réponse proposée aux problèmes de causalité évoqués plus haut. Il s'agit de créer un bloc unique par mélange des descriptions des deux problèmes. Le point bloquant est la gestion de la cohérence des variables et en particulier les problèmes de renommage des variables. A ceci s'ajoute une augmentation conséquente de la difficulté associée à l'augmentation de la taille des modèles. Il s'agit d'un travail fastidieux qui est très

largement source d'erreur, mais entièrement automatisable par des techniques symboliques (analyse des fichiers et de remplacement de chaînes de caractères).

5 - b) La définition d'un « contexte d'utilisation » (2)

La définition du contexte d'utilisation des modèles est l'opération qui consiste à déterminer, dans le jeu des grandeurs décrivant le dispositif, des paramètres, des grandeurs d'excitation (c'est-à-dire les entrées au sens des systèmes d'états) et des grandeurs de sorties.
Il s'agit en fait de l'opération qui permet de construire un modèle adapté à une utilisation en partant d'une description générique. Nous développons ces aspects dans le chapitre 3 de ce rapport.

5 - c) La détection des systèmes d'états (3)

La grande majorité des environnements de simulation temporelle utilise des techniques d'intégration numériques qui travaillent sur la représentation d'état des systèmes. L'intégration des systèmes différentiels se traduit par la résolution de l'équation suivante :

$$f(\dot{X}, X, Y, U, t, P) = 0$$

où :

- t est le temps courant
- X est le vecteur des grandeurs d'états
- U est le vecteur des sources (les excitations du système)
- Y est le vecteur des sorties
- P est l'ensemble des paramètres du sytème

Dans le cas des systèmes linéaires, le problème précédent s'écrit :

$$\begin{cases} \dot{X}(t) = A(P) \cdot X(t) + B(P) \cdot U(t) \\ Y(t) = C(P) \cdot X(t) + D(P) \cdot U(t) \end{cases}$$

où :

- les notations précédentes sont reprises
- A, B, C, D sont des matrices dont les coefficients sont constants en fonction du temps dans le cas d'un système linéaire.

Dans ce dernier cas, les manipulations sont faites sur des matrices dont la dimension est donnée par la taille du vecteur d'état, il est donc important de déterminer la taille minimale de ce vecteur.

Dans le domaine du Génie Electrique en particulier, il est possible de déterminer la taille minimale du vecteur des grandeurs d'états de manière symbolique. En effet, de nombreuses relations algébriques apparaissent entre les grandeurs d'état et il est possible de les utiliser pour réduire considérablement la taille du vecteur d'état et donc les temps de simulation.

5 - d) Le séquencement (ordonnancement) des calculs

Toujours dans l'optique de réduire les temps de calculs pour le modèle, il est nécessaire de déterminer une séquence de calculs élémentaires qui permettent la résolution d'un problème du type :

$$f(X, A) = 0$$

Il s'agit de tâches de calcul formel, qui sont productrices d'erreurs lorsqu'elles sont réalisées « à la main », mais dont l'automatisation est possible dans une large mesure. Ceci est discuté dans le chapitre 3.

C - 1 - 6 Création d'un bloc informatique pour la simulation

L'étape finale du processus est la création du bloc de calcul utilisable dans l'environnement de calcul scientifique choisi par le concepteur. Cette dernière étape est entièrement automatisée en utilisant des outils spécifiques. Ceux-ci s'appuient sur une représentation générique de ce que doit être le code à générer (cf. chapitre 3).

C - 1 - 7 Construction d'un bloc d'analyse (5)

Puisque nous décrivons les simulations de structures pour les intégrer dans un processus d'optimisation, il est nécessaire de prévoir une analyse des résultats de la simulation pour extraire les informations exploitables. Ces analyses peuvent être des évaluations de valeurs moyennes, des transformées de Fourier, des recherches de maximum ou toute autre analyse utile pour la description d'un problème de dimensionnement. La Figure 2. 8 illustre deux solutions possibles pour réaliser ces analyses.

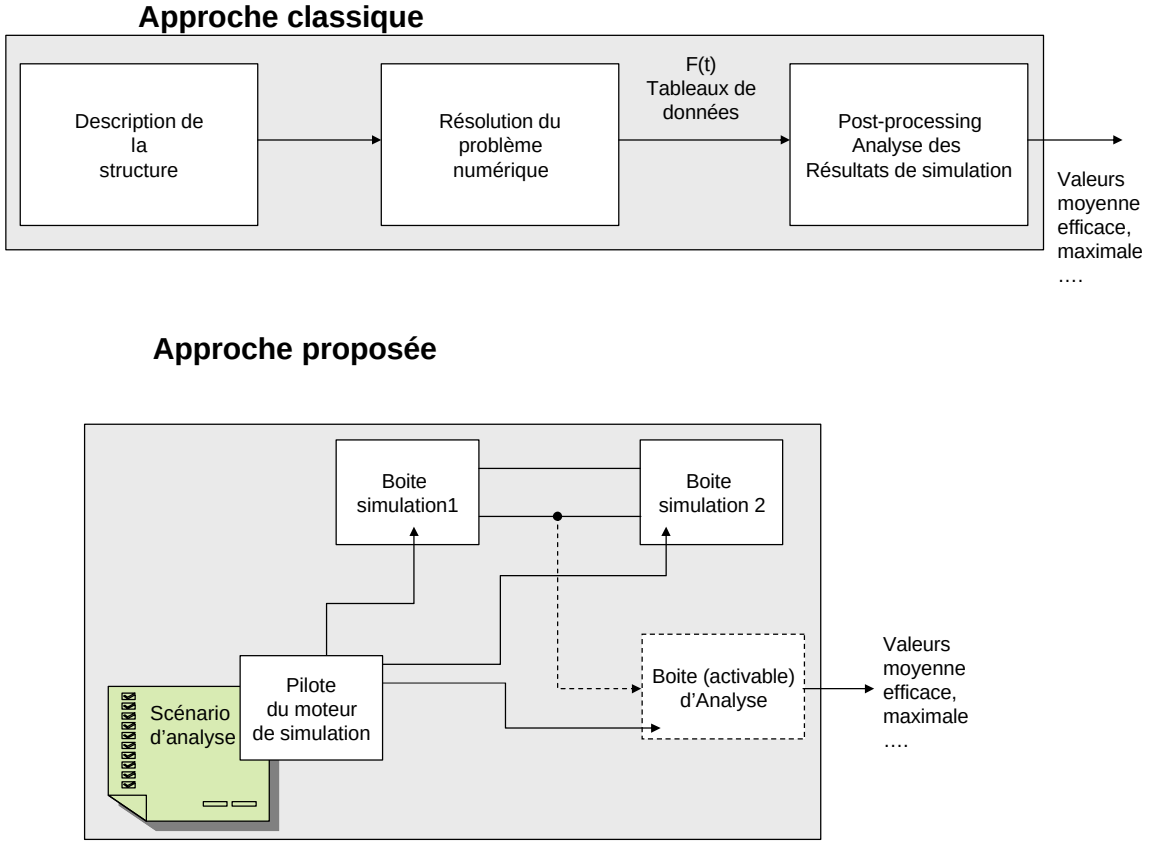


Figure 2. 8: Solution pour réaliser des analyses de simulation

Cet ensemble d'analyse peut être réalisé par deux approches :

- une classique qui repose sur un post-processeur : c'est l'utilisation la plus courante. Elle consiste à effectuer une simulation temporelle en utilisant un outil de calcul, récupérer un ensemble de tableaux de valeurs qui contiennent les courbes temporelles de chacune des variables à analyser. Puis, l'analyse est ensuite exécutée sur le tableau lui-même, par exemple dans un environnement de post-processing.
- celle que nous proposons : le concepteur utilise des analyses dynamiques, c'est-à-dire que les analyses sont réalisées en cours de simulation. Pour cela, nous proposons au concepteur de construire des boites d'analyse et des scénarii pour la simulation. Les boites d'analyse sont des modèles, au même titre que les autres sous-modèles. Les scénarii d'analyse sont des journaux d'actions utilisés pour piloter le moteur de simulation.

La solution que nous proposons permet de gagner un temps considérable durant les évaluations successives du modèle. En effet, analyser les données au moment de leur création autorise une plus grande souplesse dans les études faites et permet en particulier de limiter des

échanges de données consommateurs de ressources de calcul. Le concepteur décrit donc des blocs d'analyse, de la même façon qu'il décrit des blocs de simulation.

Les boites d'analyse peuvent être des actions simples, telles que la recherche d'un maximum ou l'évaluation d'un temps de réponse. Mais, il peut également être question d'activités plus complexes telles qu'un calcul de valeurs moyennes, auquel cas, il est nécessaire de décrire un « scénario » de simulation :

- atteindre le régime permanent, ce qui implique de le détecter [GER-2], [TOU]
- simuler durant une période en réalisant le calcul de valeur moyenne
- arrêter la simulation

En substance, la création d'un bloc d'analyse est similaire à la création d'un bloc de modélisation, le résultat est lui aussi capitalisé sous une forme directement réexploitable dans différents contextes par le concepteur.

C - 1 - 8 Définition d'une simulation contextualisée(4)

Nous disons qu'une simulation est contextualisée pour le dimensionnement, dès lors que le concepteur lui a adjoint un certain nombre de blocs d'analyse qui réalisent les opérations d'analyse des résultats de simulation réexploitables dans le cadre d'une optimisation. Par exemple, la simulation d'une machine asynchrone alimentée par un onduleur est contextualisée lorsque l'on adjoint à la description de l'association un bloc d'analyse permettant d'accéder à la valeur maximale des courants dans les composants de puissance.
Les traitements à exécuter sur les simulations contextualisées pour obtenir un bloc de calcul sont, en substance, identiques à ceux donnés pour la description d'une simulation. Le résultat est identique : une description d'actions informatiques à réaliser pour permettre la résolution numérique et la création d'un bloc de calcul.

C - 1 - 9 Conclusions sur la simulation

Nous avons proposé dans cette partie une méthode de travail dont l'objectif est la simulation d'une structure dont le modèle est basé sur des équations, des fonctions mathématiques ou des algorithmes.
Nous avons proposé d'automatiser certaines de ces étapes afin de libérer le concepteur des tâches fastidieuses et sources d'erreurs.

Notons enfin que l'objectif final de dimensionnement impose de créer un bloc informatique de calcul dédié à cette activité. La méthode de travail que nous proposons permet non seulement de valider les modèles de simulations, de vérifier leur comportement dans des environnements divers mais également de générer des blocs de simulation utilisables dans un contexte de dimensionnement.

C - 2 Traitement pour le dimensionnement de structure

Pour la construction d'un bloc de calcul, il est également possible de réaliser automatiquement un certain nombre de traitements formels sur le modèle du concepteur afin de pouvoir écrire un code informatique. La Figure 2. 9 présente en particulier une démarche adoptable pour le dimensionnement de structures.

Nous présentons dans cette partie en quoi la reprise des concepts de modélisation par « blocs » permet d'apporter une solution intéressante aux problèmes classiques de gestion des gammes, ou des points de fonctionnement multiples.

C - 2 - 1 Méthode de travail pour la modélisation en dimensionnement

La Figure 2. 9 présente une vue globale de l'enchaînement des différentes étapes du travail du concepteur qui prépare un modèle pour le dimensionnement de structures.

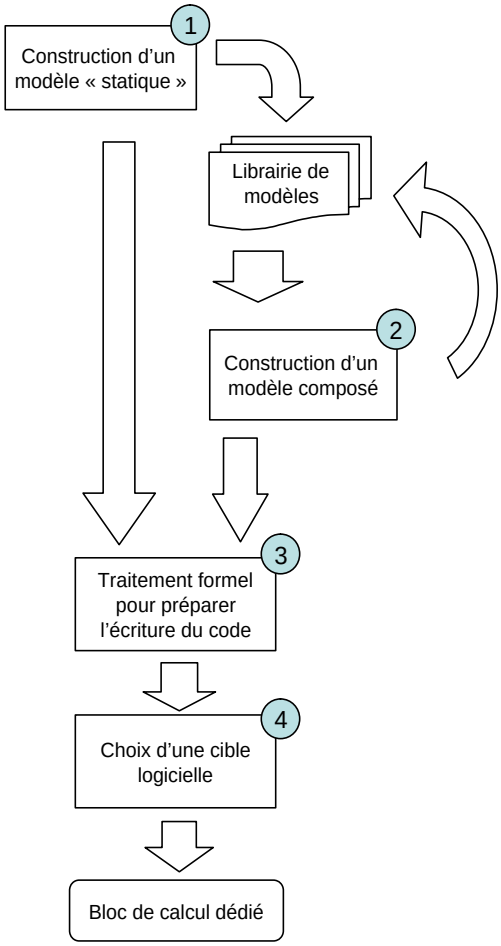


Figure 2. 9: Procédure de création d'un bloc de calcul en dimensionnement

Le travail du concepteur consiste essentiellement en 4 actions:

- la modélisation du dispositif
- la composition de modèles
- la préparation du modèle pour l'écriture du code informatique de calcul
- l'écriture du code lui-même

Etant données les fortes similitudes entre les actions de modélisation pour la simulation et celles pour le dimensionnement, nous ne décrivons pas en détail chacune de ces étapes.

C - 2 - 2 La modélisation des structures

Au cours de cette étape le concepteur modélise et documente son modèle. Un ensemble d'outils lui fournissent une aide dans la description de ce modèle.

Notons simplement les besoins du concepteur au cours de cette tâche :

- il doit être capable de décrire des tableaux qui contiennent des données utilisables pour réaliser une interpolation, par exemple, pour prendre en compte la saturation des matériaux.

- il doit décrire un modèle de la manière la plus naturelle possible, il est donc nécessaire de ne pas l'enfermer dans un formalisme rigide d'écriture de ses relations.
- il peut avoir besoin de décrire des relations complexes entre les grandeurs, dans ce cas, il utilise des éléments algorithmiques.

***Remarque** : les applications de simulation contextualisées constituent également des éléments de modélisation qu'il est nécessaire d'appréhender pour, par exemple, évaluer les pertes dans un convertisseur ou des valeurs moyennes.*

C - 2 - 3 Traitement des modèles pour l'édition de code

Comme pour la simulation, le concepteur utilise son modèle dans un environnement de calcul scientifique, ce qui implique qu'il est nécessaire de créer un code de calcul.
Dans le meilleur des cas, il trouve une expression analytique, ou algorithmique, directe qui permet d'évaluer une grandeur en fonction des autres. Mais, il peut se présenter des cas particuliers pour lesquels il est impossible d'expliciter certaines grandeurs en fonction des autres. La cause principale de cette impossibilité tient à la nature non linéaire du système d'équations, l'inversion formelle est alors impossible. Un cas classique de système implicite en électromagnétisme est la prise en compte de la saturation, et en particulier d'une saturation non linéaire dans les matériaux ; dans un tel cas, l'application conjuguée du théorème d'Ampère et de la conservation du flux magnétique mène à l'apparition d'un système d'équations non linéaires.
L'identification de ces systèmes est souvent une tâche ardue, de plus, dans de nombreux cas, il est possible de diminuer le nombre de grandeurs impliquées dans ce système. Les relations algébriques, telles que les lois des nœuds des réseaux de réluctances entre les différents flux, permettent de réduire considérablement les tailles des systèmes implicites.
Il est remarquable de constater que ces tâches sont automatisables de la même façon que l'on peut construire un vecteur d'état minimal pour la simulation.

Les activités produites par le concepteur pour préparer son modèle à l'écriture du code de calcul sont donc similaires à celles présentées pour la simulation.
Par ailleurs, dans le cadre du dimensionnement de structures, et en particulier lors du dimensionnement par descente de gradient, il est intéressant de connaître les dérivées partielles exactes des différentes grandeurs de sortie en fonction des grandeurs d'entrée. La connaissance des dérivées partielles de manière formelle garantit l'exactitude et élimine les

erreurs introduites par les calculs des dérivées partielles de manière numérique. Cette activité fastidieuse est largement automatisable, et automatisée par le biais d'un outil de dérivation formelle spécifié dans le chapitre 4.

C - 2 - 4 La création d'un bloc ou d'un composant informatique de calcul

Finalement, il est nécessaire de produire un code informatique de calcul. Ce code de calcul est dédié à un environnement préalablement choisi par le concepteur (cf. chapitre 1).
Dans le chapitre 3, nous présentons comment cela est possible automatiquement.

C - 2 - 5 Conclusion sur le dimensionnement

La modélisation pour le dimensionnement suit, comme cela est annoncé en B - 1 , une démarche identique à celle que nous utilisons pour la modélisation de structures en vue de leur simulation. Il est nécessaire de répondre aux mêmes questions :

- de capitalisation de la connaissance du concepteur
- de libération des astreintes d'environnement de calculs formels (sachant que cela peut se résumer à une feuille et un crayon)
- de préparation de modèles pour l'écriture d'un code de calcul
- de multitude des cibles logicielles possibles

S'ajoute à cela, dans notre cas, la nécessité de pouvoir décrire des scénarii autour de simulation de structures. Nous voyons donc qu'il est possible, dans une large mesure, d'apporter une aide substantielle au concepteur.

D - Méthodes et outils pour l'aide à la formulation de modèle

Ce chapitre propose une méthodologie de travail qui permet d'appréhender à moindre frais la complexité de la création d'un code de calcul dédié.
Une méthode est présentée, qui permet de décrire un problème de simulation et d'utiliser cette description dans les contextes de nombreux outils, mais aussi dans le contexte de l'optimisation par la création d'un composant dédié à l'optimisation. Dans ce cadre, la notion de bloc d'analyse a été introduite qui permet de réaliser des analyses sur les simulations temporelles.

Notons, toutefois, que la démarche présentée ici n'est pas réductrice. En effet, il est possible de décrire un problème de dimensionnement sans avoir au préalable décrit un modèle de simulation temporelle. De la même façon, la description d'un problème de simulation temporelle ne conduit pas nécessairement à une description de problème de dimensionnement. La Figure 2. 10 présente deux axes parallèles, et le lien entre les deux n'est pas une nécessité, seulement la réponse à un besoin particulier.

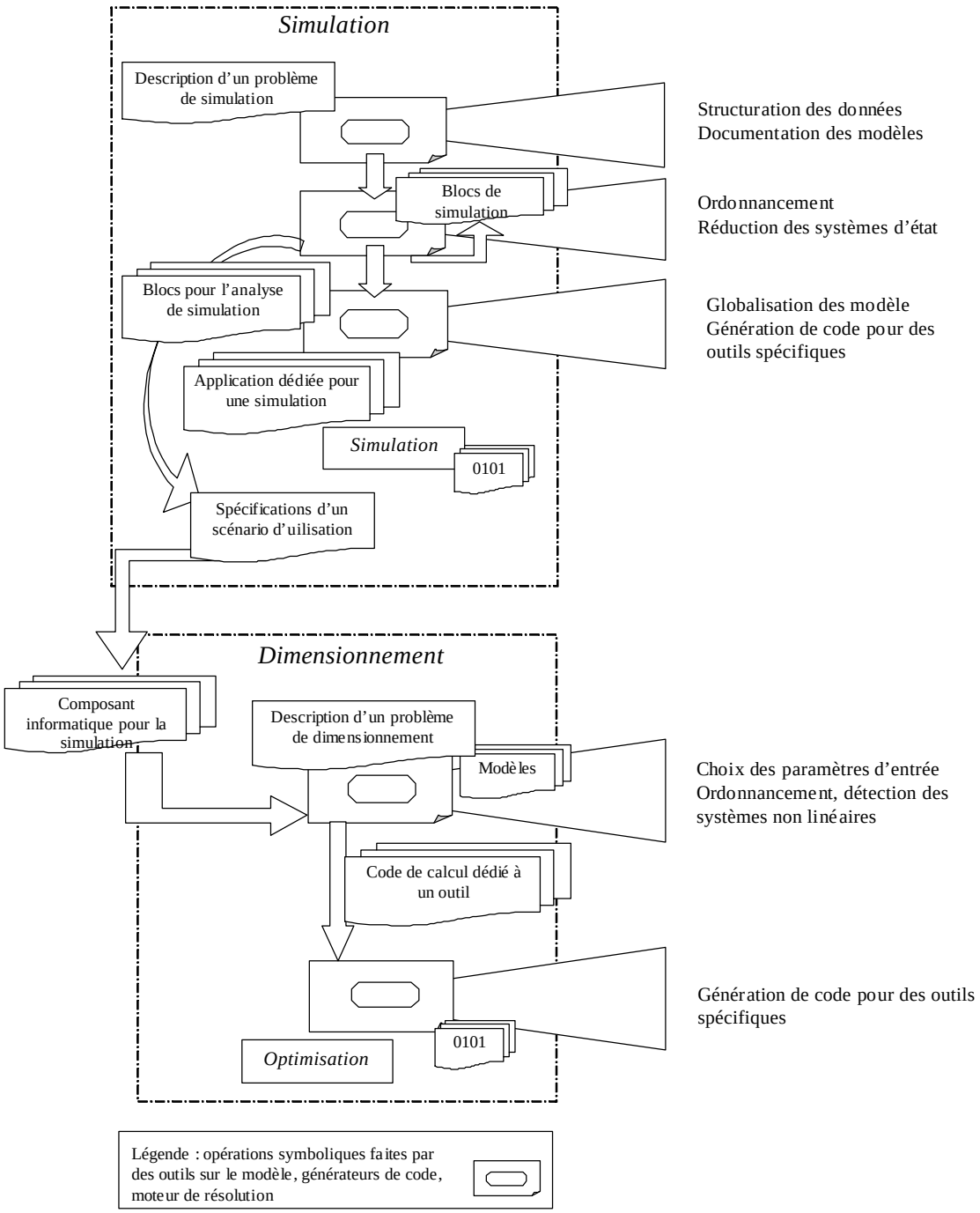


Figure 2. 10: Méthodologie de travail pour la modélisation en conception

Au cours de la description de cette méthode de travail, nous avons proposé d'automatiser des activités qui peuvent être automatisées, déchargeant ainsi le concepteur de tâches fastidieuses et sources d'erreur. Ces activités sont, dans une large mesure automatisée, et le chapitre 4 de ce rapport présente la mise en œuvre informatique de cette automatisation dans le contexte d'un logiciel d'aide à la formulation des modèles.

Il est donc nécessaire de proposer un formalisme unifié de description des modèles pour la simulation et la conception, ce que nous présentons maintenant dans le chapitre 3.

Chapitre 3

Capitalisation et traitement des modèles

Chapitre 3 Capitalisation et traitements des modèles

A - Introduction

Nous présentons dans ce chapitre un moyen de réaliser la capitalisation de l'expertise du concepteur. Dans la première partie de ce chapitre, nous présentons un moyen de réaliser une capitalisation « a-contextuelle » de la connaissance ; puis dans la seconde partie, nous abordons le problème de la « contextualisation » de cette expertise. Nous nous concentrons, en premier lieu, sur la modélisation pour la simulation des structures dont les traitements formels sont plus riches et plus complexes, avant d'étendre notre propos à l'aspect calcul et dimensionnement des structures.

B - Format de capitalisation choisi

Au regard du bilan réalisé au chapitre 1, nous proposons, d'utiliser un support du type boite blanche.

B - 1 La capitalisation a-contextuelle : les boules

Nous appelons contexte d'un modèle, l'ensemble des informations qui décrivent l'utilisation qui est faite de la connaissance qu'il contient. Ainsi, un contexte est défini par :

- une orientation du modèle : c'est le choix délibéré d'un sens de calcul, matérialisé, notamment, par un jeu de paramètres et d'entrées du modèle qui permettent d'en déduire les sorties.
- un environnement de calcul scientifique, qui impose un format de code spécifique.

Un objectif de ces travaux étant de fournir une aide à la formulation des modèles, il est nécessaire de fournir au concepteur les moyens de :

- reprendre, étendre, modifier ou visualiser un modèle
- modifier le contexte d'utilisation de ce modèle.

Dans cette optique, nous proposons de capitaliser la connaissance de l'expert hors de tout contexte d'utilisation. Nous proposons ainsi un formalisme de représentation de la connaissance nommé « boule » qui s'oppose aux boites précédemment étudiées par l'absence de typage des variables par un éventuel statut de paramètres, entrées ou sorties. Ce formalisme reprend les concepts de la Modélisation Orientée Objet décrit dans le chapitre 1.

La Figure 3. 1 illustre la notion de boule sur le cas de la modélisation d'une machine synchrone qui, en fonction de son utilisation, est modélisée de deux façons différentes : moteur ou générateur.

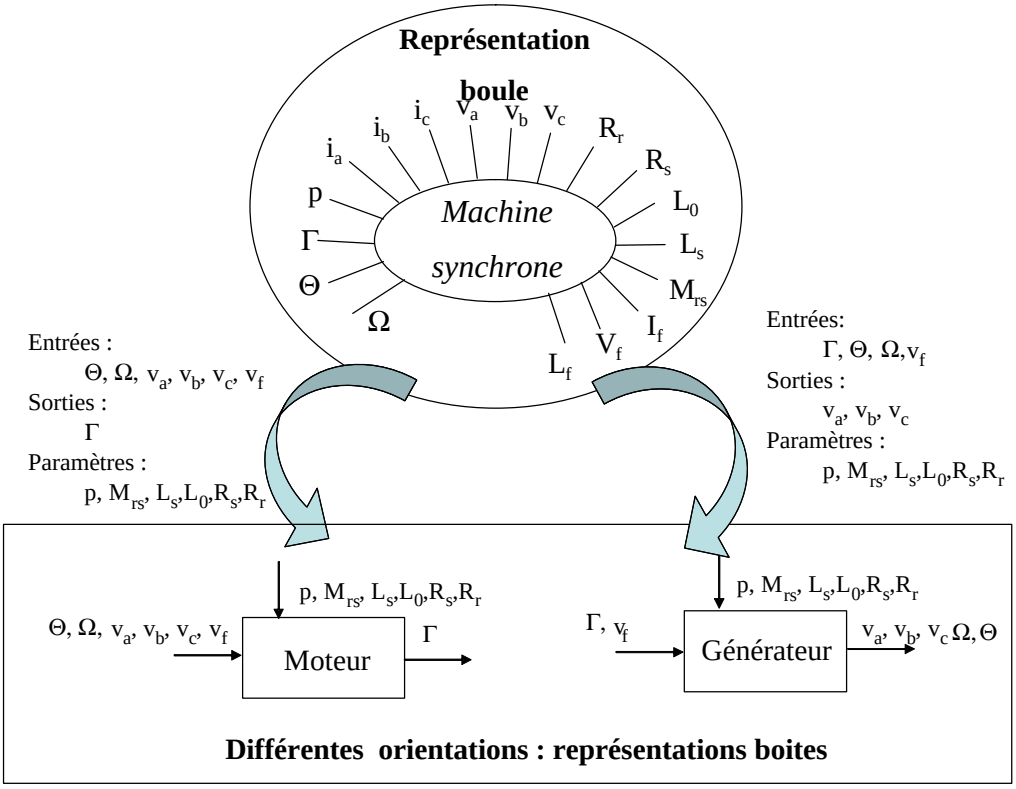


Figure 3. 1: Représentation boule-boites d'une machine synchrone

Cette représentation d'un modèle de dispositif sous la forme d'une « boule » offre la possibilité de disposer d'une représentation hors-contexte de la connaissance. En effet, dans le cas de la machine synchrone, les relations comportementales pour le moteur ou l'alternateur sont identiques. Pourtant, suivant le cas d'utilisation de la machine, les entrées changent, ainsi :

- dans le cas de la simulation d'un alternateur, les entrées du système sont la tension de l'inducteur et le couple d'excitation. Les sorties sont naturellement les trois tensions d'induits, la position angulaire et la vitesse angulaire.
- dans le cas de la simulation d'un moteur, les entrées de la structure sont les position et vitesse angulaire, les tensions d'induit et la tension d'inducteur, la sortie étant alors le couple électromagnétique développé par la machine.

Nous proposons de considérer cette notion de « boule » comme l'élément de base de la capitalisation. Le concepteur décrit un modèle hors d'un contexte d'utilisation. Seules sont définies les lois physiques décrivant le comportement de son dispositif.

Dans la suite de ce chapitre, nous exposons les solutions que nous proposons pour transformer la description « hors-contexte » des modèles en une description orientée et adaptée à la résolution d'un problème particulier. Nous nous appuyons sur les modèles de simulation, dont les traitements sont plus complexes. L'extension aux problèmes de modélisation sans simulation, notamment pour le dimensionnement, est alors naturelle et présente des originalités exploitables pour répondre à des problèmes complexes de dimensionnement (validité d'hypothèse, optimisation multi-structures…).

C - Modélisation des structures

C - 1 Le contexte de la simulation temporelle

C - 1 - 1 Une boite de simulation

Afin de mieux comprendre la suite de ce chapitre, nous donnons ici quelques définitions (voir Figure 3. 2).

Un bloc de simulation est un élément de calcul en simulation qui dispose de :

- paramètres physiques notés $\mathbf{P_P}$, ce sont des grandeurs constantes au cours du temps. Initialement fixés par le concepteur, ils ne changent pas en fonction du temps pour une simulation
- paramètres numériques notés $\mathbf{P_N}$, ce sont des constantes numériques fixées par le concepteur pour paramétrer des algorithmes de calculs numériques. Par exemple, il peut s'agir de paramètres réglant des précisions à atteindre
- des entrées notées **E**, ce sont des grandeurs variables en fonction du temps qui constituent les entrées du bloc de simulation. Par exemple, un signal de tension ou une loi de commande pour un onduleur
- des sorties notées **S**, ce sont des grandeurs dépendantes du temps qui sont sorties de la boite de calcul afin d'être utilisées pour une visualisation ou comme entrée d'une autre boite. Par exemple, la tension aux bornes d'un condensateur ou les états des interrupteurs d'un onduleur.

En interne, la boite de simulation peut contenir un système d'état, nous distinguons donc :

- des grandeurs d'états, notées **X**, qui sont des grandeurs dérivées par rapport au temps, par exemple le courant dans une inductance
- des excitations, notées **U**, qui sont les « entrées » du système d'état, par exemple la tension aux bornes d'une source. Nous les nommons « excitations » afin d'éviter toute confusion avec les entrées du bloc.

- des sorties, notées **Y**, qui sont les grandeurs évaluées par la résolution du système d'état, par exemple la tension aux bornes d'une résistance
- des paramètres notés **P** qui sont des termes constants pour le système d'état

La Figure 3. 2 illustre une partie de ces éléments dans le cas d'un circuit RLC :

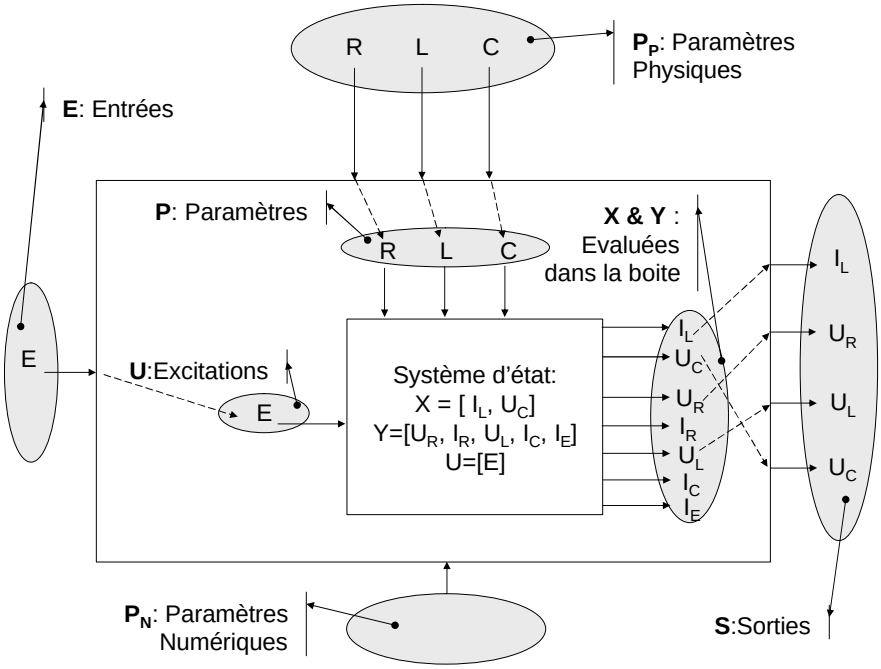


Figure 3. 2: Notation employée pour les boites de simulation

Il est important de noter que les entrées du bloc (**E**) ne sont pas nécessairement les excitations (**U**) du système d'état, de même qu'il n'y a pas correspondance entre les paramètres physiques ($\mathbf{P_P}$) et les paramètres du système d'état (**P**), il n'y a pas non plus de lien nécessaire entre les sorties du système d'état (**Y**) et les sorties de la boite (**S**).

En cours de simulation, les blocs décrivant une association sont résolus séquentiellement en respectant les règles de causalité associées aux entrées et aux sorties.

C - 1 - 2 <u>Processus d'intégration temporelle</u>

Les principes d'intégration numérique utilisés dans les outils de simulation temporelle sont illustrés sur la Figure 3. 3.

Notre objectif n'est pas de créer un moteur de résolution mais de préparer une description des modèles pour une génération automatique de code informatique utilisé par les moteurs existants. Nous devons donc connaître le processus général de résolution numérique de ces outils afin de produire une description qui simplifie la génération de ce code de calcul.

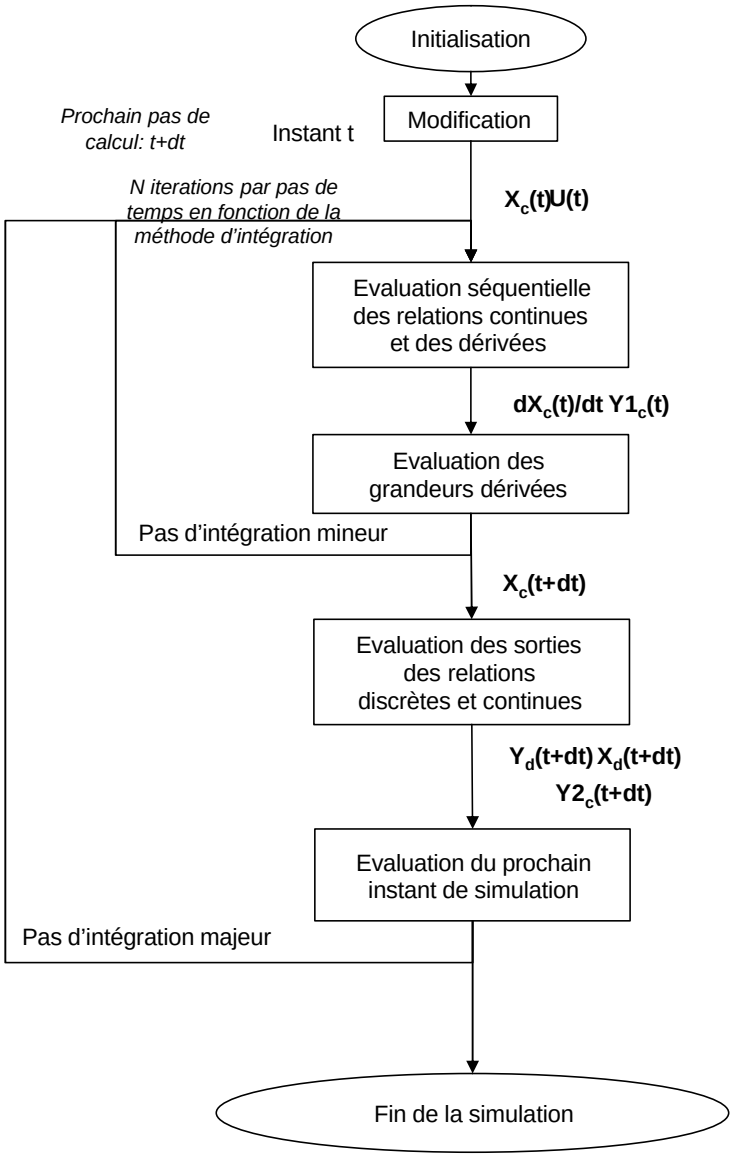


Figure 3. 3: Processus d'intégration numérique

A chaque étape de ce processus, correspond un ensemble d'évaluations spécifiques :

- l'étape nommée modification ne prend en compte que les calculs pour les grandeurs qui ne sont fonctions que des paramètres ou de constantes
- les étapes contenues dans le pas mineur d'intégration ne doivent pas contenir de calculs discrets
- les étapes du pas majeur contiennent la résolution de tous les aspects discrets ainsi que l'évaluation des grandeurs des sorties (**Y**) qui ne sont pas, dans la mesure du possible, résolues dans le pas mineur pour minimiser les temps de simulation

***Remarque** : Certains aspects peuvent changer en fonction de la méthode d'intégration (Runge Kutta, Euler,…) et des outils.*

Il ressort de l'analyse de ce processus de calcul la nécessité d'identifier différents groupes de grandeurs et de relations en fonction de leurs rôles dans le processus de résolution. Il est ainsi nécessaire de séparer les grandeurs discrètes des grandeurs purement continues.

C - 2 Structuration des modèles

C - 2 - 1 Structuration des modèles non orientés

1 - a) Organisation de l'arborescence

Nous structurons les modèles de simulation sous la forme d'arbres dont les nœuds constituent les éléments de la modélisation. La Figure 3. 4 donne une représentation de la structure des modèles que nous proposons.

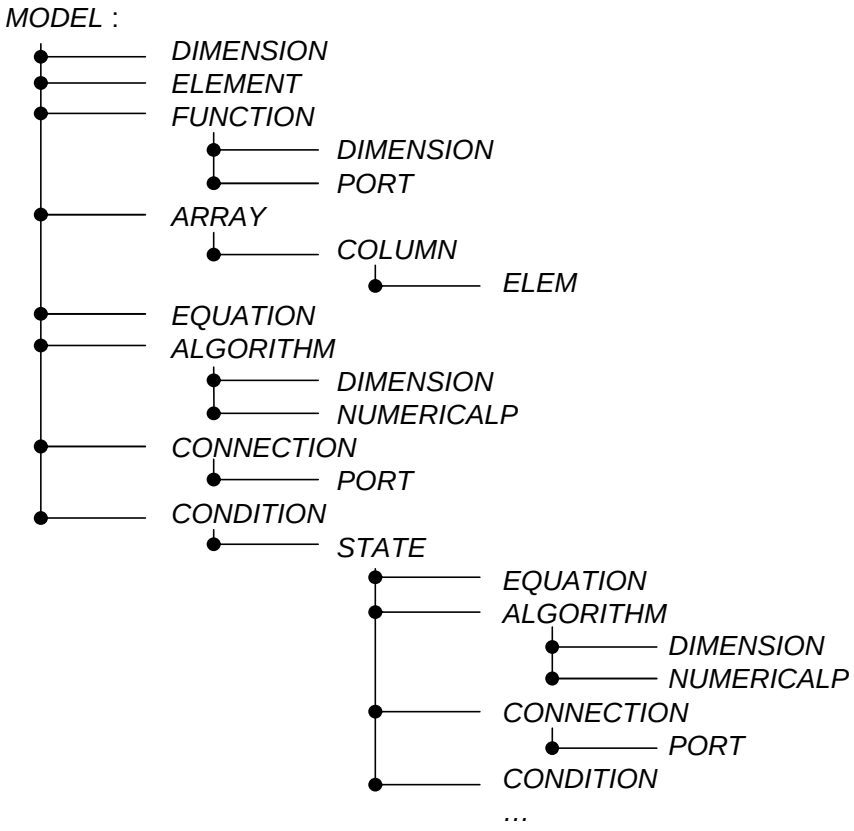


Figure 3. 4: Structure des modèles de simulation

1 - b) Les nœuds de l'arbre

Dans cette structuration, nous identifions trois familles d'entités de modélisation :

- les **éléments constitutifs** : il s'agit de l'ensemble des entités qui permettent de décrire la constitution du modèle.

- les « *DIMENSION* », ce sont les grandeurs physiques intervenant dans la description du modèle du dispositif. Les grandeurs sont, par exemple, les valeurs des tensions aux bornes des composants.
- les « *ELEMENT* », ce sont des entités de description d'une agrégation. Il s'agit des sous-modèles utilisés dans le cadre d'une composition et dont la description est contenue dans une librairie. Par exemple, une association Onduleur-Machine est composée d'un modèle d'onduleur et d'un Machine.

- les **utilitaires de description**, il s'agit de facilités pour la description des modèles :
  - les « *FUNCTION* », ce sont des artifices mathématiques utilisés pour représenter une opération utilisée à plusieurs endroits. Nous définissons, par exemple, la fonction : $f(V_{\max}, w, j) = V_{\max} \times \cos(w \times t + j)$
  - les « *ARRAY* », ce sont des tables de données qui traduisent, par exemple, des abaques de saturations des matériaux
- les **relations**, qui sont les déclarations des relations existantes entre les différents éléments constitutifs du modèle. On y retrouve :
  - les « *EQUATION* », ce sont les entités les plus simples de cette famille. En effet, il s'agit des représentations mathématiques classiques des relations entre grandeurs physiques. Par exemple, la loi d'ohm : $U = R \times I$.
  - les « *CONNECTION* », ce sont les déclarations des relations d'identité ou de somme entre les grandeurs des « ELEMENT » pour relier des sous-modèles entre eux pour construire une agrégation. Ainsi, par exemple, dans le cas de la description d'une association entre un onduleur triphasé et une machine triphasée, cela correspond à établir la correspondance entre des ports (*PORT*) de sous-modèles.
  - les « *ALGORITHM* », qui traduisent des relations plus complexes entre un jeu de grandeurs d'entrées et un jeu de grandeurs de sorties. Dans ce cas particulier, une orientation est donnée à l'algorithme. Par exemple, dans le cas d'un convertisseur statique, nous y décrivons l'algorithme qui permet de déterminer l'état des semi-conducteurs ; en entrée, nous donnons les tensions et courants des différents semi-sonducteurs et en sortie, nous connaissons leur état.

Il existe une entité particulière dans la description d'un modèle hybride, il s'agit des « *CONDITION* ». Ces entités sont utilisées pour permettre la description des systèmes d'état

hybrides (enchaînant différents systèmes d'état continus). Ceci se traduit par la représentation d'un ensemble de « *STATE* » qui traduisent chacun la validation ou non d'une condition. Par exemple, une condition booléenne est composée de deux états selon qu'elle est vérifiée (*true*) ou non (*false*).

***Remarque** : la structure proposée est récursive, c'est-à-dire que*

- *un modèle peut être lui-même composé de sous-modèles connectés entre eux*
- *une condition est définie par des états qui peuvent également contenir des conditions.*

1 - c) Exemple

La Figure 3. 6 présente un exemple d'arbre utilisé pour la description du modèle d'un hacheur série que nous présentons sur la Figure 3. 5.

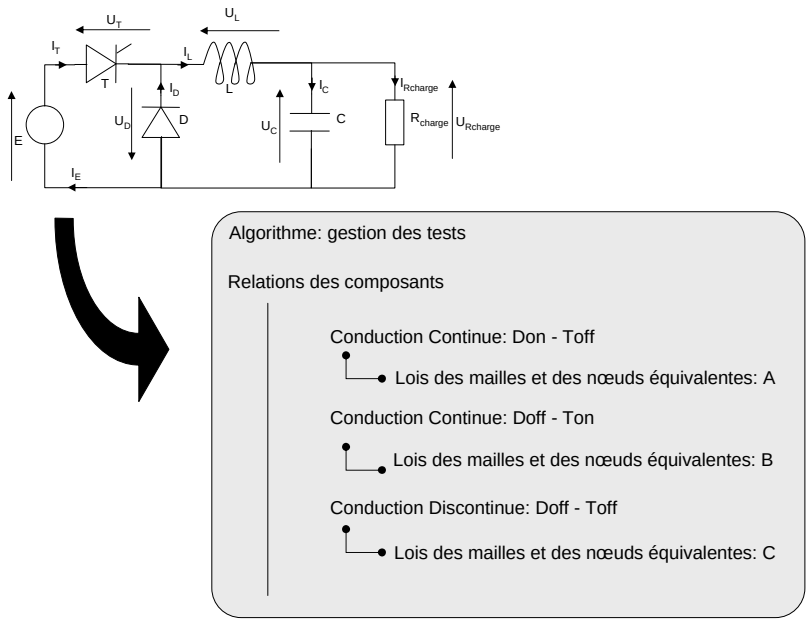
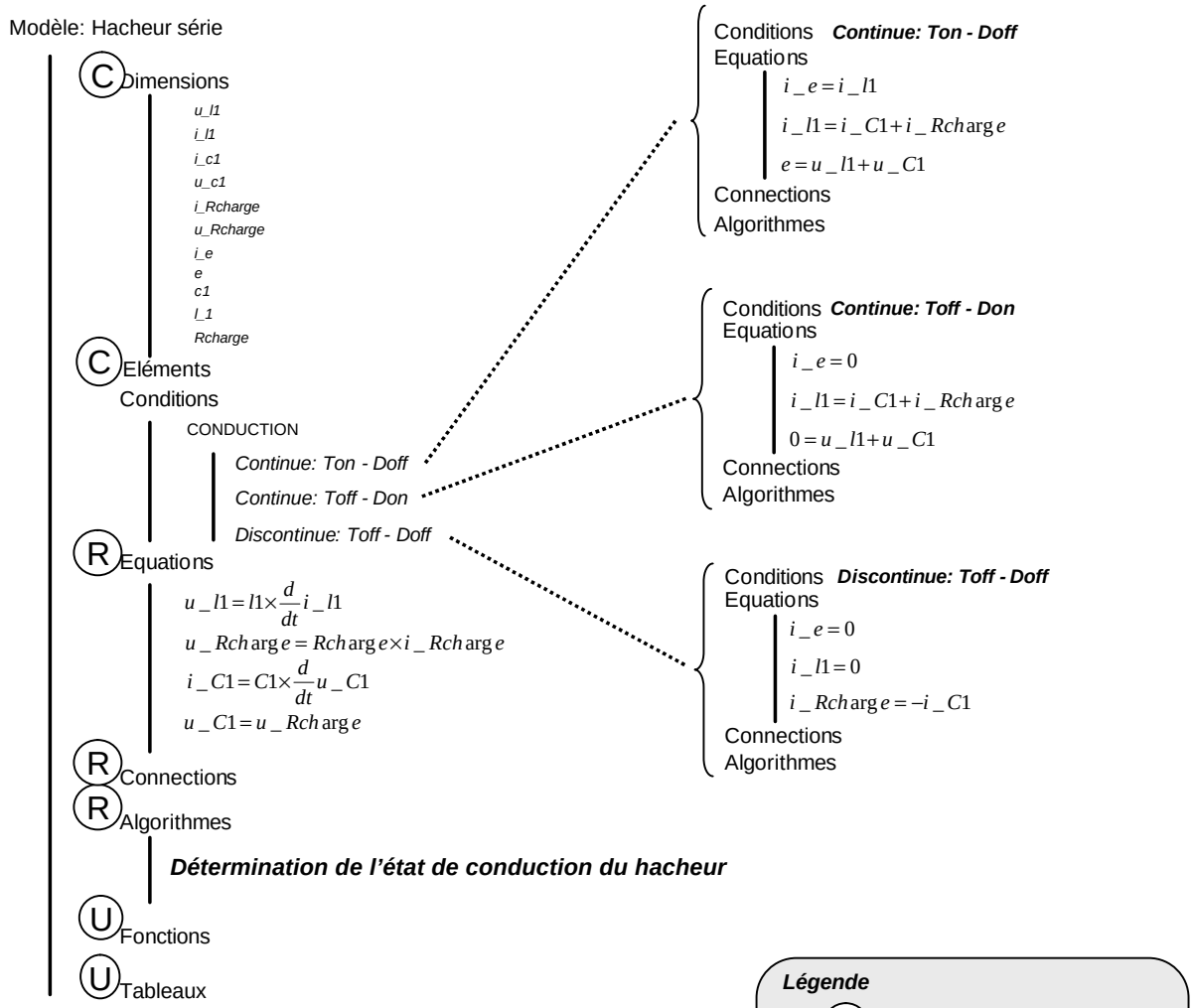


Figure 3. 5:Description d'un hacheur série en topologie variable. Outre les trois circuits équivalents, un algorithme permet de déterminer l'activation d'une configuration ou d'une autre



Formalisme visuel

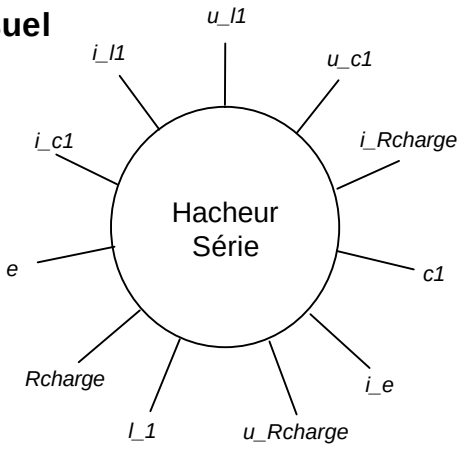


Figure 3. 6: Représentation arborescente du modèle à topologie variable du hacheur série de la Figure 3. 5

Le modèle de ce hacheur est hybride. Nous y trouvons un ensemble de relations qui sont toujours valides, les équations des branches, complétées par les relations des nœuds et des mailles qui sont dépendantes des états des semi-conducteurs de la structure. La validité de ces ensembles de relations est déterminée par la condition « conduction » dont la valeur est déterminée par un algorithme qui évalue l'état de conduction du hacheur.

*C - 2 - 2 **Structuration des boites***

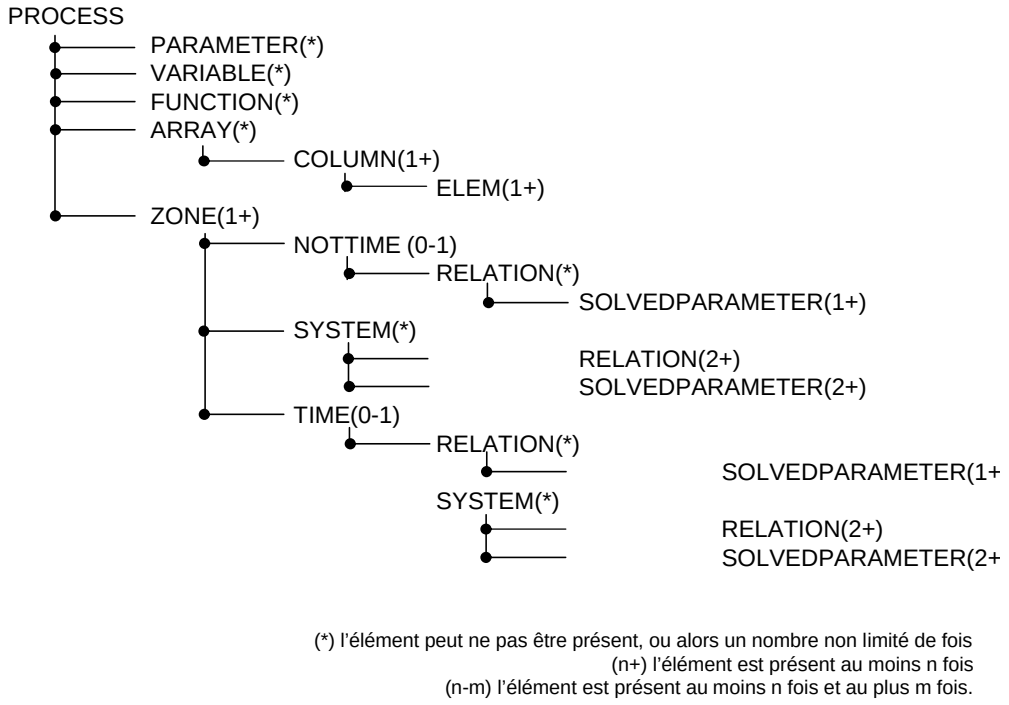
2 - a) Arborescence des modèles orientés

Lorsqu'un modèle est orienté, le modèle boule devient une boite. Cela signifie que le concepteur a choisi de typer les grandeurs en leur attribuant un statut de paramètre (**P_P** ou **P_N**), d'entrées (**E**) ou de sortie (**S**) dans le cadre des calculs numériques. Les modèles orientés sont chargés d'une double sémantique :

- la description d'un processus de calcul, c'est-à-dire d'une liste de séquences d'opérations informatiques de traitements numériques du modèle
- une information sur les liens de dépendances entre les variables du modèle, c'est la matrice d'incidence (cf C - 2 - 3)

Nous avons choisi de structurer les modèles de boites blanches sous la forme d'un arbre.

Nous présentons sur la Figure 3. 7 la structure générale de l'arbre des modèles orientés :



(*) l'élément peut ne pas être présent, ou alors un nombre non limité de fois
(n+) l'élément est présent au moins n fois
(n-m) l'élément est présent au moins n fois et au plus m fois.

Figure 3. 7: Structure générale des boites blanches

Sur cette figure, nous faisons apparaître qu'une boite est caractérisée par deux types d'entités :

- les **grandeurs**, il s'agit de la déclaration des grandeurs intervenant dans la description du modèle avec le statut qui les caractérise :
    - les paramètres $\mathbf{P_P}$ (*PARAMETER*): ce sont les grandeurs qui sont déclarées constantes au cours d'une simulation temporelle
    - les entrées **E** qui sont les grandeurs d'entrées, ce sont des « *VARIABLE* » typées comme *input*. Par exemple, les entrées sont des sources ou des commandes.
    - les sorties **S** qui sont les grandeurs qui sont évaluées par le modèle et que l'on souhaite fournir à d'autres boites. Ce sont des *VARIABLE* marquée de l'attribut *out=true*
    - les grandeurs d'état continues, des *VARIABLE* identifiées comme *state*
- les **zones** de calcul ZONE. Pour chaque combinaison d'état (STATE) de condition et de relations d'une boule, une zone est définie. On y distingue ce qui est traité en pas majeur et en pas mineur dans la résolution temporelle :
    - les aspects continus sont traités dans le pas mineur ; par exemple, les lois comportementales des composants
    - des aspects discrets sont traités dans le pas majeur ; par exemple, les tests de contrôle

Sur la Figure 3. 8, nous présentons la structuration du hacheur précédent sous la forme d'une boite.

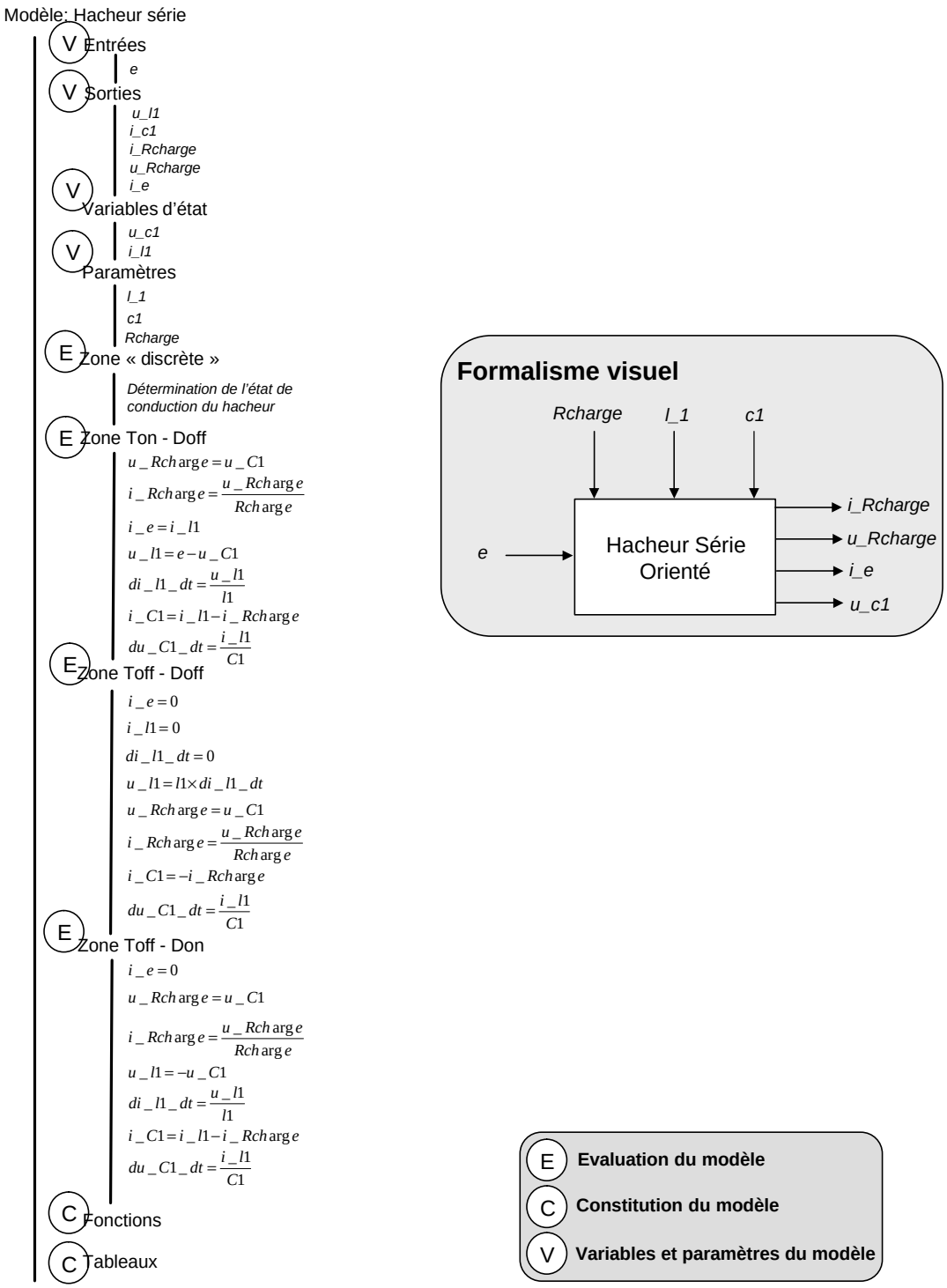


Figure 3. 8: Exemple de description du modèle du hacheur avec la description des opérations numériques à réaliser

Nous proposons une forme de représentation des liens entre les grandeurs qui est utile dans les deux formes de capitalisation (boule et boite) sous la forme d'une matrice : la matrice d'incidence.

C - 2 - 3 Matrice d'incidence

La matrice d'incidence est un outil visuel qui permet d'évaluer la complexité d'un modèle. La Figure 3. 9 présente l'exemple de la matrice d'incidence pour l'état « Ton – Doff » du hacheur série :

	u_Rcharge	**l1**	**i_e**	**di_l1_dt**	**e**	**C1**	**u_l1**	**i_Rcharge**	**i_C1**	**Rcharge**	**i_l1**	**du_C1_dt**	**u_C1**
$u\_Rch\arg e = u\_C1$	**1**	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	**1**
$i\_Rch\arg e = \frac{u\_Rch\arg e}{Rch\arg e}$	**1**	*0*	*0*	*0*	*0*	*0*	*0*	**1**	*0*	**1**	*0*	*0*	*0*
$i\_e = i\_l1$	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	*0*	*0*	**1**	*0*	*0*
$u\_l1 = e - u\_C1$	*0*	*0*	*0*	*0*	**1**	*0*	**1**	*0*	*0*	*0*	*0*	*0*	**1**
$i\_C1 = i\_l1 - i\_Rch\arg e$	*0*	*0*	*0*	*0*	*0*	*0*	*0*	**1**	**1**	*0*	**1**	*0*	*0*
$di\_l1\_dt = \frac{u\_l1}{l1}$	*0*	**1**	*0*	**1**	*0*	*0*	**1**	*0*	*0*	*0*	*0*	*0*	*0*
$du\_C1\_dt = \frac{i\_C1}{C1}$	*0*	*0*	*0*	*0*	*0*	**1**	*0*	*0*	**1**	*0*	*0*	**1**	*0*

Figure 3. 9: Exemple de matrice d'incidence

La matrice d'incidence permet d'évaluer en fonction de son remplissage le couplage des différentes grandeurs du modèle. En effet, plus la matrice apparaît pleine plus cela est révélateur d'un couplage important entre les grandeurs.

Elle permet également de construire l'arbre de dépendance de grandeurs de sortie en fonction des grandeurs d'entrée, ce qui offre une perception condensée de la sensibilité des grandeurs de sortie aux variations des grandeurs d'entrée. Un exemple d'arbre de dépendance est donné sur la Figure 3. 10.

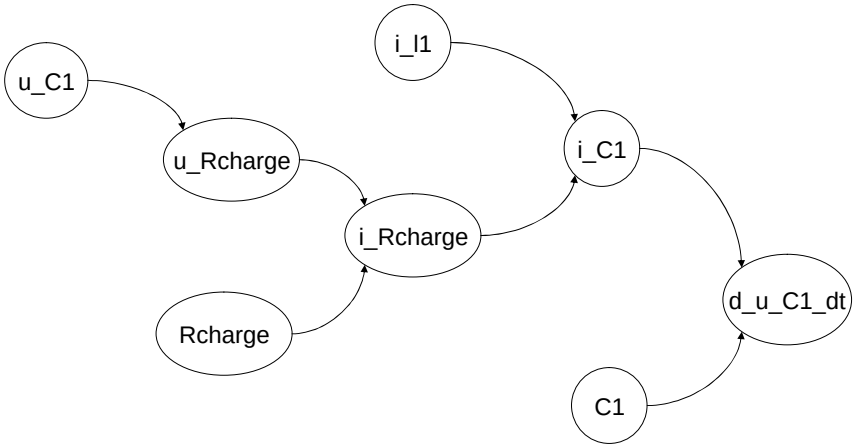


Figure 3. 10: Exemple d'arbre de dépendance

Nous déduisons de cet arbre de dépendance que la dérivée *d_u_C1_dt* dépend de *C1*, *i_l1*, *Rcharge* et *u_C1*.

D - Passage de la boule à la boite

Le passage du formalisme boule à celui des boites est automatisé à partir de choix faits par le concepteur.

Cette activité se compose de deux aspects fortement imbriqués :

- les choix du concepteur, qui dépendent de l'orientation à donner au modèle
- le travail formel à effectuer sur le modèle pour préparer l'écriture d'un code informatique.

D - 1 Choix d'un contexte d'utilisation

La première étape de la contextualisation est le typage des grandeurs qui le décrivent en termes de paramètres physiques $\mathbf{P_P}$ ou numériques $\mathbf{P_N}$, d'entrées **E** ou de sorties **S**. Cette étape est le processus qui permet de passer d'une boule à une boite tel que cela est illustré sur la Figure 3. 11.

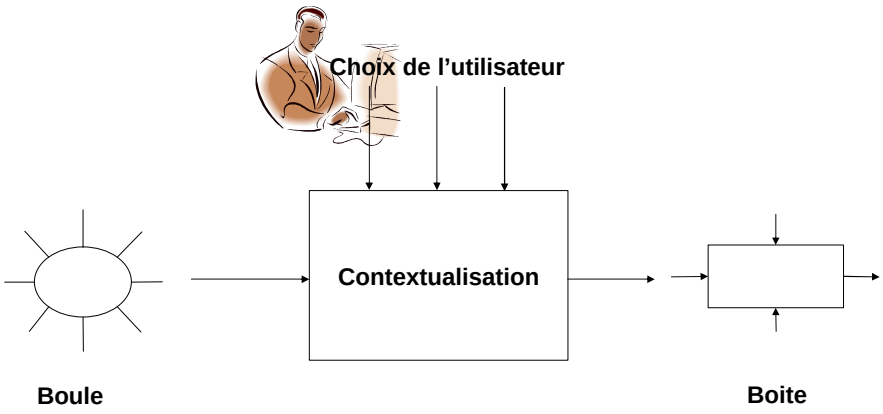


Figure 3. 11: Contextualisation d'un modèle

D - 2 L'orientation d'un modèle

Lorsque le concepteur doit réaliser cette opération sur un modèle comportant un grand nombre de relations, l'opération devient rapidement fastidieuse et source d'erreurs avec, par exemple, la présence de cycles [WUR] dans les équations du modèle ce qui peut nécessiter une reformulation du modèle pour résoudre un système d'équations. Nous proposons donc une approche systématique et automatisée pour réaliser ces opérations.

Le synoptique de la Figure 3. 12 illustre l'enchaînement des opérations réalisées ainsi que les rôles que nous donnons au concepteur et à l'outil de reformulation des modèles.

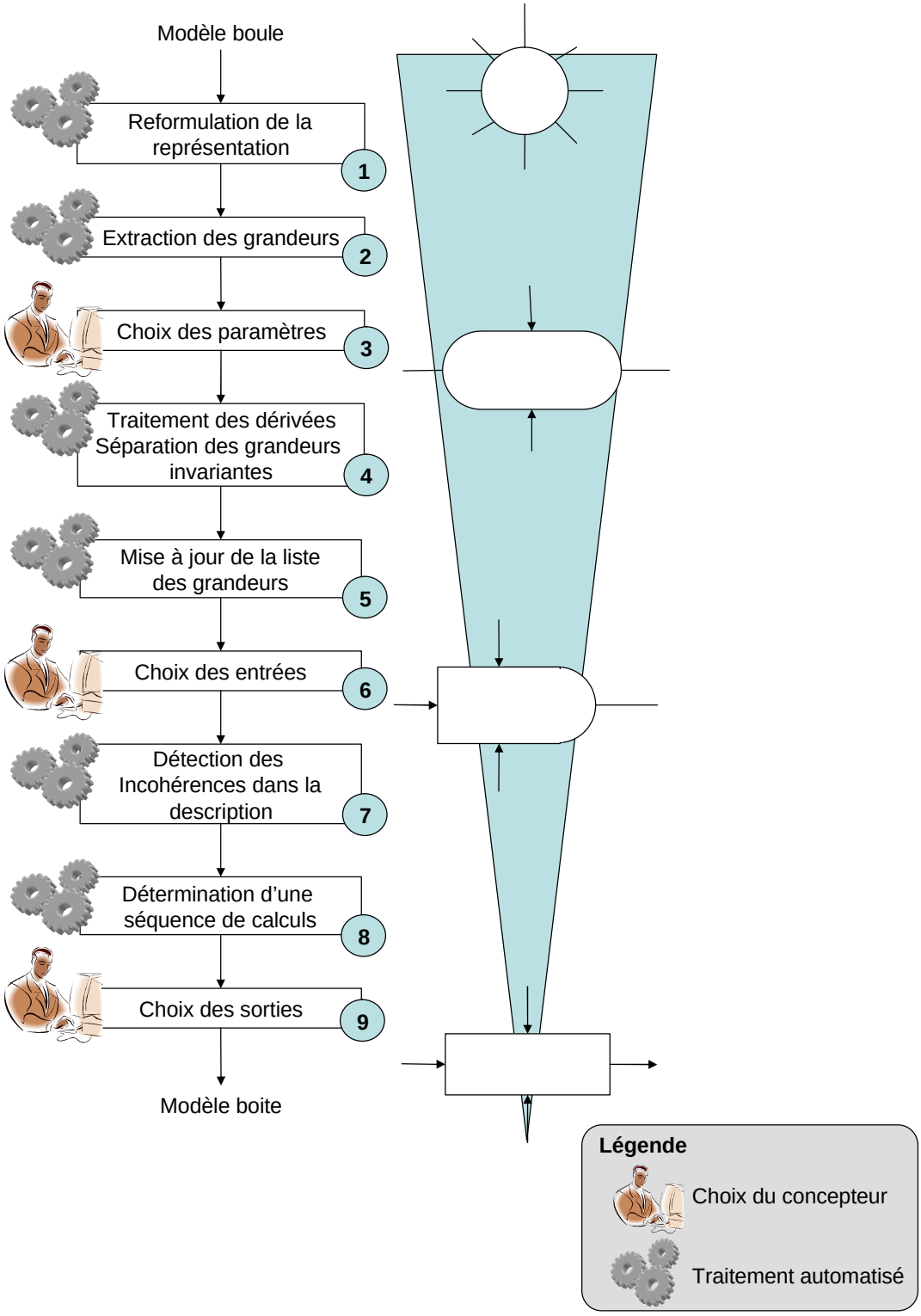


Figure 3. 12: Processus de transformation d'une "boule" en "boite"

Nous présentons sur la Figure 3. 12, le processus de transformation des modèles boules en boite :

- l'outil reformule les modèles **(1)**, c'est-à-dire qu'il reprend et modifie les relations pour avoir une forme exploitable automatiquement.

- l'outil extrait la liste des variables intervenant dans la description du modèle **(2)**
- le concepteur choisit un jeu de grandeurs qu'il type en paramètres $\mathbf{P_P}$. Par exemple, dans une approche classique, la valeur de la résistance d'un conducteur est constante au cours du temps et la tension à ses bornes et le courant la traversant sont des variables **(3)**.
- l'outil réalise alors la dérivation temporelle des expressions **(4)** et pour identifier l'ensemble des grandeurs invariantes. Cette action impose de modifier la représentation des relations.
- l'outil actualise la liste des grandeurs du modèle **(5)** afin de limiter les choix du concepteur
- le concepteur sélectionne un jeu d'entrées **E** pour la boite **(6)**
- l'outil détecte alors d'éventuelles incohérences dans la description du modèle **(7)**
- l'outil détermine ensuite une séquence de calculs, c'est-à-dire qu'il ordonnance les résolutions numériques **(8)**
- le concepteur choisit les sorties **S** de la boite **(9)**.

D - 2 - 1 Reformulation du modèle

La formulation choisie pour la capitalisation des modèles sous formes de boules n'est pas efficace pour un traitement automatique de la transformation. La première étape du processus de transformation de la boule est donc une reformulation symbolique dont le résultat fait apparaître les grandeurs de bases. Ainsi, nous:

- reformulons les notions d'héritage avec et sans surcharge de la modélisation orientée objet
- reformulons les notions d'agrégation ou de composition des modèles entre eux
- préparons des zones de calculs qui correspondent à chaque combinaison des états des conditions.

Les reformulations qui viennent de la modélisation Orientée Objet (héritage et agrégation) sont décrites dans l'annexe D de ce rapport. Nous nous concentrons ici sur la création des zones de calculs.

i)Le cas des systèmes hybrides

Lors du traitement d'un système hybride, nous créons une liste de systèmes d'équations qui reprennent l'ensemble des relations qui caractérisent chaque état.

Les états globaux sont construits par combinaison des différents états (STATE) des conditions (CONDITION) du modèle boule avec les équations originelles de la description.
Ceci est illustré sur la Figure 3. 14.

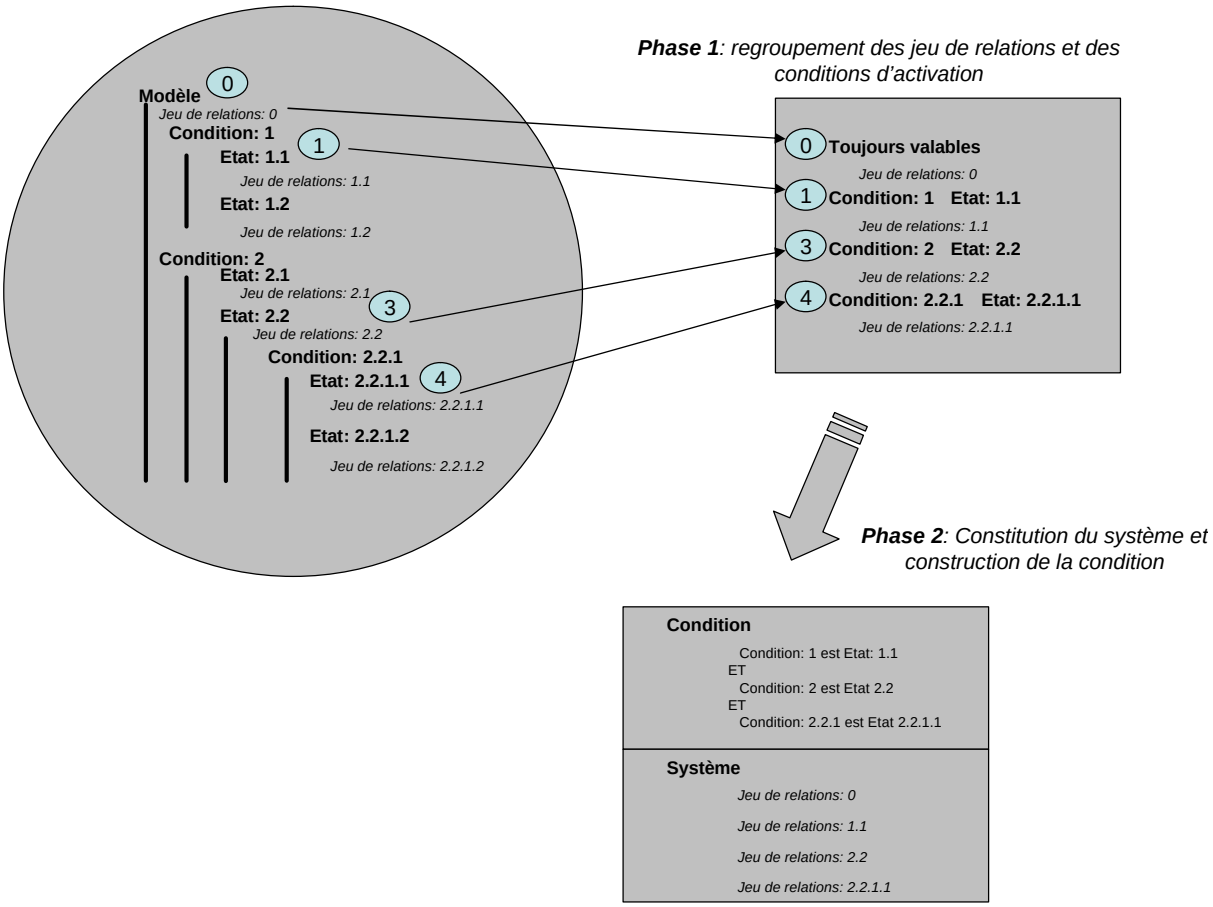


Figure 3. 13: Illustration de la démarche de construction d'un état pour un modèle hybride

Pour construire une zone du système hybride, nous proposons de :

- partir de la racine de l'arbre des conditions. C'est-à-dire que nous prenons les relations qui sont toujours « ***actives*** ». Par exemple, dans le cas du hacheur, nous commençons la construction des systèmes en incluant l'ensemble des relations qui caractérisent les composants du système.
- puis, nous considérons chacune des conditions filles pour construire un système comprenant la totalité des relations actives lorsqu'un ensemble de conditions est vérifié. Enfin, nous progressons de cette façon en descendant dans l'arbre.

Nous montrons le résultat de la construction d'une *zone* dans le cas du hacheur série sur la Figure 3. 14 :

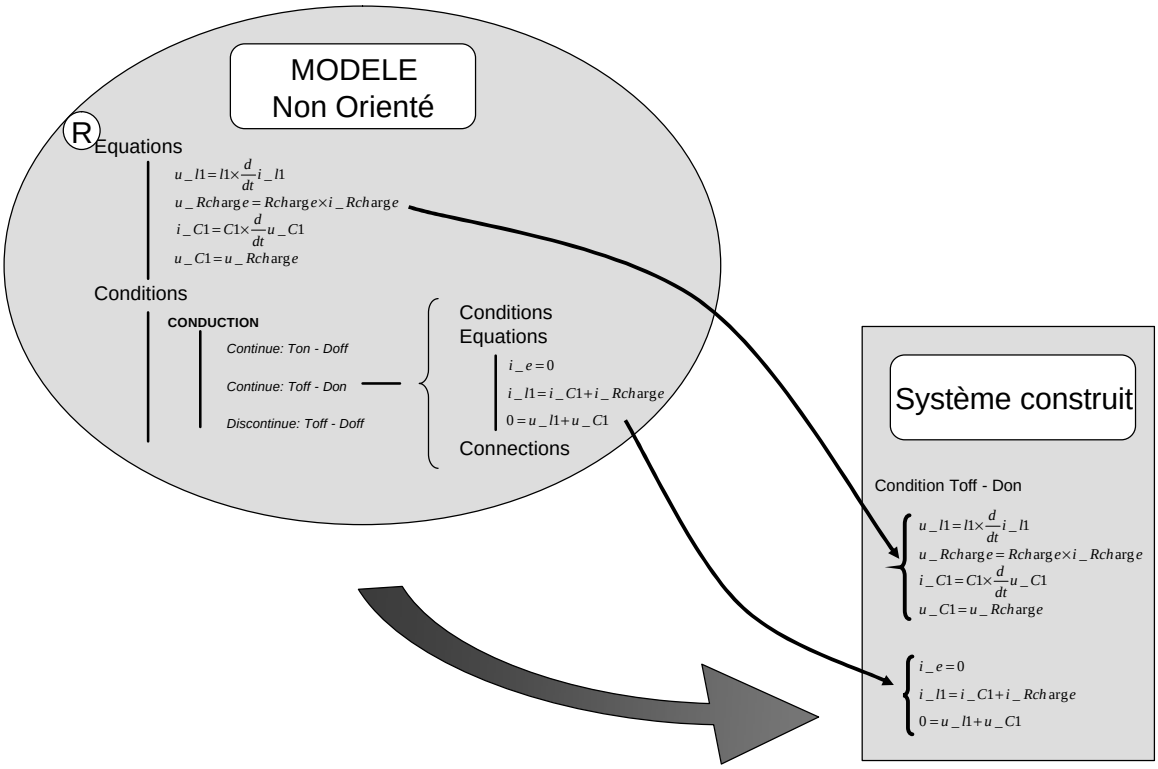


Figure 3. 14: Représentation d'un état du modèle du hacheur série

Sur cette figure nous représentons le résultat de la construction de la zone correspondant à l'état « Don – Toff » du hacheur où nous reprenons les relations principales et équations conditionnelles.

Cet algorithme de construction est illustré sur la Figure 3. 15.

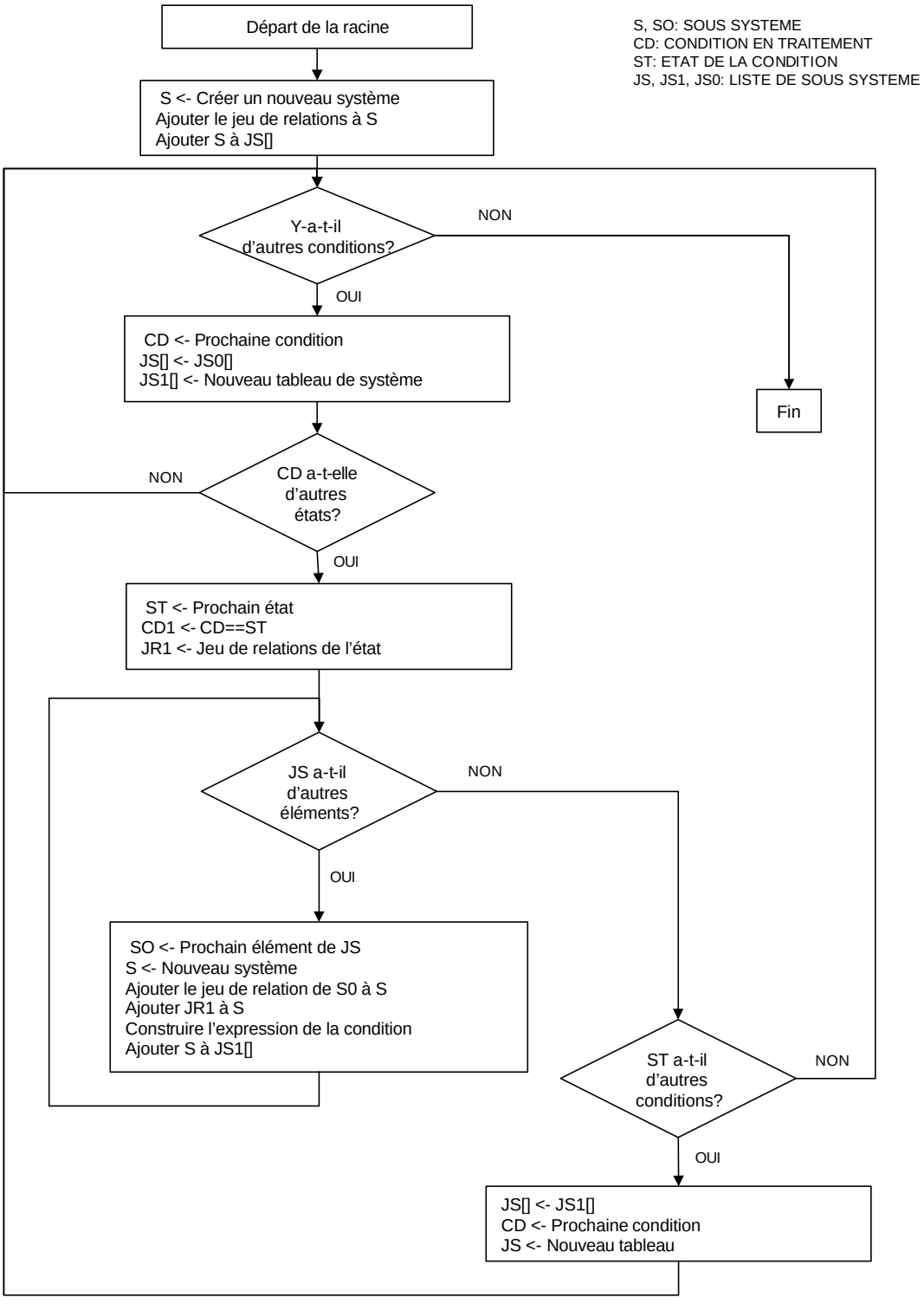


Figure 3. 15: Présentation de l'algorithme proposé pour la construction des sous-systèmes d'un modèle mixte

D - 2 - 2 Choix des paramètres

La liste des grandeurs intervenant dans la description du modèle est automatiquement extraite, puis nous proposons au concepteur de choisir un jeu de grandeurs qui seront des paramètres physiques $\boldsymbol{P_P}$ pour la boite de simulation.

Avec cette information, nous pouvons désormais :

- construire les dérivées temporelles des expressions du système
- déterminer l'ensemble des grandeurs qui sont fixées par les paramètres
- construire une séquence de calcul pour cet ensemble de grandeurs

La Figure 3. 16 illustre l'enchaînement des actions que nous automatisons.

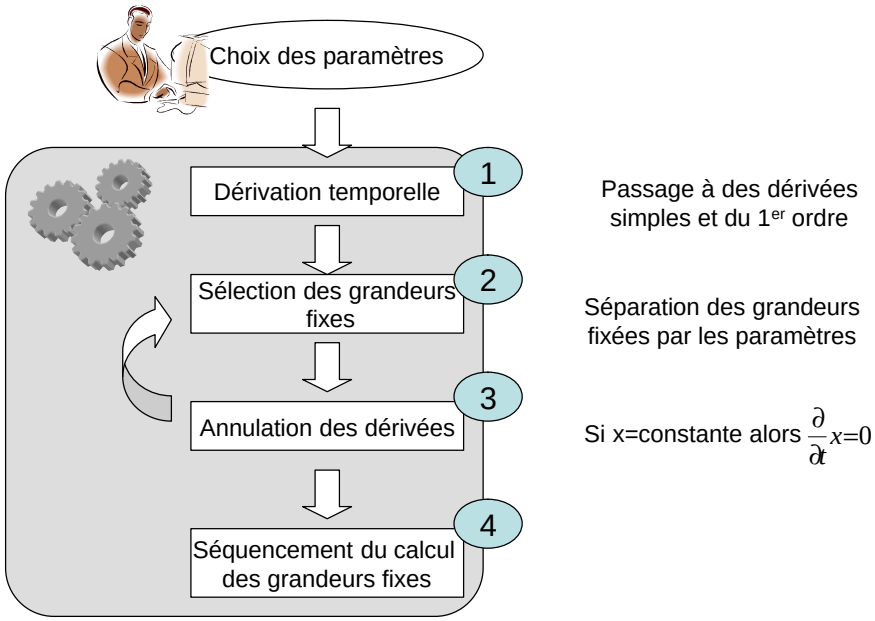


Figure 3. 16: Enchainement des actions sur le modèle après le choix des paramètres

Avec la connaissance des paramètres du système, nous construisons les dérivées simples et du premier ordre des expressions dérivées par rapport au temps **(1)**. Puis, nous identifions les grandeurs qui ne sont pas fonction du temps **(2)**, car elles sont fixées par les paramètres. Si des grandeurs d'état **X** sont fixées par les paramètres, alors nous annulons leurs dérivées successives **(3)**. Enfin, dès que l'ensemble des grandeurs ne dépendant pas du temps est identifiés, nous construisons une séquence de calculs numériques **(4)**.

2 - a) Dérivation temporelle (1)

Il s'agit de transformer les relations du modèle de façon à n'avoir que des dérivées simples et du premier ordre. Nous utilisons des techniques de calcul formel pour modifier l'écriture des relations. Il est nécessaire de modifier cette écriture car, les procédures classiques d'intégration temporelles ont besoin de la valeur des grandeurs dérivées.

- La **première opération** de la dérivation temporelle est la création des dérivées simples. Par exemple, l'expression

$$U = \frac{d}{dt}(L \times I)$$

est transformée, si L est un paramètre physique, en :

$$U = L \times dI\_dt$$

- la **seconde opération** est la réduction des dérivées d'ordre supérieur à 1 en dérivées d'ordre 1. Pour cela, nous créons automatiquement des variables intermédiaires. L'exemple le plus simple est celui de l'équation dynamique d'un arbre rotor :

$$J \times \frac{d}{dt}\left(\frac{d}{dt}\Theta\right) = \Gamma_e - \Gamma_{charge}$$

dans ce cas cette équation est modifiée en

$$J \times d\_tempo1\_dt = \Gamma_e - \Gamma_{charge}$$

et nous ajoutons la relation suivante à la description du modèle :

$$tempo1 = d\Theta\_dt$$

Remarque technique : *La notation choisie marque les grandeurs dérivées d'un préfixe '**d_**' et d'un suffixe '**_dt**'. Cette notation est utilisée pour augmenter la lisibilité du modèle sous la forme du code généré. Après avoir modifié l'écriture du modèle, nous construisons le vecteur d'état du modèle, en effet, cette connaissance nous est utile pour la suite afin de déterminer le nombre et la nature des inconnues dans la résolution numérique de son modèle.*

2 - b) Sélection des grandeurs fixées (2)

Afin de produire un code informatique qui soit le plus rapide possible, il est important d'éviter de réaliser des opérations inutiles, en particulier de répéter inutilement des opérations numériques dont le résultat est constant. Pour pouvoir prendre en compte cette contrainte, nous séparons les équations qui relient des grandeurs variant dans le temps de celles qui sont fixées par les paramètres.

Nous présentons en Figure 3. 17 l'algorithme permettant de séparer les grandeurs ne dépendant pas du temps de celles qui en dépendent.

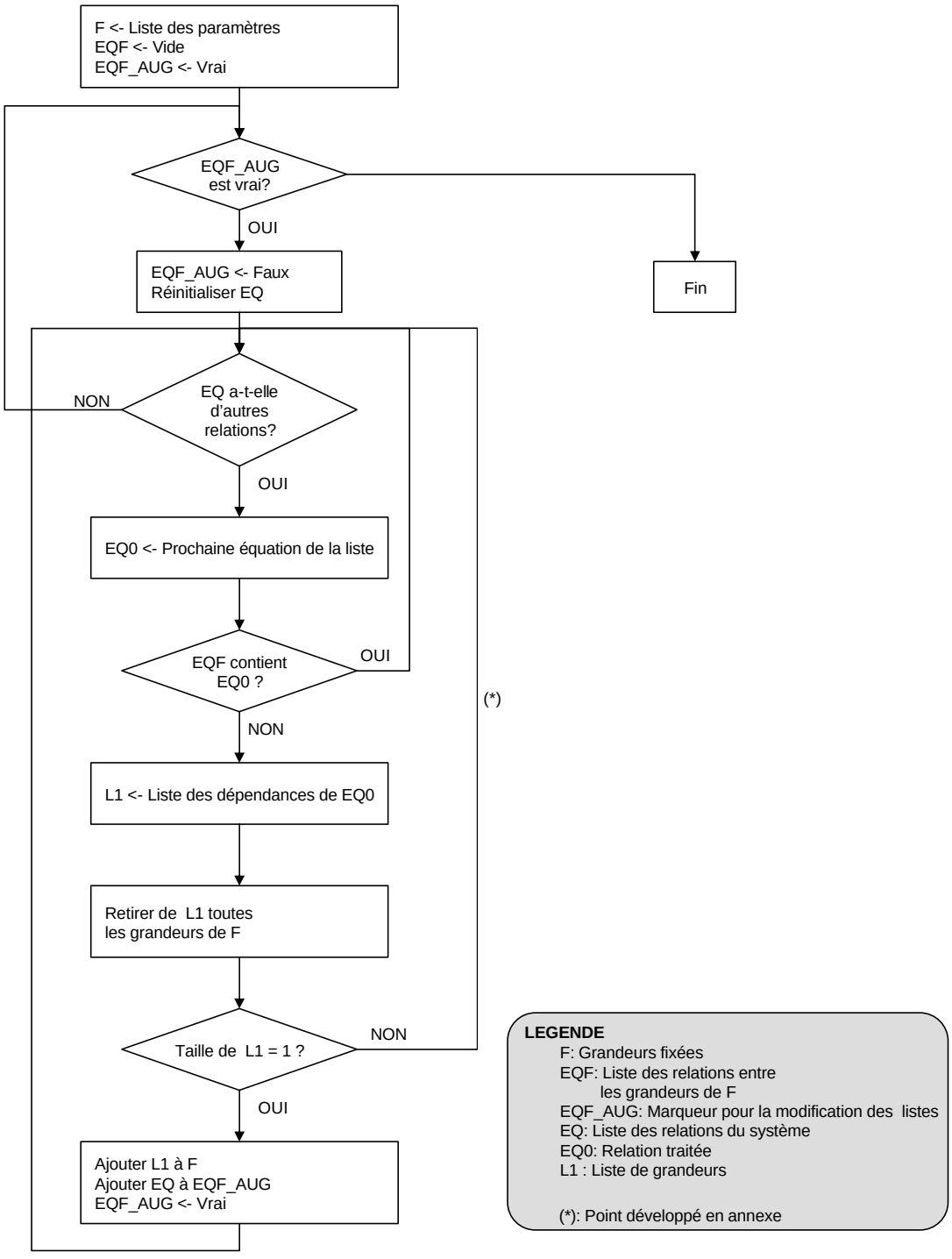


Figure 3. 17: Algorithme pour la séparation des grandeurs fixées par les paramètres des variables

L'algorithme présenté sur cette figure est limité, notamment en ce qui concerne la détection des systèmes de relations entre variables fixes. La solution de ce problème est expliquée en annexe C de ce rapport.

Le résultat de ce traitement est une liste de grandeurs qui ne seront évaluées qu'une fois, lors du processus d'intégration numérique sur le temps.

2 - c) Annulation des dérivées temporelles (3)

Il est possible que dans le résultat de cette opération, des grandeurs d'états soient fixées à une valeur ou par les paramètres (c'est par exemple le cas du courant dans l'inductance du hacheur série lors de la mise en conduction discontinue de la structure).
Si des grandeurs d'états sont fixées, il faut annuler leurs dérivées successives. Cette annulation peut entraîner que d'autres grandeurs soient fixées et ne dépendent donc plus du temps. Il est alors nécessaire de reboucler sur l'étape de recherche des grandeurs fixées afin de rendre possible l'écriture d'un code de calcul optimal.

2 - d) Séquencement des calculs des grandeurs ne dépendant pas du temps(4)

Lorsque les grandeurs fixées par les paramètres sont identifiées ainsi que les relations qui les lient, nous déterminons une séquence optimale de calcul, en effet, en terme d'opérations numériques, il est préférable de disposer d'une séquence d'opérations élémentaires plutôt que de devoir résoudre un gros problème numérique. Duff [DUF] propose une adaptation de l'algorithme de Tarjan [TAR] concernant la séparation des composantes connexes d'un graphe pour la triangularisation par blocs des matrices creuses. Nous exploitons cet algorithme sur une représentation sous forme de matrice d'incidence. Dans ce contexte il permet de :

- déterminer des associations entre les grandeurs et relations
- déterminer une séquence de calcul par une méthode d'ordonnancement de tâches.

L'algorithme complet est présenté en annexe C.
La Figure 3. 18 illustre les résultats attendus de l'application de cet algorithme dans le cas du jeu de relations fixées par les paramètres pour la zone de conduction discontinue du hacheur série.

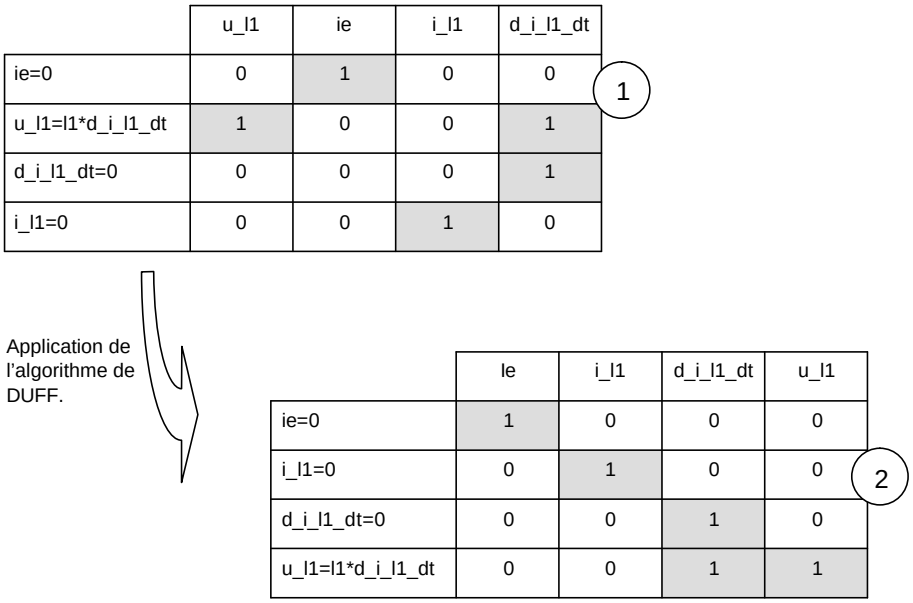


Figure 3. 18: Séquencement des calculs indépendants du temps pour la zone Toff – Doff du hacheur série

La matrice d'incidence **(1)** montre la structure du modèle avant l'ordonnancement des équations pour la résolution du problème. La matrice **(2)** présente le résultat de l'application de l'algorithme de triangularisation inférieure de Duff. Il apparaît alors une séquence de calcul dans la laquelle la variable de la colonne ***i*** est évaluée grâce à l'équation de la ligne de même indice. Il est ainsi possible de réaliser des opérations élémentaires plutôt que la résolution d'un problème numérique de taille conséquente.

La dernière étape de ce processus est la détermination de méthodes numériques à mettre en œuvre pour le calcul du problème. Ce point est détaillé au paragraphe D-2-3-c de ce chapitre.

D - 2 - 3 Le choix des entrées

Une fois les paramètres choisis et le vecteur d'état construit, nous proposons au concepteur une liste mise à jour des grandeurs du système (c'est-à-dire que les grandeurs fixées et les grandeurs d'états sont retirées) pour qu'il choisisse les entrées de son système. Dans le cas du circuit RLC série, l'entrée naturelle du système est la source de tension ; le concepteur choisit donc de donner E comme entrée de son circuit. Notons que la description que nous donnons pour le circuit RLC est classique, en revanche, le concepteur aurait fort bien pu choisir une autre entrée pour son dispositif, comme par exemple la tension au bornes de la résistance.

A l'aide de cette information, nous pouvons réaliser un certain nombre de traitements formels sur le modèle :

- détection des incohérences dans la description du modèle
- réduction du système d'état

- séquencement de opérations numériques

3 - a) Détection des incohérences

Lorsque le concepteur a choisi les entrées de sa boite, nous vérifions que les choix faits ne sont pas incompatibles avec les grandeurs d'état **X** détectées précédemment. En effet, il est déconseillé d'avoir des relations directes (ou indirectes) entre les grandeurs d'états (**X**) et les grandeurs d'entrée et en particulier avec les grandeurs d'excitations (**U**). Pourtant, des cas se présentent pour lesquels ces relations directes sont autorisées. Dans ce cas, le concepteur doit être capable de fournir une donnée supplémentaire : l'évolution de la dérivée temporelle de l'excitation. En pratique, nous rajoutons une entrée à la boite de simulation.

En électricité, par exemple, ce cas peut se présenter lors du branchement du réseau illustré sur la Figure 3. 19 a. qui montre une connexion interdite, tandis que la Figure 3. 19 b illustre un exemple de connexion qui peut être acceptée.

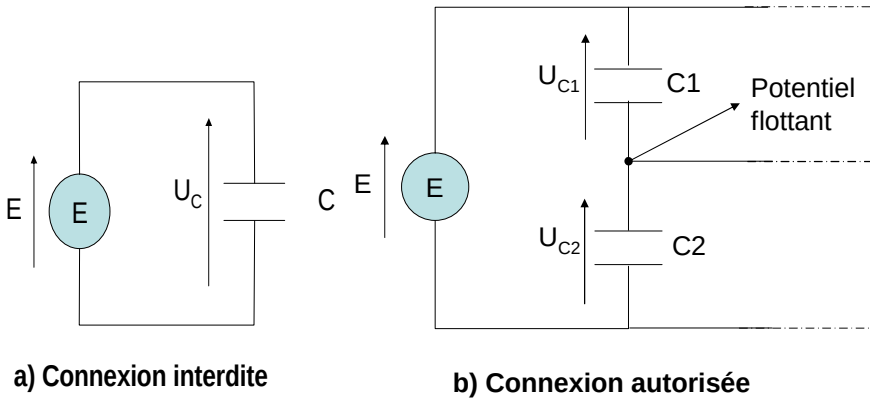


Figure 3. 19: Exemple de relations: connexion d'une source de tension sur des condensateurs

Dans le cas ***a)***, la tension U_C qui est une grandeur d'état est imposée par la source, ce type de configuration est interdit. En revanche le cas ***b)*** autorise la connexion, en raison de la présence d'un potentiel flottant. Dans ce cas, pour des raisons numériques, nous rajoutons une entrée supplémentaire : la dérivée du signal d'excitation (dE/dt dans l'exemple).

La recherche de ces incohérences potentielles se fait en deux temps :

- la recherche de l'ensemble des grandeurs équivalentes à des excitations (**U**). Cette recherche se fait en utilisant la même technique que pour rechercher les grandeurs fixées par les paramètres.
- Repérer les points communs entre la liste des grandeurs dérivées et la liste des grandeurs équivalentes à des entrées.

3 - b) Détermination du vecteur d'état minimal

i) Présentation de la réduction du système d'état

Nous avons présenté dans le paragraphe C - 1 - 2 le principe de résolution des problèmes de simulation. Il apparaît que les moteurs de résolution travaillent sur des matrices dont le temps de traitement est fonction de la taille de la liste des grandeurs dérivées (**X**). Nous avons donc tout intérêt à déterminer une taille minimale pour cette liste.

De nombreux dispositifs sont décrits par des équations qui permettent d'établir des relations linéaires entre les différentes grandeurs dérivées, nous utilisons ces relations pour diminuer la taille du vecteur des grandeurs dérivées. C'est l'étape que nous appelons réduction du vecteur d'état. Cette étape donne une représentation du modèle faisant intervenir un nombre minimal de grandeurs dérivées.

Nous découpons cette phase du traitement du modèle en 3 étapes intermédiaires :

- la constitution d'un ensemble de relations susceptibles de permettre de construire des équivalences entre grandeurs dérivées
- la sélection d'un jeu minimal de grandeurs dérivées et la construction des relations d'équivalences entre les grandeurs retenues et les autres
- la dérivation de la relation d'équivalence et la substitution dans les relations de toutes les occurrences de dérivées des grandeurs non retenues

ii) Présentation d'un exemple de réduction

Nous détaillons maintenant chacune de ces étapes en nous appuyant sur la recherche du vecteur d'état minimal pour le circuit illustré sur la Figure 3. 20.

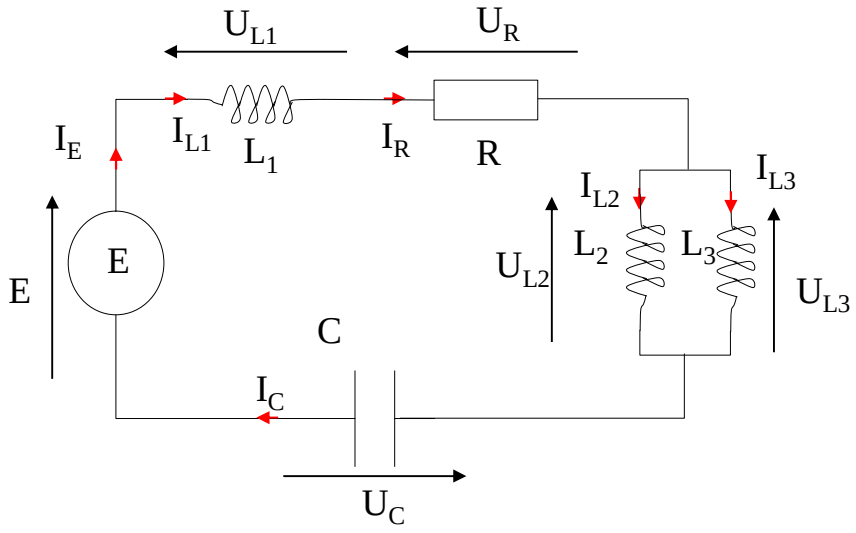


Figure 3. 20:Circuit électrique utilisé pour illustrer la détermination d'un système d'état minimal

Les relations utilisées dans la suite de l'explication sont :

$$\begin{cases} E = U_C + U_R + U_{L1} + U_{L2} \\ I_E = I_{L1} \\ I_{L1} = I_R \\ I_{L2} + I_{L3} = I_R \\ I_{L2} + I_{L3} = I_C \\ U_{L2} = U_{L3} \end{cases} \qquad \begin{aligned} I_C &= \frac{1}{C} \cdot \frac{d}{dt} U_C \\ U_{L1} &= L_1 \cdot \frac{d}{dt} I_{L1} \\ U_{L2} &= L_2 \cdot \frac{d}{dt} I_{L2} \\ U_{L3} &= L_3 \cdot \frac{d}{dt} I_{L3} \\ U_R &= R \cdot I_R \end{aligned}$$

iii) Constitution d'un ensemble de relations utilisables

Pour la détermination d'une taille minimale du vecteur d'état, il est nécessaire de disposer d'un ensemble d'équations qui sont susceptibles de construire des relations entre les grandeurs dérivées. Etant donné que cette opération nécessite de nombreuses inversions d'équations ne donnant qu'une seule solution, nous choisissons de ne retenir dans cet ensemble que les relations linéaires par rapport aux grandeurs dérivées et non par rapport à leurs dérivées.

Dans l'exemple sur lequel nous nous appuyons, les relations qui sont retenues sont les suivantes :

$$\begin{cases} E = U_C + U_R + U_{L1} + U_{L2} \\ I_E = I_{L1} \\ I_{L1} = I_R \\ I_{L2} + I_{L3} = I_R \\ I_{L2} + I_{L3} = I_C \\ U_{L2} = U_{L3} \end{cases}$$

iv) Sélection d'un jeu minimal de grandeurs dérivées

Puis, nous opérons sur la matrice d'incidence du système constitué par ces relations et les variables qui interviennent dans ces relations. Cette matrice est une image du graphe constitué par les relations et les grandeurs. Nous donnons sur la Figure 3. 21 l'exemple de matrice d'incidence utilisée pour la réduction du système d'état pour le circuit considéré en Figure 3. 20.

	E	Ur	Uc	UL1	UL2	Ie	Ic	Ir	Il1	Il2	Il3
$E = U_C + U_R + U_{L1} + U_{L2}$	1	1	1	1	1						
$I_E = I_{L1}$						1			1		
$I_{L1} = I_R$								1	1		
$I_{L2} + I_{L3} = I_R$								1		1	1
$I_{L2} + I_{L3} = I_C$							1			1	1

Figure 3. 21 : Matrice d'incidence utilisée pour la détermination d'un vecteur d'état minimal

Dans cet exemple, nous reconnaissons que si I_{L3} est connu, alors I_{L2} peut être déduite de la connaissance de I_R et donc de I_{L1}.

En d'autres termes, cela signifie qu'une relation entre ***n*** grandeurs dérivées permet de déduire une équivalence entre ***n-1*** grandeurs et la dernière. Cette propriété est très largement utilisée dans l'algorithme qui est illustré en Figure 3. 22.

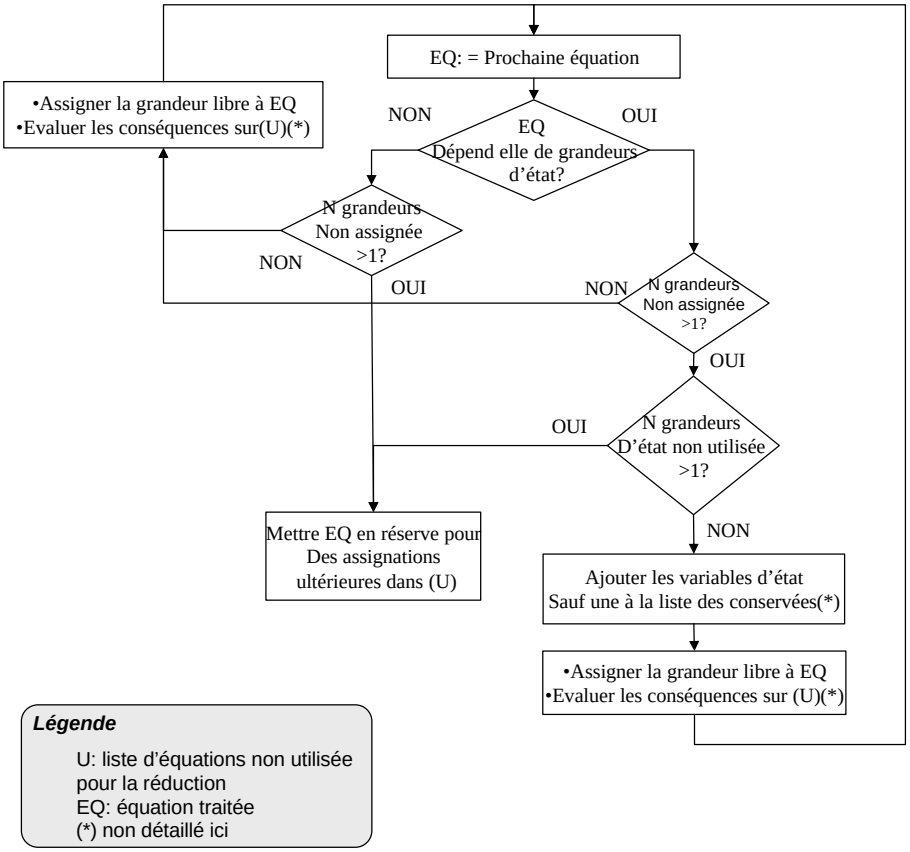


Figure 3. 22: Algorithme proposé pour la détection d'un vecteur d'état minimal

Le principe de base de cet algorithme est une observation systématique des relations pour détecter la possibilité de réduire le nombre de grandeurs dérivées en trouvant des relations linéaires entre ces différentes grandeurs.

Si une relation contient une grandeur dérivée ou une grandeur équivalente, il est possible de construire une relation permettant d'en éliminer une. Dans ce cas, une équivalence est construite qui donne une relation entre la grandeur non retenue et les autres, par exemple :

$$I_{L3} = I_{L1} - I_{L2}$$

Le résultat de cette étape pour l'exemple présenté est le choix des grandeurs I_{L1}, I_{L2} et U_C pour la constitution du vecteur d'état « *réduit* ».

***Remarque** : L'ensemble de l'algorithme n'est pas présenté ici pour des raisons de lisibilité. Ceci est complété dans l'annexe C.*

v) Substitution

Afin de réduire le nombre de grandeur dérivées intervenant dans la description du modèle, nous modifions les équations du modèles en remplaçant systématiquement les occurrences des dérivées des grandeurs par la dérivées de la relation d'équivalence construite. Ainsi dans l'exemple donné, ce sont les occurrences de ***di_l3_dt*** qui sont remplacées par ***di_l1_dt-di_l2_dt***. Le système d'équations est ainsi modifié en :

$$\begin{cases} E = U_C + U_R + U_{L1} + U_{L2} \\ I_E = I_{L1} \\ I_{L1} = I_R \\ I_{L2} + I_{L3} = I_R \\ I_{L1} = I_C \\ U_{L3} = U_{L2} \end{cases} \qquad \begin{aligned} I_C &= \frac{1}{C}\frac{d}{dt}U_C \\ U_{L1} &= L_1 \frac{d}{dt} I_{L1} \\ U_{L2} &= L_2 \frac{d}{dt} I_{L2} \\ U_{L3} &= L_3\left(\frac{d}{dt} I_{L1} - \frac{d}{dt} I_{L2}\right) \\ U_R &= R \times I_R \end{aligned}$$

Maintenant que nous disposons d'un ensemble d'équations qui contient un nombre minimal de grandeurs dérivées, il est encore nécessaire de construire une séquence optimale de calculs pour l'évaluation du modèle.

3 - c) Détermination d'une séquence optimale d'opérations numériques

i) Le séquencement

Cette opération utilise la même technique que celle que nous avons présentée au paragraphe D-2-2-d de ce chapitre.

Nous utilisons donc l'algorithme proposé par Tarjan pour la séparation des composantes connexes d'un graphe. Cet algorithme nous permet de construire une séquence optimale d'opérations numériques par la spécification d'une séquence de résolution de couple 'variables-relations' illustrée sur la Figure 3. 23

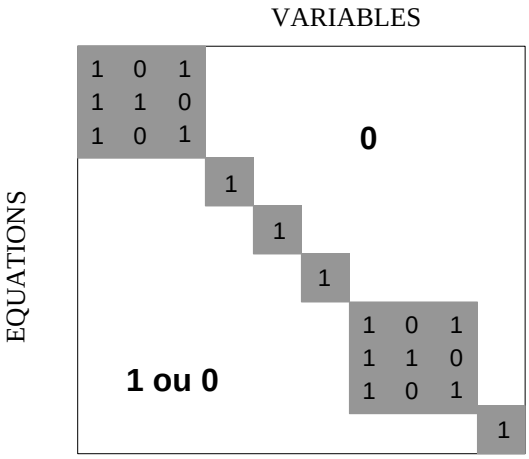


Figure 3. 23: Visualisation de la séquence de calcul optimale

ii) Méthode de résolution

Une fois que les appairages sont réalisés et que la séquence de calcul est finalisée, il est encore nécessaire de déterminer le type de résolution que nous utiliserons pour chaque élément de résolution. C'est-à-dire pour chaque « blocs » de la diagonale sur la matrice d'occurrence résultant de l'ordonnancement.

Par « type de résolution », nous voulons préciser la nature de l'opération numérique à réaliser, en effet, dans de nombreux cas, il est possible d'expliciter la résolution de l'équations pour en extraire l'expression de la grandeur appariée. Mais, il existe des cas dans lesquels cette explicitation est difficile, voire impossible ; il s'agit alors de problèmes implicites.

La même démarche peut être appliquée pour la résolution des blocs constitués de plus d'une équation : les systèmes d'équations. Dans ce cas, la résolution numérique utilisera soit des algorithmes classiques d'inversion de systèmes linéaires ou bien des résolutions de problèmes implicites (par exemple Newton-Raphson). L'information est, dans tous les cas, conservée pour être réutilisée pour la dernière étape du processus de contextualisation : la génération de code.

D - 2 - 4 Choix des sorties

Le concepteur n'a plus qu'à choisir la liste des variables qu'il désire sortir de la boite. L'information est conservée dans la boite résultante.

E - Modélisation pour le calcul

Nous avons présenté les concepts de modélisation et de capitalisation des modèles pour la simulation. Il existe une grande similitude entre la modélisation pour la simulation et la modélisation pour le calcul. Aussi, dans cette partie, nous présentons la modélisation pour le calcul en relation avec ce que nous avons déjà vu.

E - 1 Nature des modèles

Les modèles utilisés pour le calcul sont constitués des mêmes éléments que les modèles de simulation. Par ailleurs, nous fournissons les mêmes types d'utilitaires de description : des fonctions, des tableaux utilisés pour faciliter la description du modèle.

Nous proposons l'utilisation des mêmes concepts et formats pour la capitalisation, des boules pour une capitalisation a-contextuelle des modèles et des boites pour les modèles orientés.

E - 2 La composition de boites de dimensionnement

Dans une optique de dimensionnement, la composition de modèles apparaît aussi primordiale que pour la description des modèles de simulations [DEL]. En effet, la composition apporte non seulement une certaine intelligibilité à des modèles complexes, mais elle permet également d'envisager une réutilisation de modèles existants pour des finalités différentes.
De même qu'en simulation, la composition par ***boites noire*** permet de répondre à une certaine problématique : essentiellement si le concepteur n'a pas la compétence ou la connaissance pour modifier les modèles. Si elle présente l'avantage considérable de la simplicité d'utilisation, elle se heurte à des problèmes numériques similaires aux problèmes de causalité rencontrés en simulation.
La composition de modèles en dimensionnement est intéressante, par exemple, lorsque les critères dimensionnant sont issus de différents points de fonctionnements ou dans le cas du dimensionnement d'une gamme. La Fig. 3. 24 illustre l'exemple du dimensionnement d'une gamme d'actionneurs électro-mécaniques dont deux points de fonctionnement sont dimensionnant. Ce modèle est construit par compositions de sous-modèles.

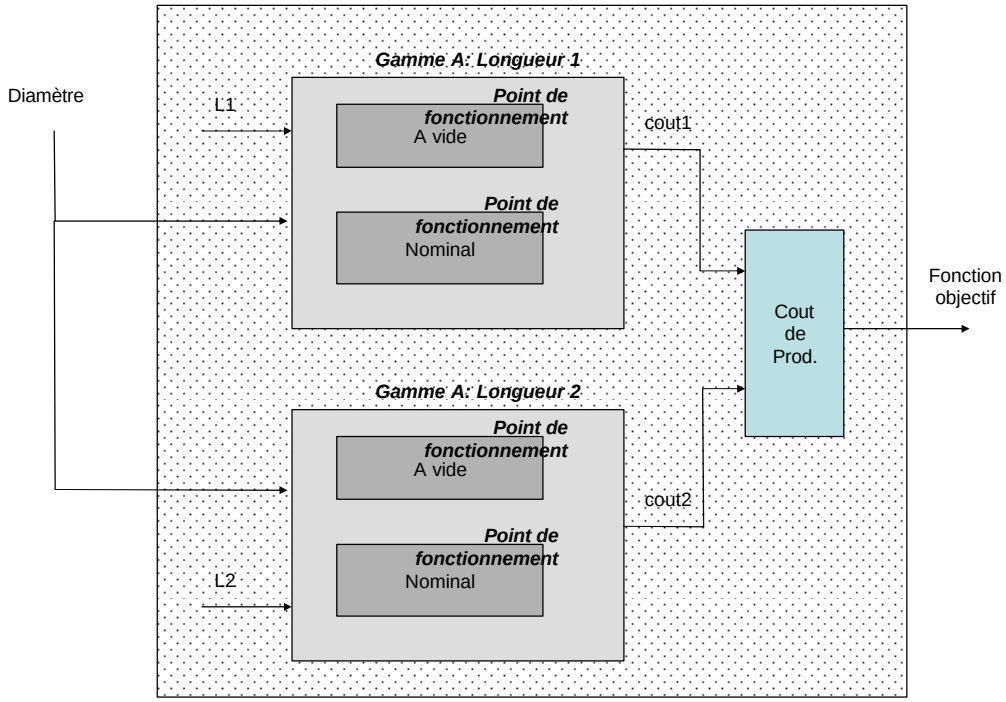


Fig. 3. 24 : Problème de dimensionnement d'une gamme d'actionneurs suivants plusieurs points de fonctionnement

Il est intéressant de noter que la composition proposée est récursive, c'est-à-dire que le résultat d'une composition de modèles peut être lui-même un modèle réutilisable dans un but de composition.

On suppose que le modèle de l'actionneur est connu pour un point de fonctionnement, les différents éléments de la gamme d'actionneurs ont des dimensions telles qu'ils restent dans le domaine de validité du modèle. Une gamme d'actionneur est caractérisée par un diamètre unique, décliné dans des longueurs différentes. L'objectif du dimensionnement est alors de minimiser le coût de l'ensemble de la gamme en conservant, par exemple, une découpe de tôle commune.

E - 3 Le cas particulier des modèles hybrides

L'originalité de cette modélisation est la possibilité de décrire des modèles de calculs associés avec des règles.

Nous reprenons les éléments de la modélisation hybride pour servir un objectif différent : la modélisation multi-structure. Un modèle de calcul peut ainsi être composé de différents sous-modèles qui sont considérés comme actifs en fonction de la validation d'une condition (hypothèse de validité du modèle). Cela permet, par exemple, de commuter le modèle de calcul en fonction de l'espace de recherche parcouru dans un outil de dimensionnement.

1 - a) Optimisation multi-structure

Cette approche peut être utilisée pour réaliser, par exemple, de l'optimisation avec un choix de structure. Nous illustrons cette possibilité sur la Figure 3. 25.

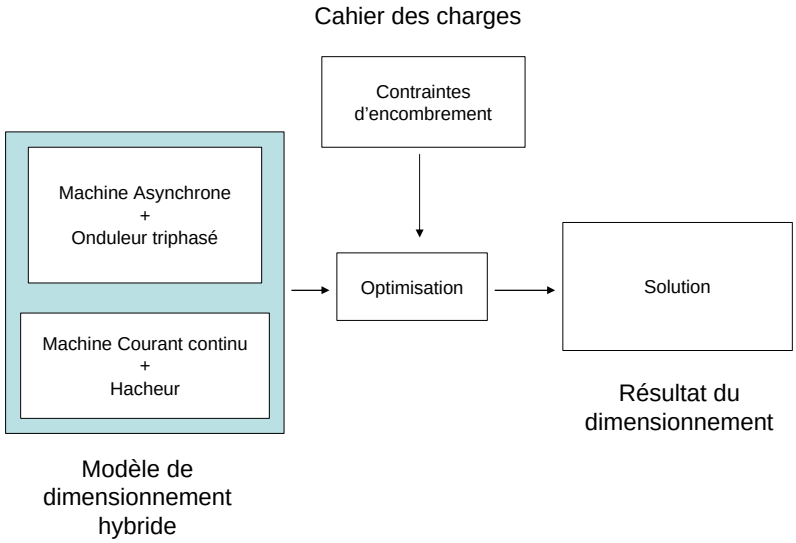


Figure 3. 25: Dimensionnement multi-structure

Le modèle de dimensionnement est composé de deux A.M.C.C., le choix entre les deux se fait en fonction d'un cahier des charges défini par le concepteur. Le résultat de l'optimisation donne la meilleure structure possible, par exemple, en prenant le meilleur des deux optimums de chaque A.M.C.C.

1 - b) Modèle macroscopiques et hypothèses

La majorité des modèles macroscopiques sont construits en posant des hypothèses ; par exemple de dimension, de non saturation des matériaux. En fonction de la validité des hypothèses, nous orientons la modélisation différemment, jusqu'à avoir des modèles différents en fonction de la validité des hypothèses. Or ces modèles représentent la même structure, l'utilisation des concepts de « *condition* » et d'« *état* » des modèles hybrides permet d'apporter une solution pertinente au problème de domaine de validité des modèles macroscopiques.

En utilisant cette approche, le concepteur dispose d'un modèle qui est valable sur une large plage de dimensions par le biais de sous-modèles qui sont utilisés en fonction de la justesse des hypothèses Figure 3. 26.

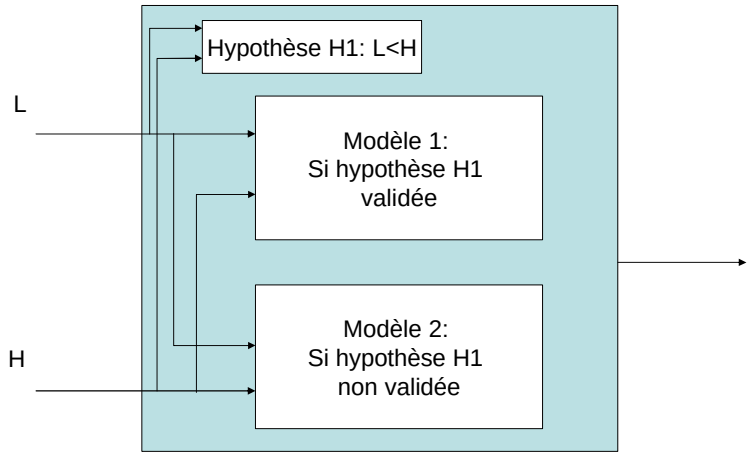


Figure 3. 26: Modèles macroscopiques et hypothèses

E - 4 Transformation de la boule en boite

Dans le cas du calcul, le processus de transformation fait intervenir le même nombre d'étapes. Dans le cadre du calcul, il est possible de différencier les entrées constantes du modèle des autres entrées. Ainsi pour un modèle utilisé dans le cadre de l'optimisation, il est possible de n'autoriser les optimisations que sur certains paramètres, en excluant, par exemple, les constantes physiques telles que la résistivité d'un matériau.

Nous retrouvons donc une analogie entre les paramètres ($\mathbf{P_P}$) de la simulation et les paramètres constants du calcul. De même, les entrées (**E**) sont les paramètres optimisables.

En ce qui concerne les traitements symboliques, il n'est pas nécessaire de déterminer un vecteur d'état minimal puisque la dimension temporelle n'intervient pas.

Le concepteur choisit des entrées constantes, des entrées optimisables et des sorties et nous automatisons l'ensemble des opérations restantes, essentiellement de l'ordonnancement de calcul.

F - Passage d'une boite à un bloc de calcul

F - 1 Introduction

La dernière étape de la contextualisation du modèle est la création d'un code de calcul dédié à un environnement. Nous voulons, ici encore, aider le concepteur dans cette tâche que nous automatisons. Cette étape est valable aussi bien pour la simulation temporelle que pour le calcul.

Pour l'automatisation de cette activité, nous définissons deux niveaux de compétences, illustrés sur la Figure 3. 28.

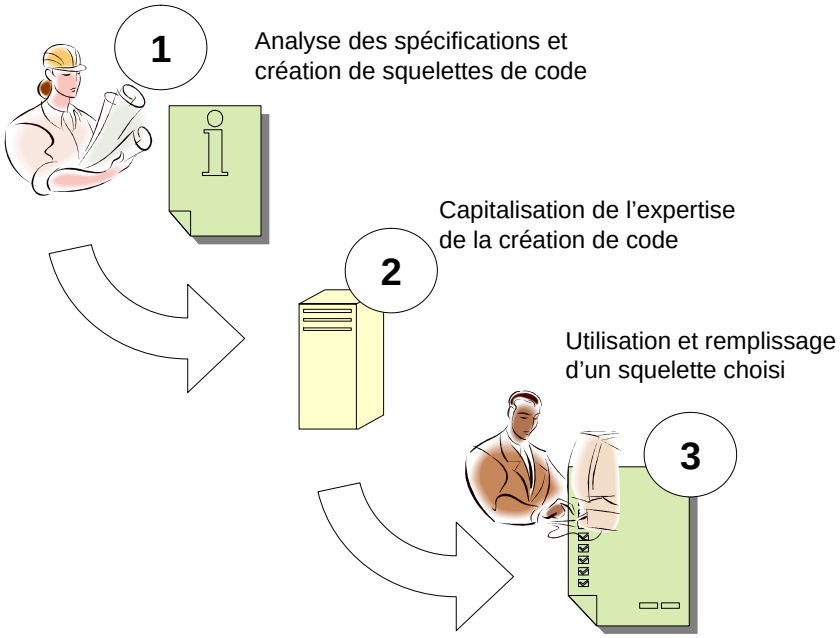


Figure 3. 27: Bilan de l'approche pour l'écriture de code

La production de code informatique ne relève pas du domaine de compétence du concepteur, en effet, il s'agit d'une activité pointue et consommatrice de temps. Par ailleurs, l'ensemble des activités de la création de code demande en général une expertise dans ce domaine (utilisation d'environnements spécifiques, connaissance des langages de programmation avancé,…)

Nous proposons donc de lui fournir une aide qui automatise l'écriture de ce code en lui évitant les embûches de cette activité (erreur de syntaxe, problème de compilation, bogue….).

Pour cela, nous nous appuyons sur la compétence d'un informaticien qui est chargé de comprendre les spécifications du code à produire et de proposer un squelette pour ce code **(1)**. Ce squelette est conservé **(2)** et utilisé par un outil de génération automatique de code **(3)**.

F - 2 Processus de création de code

La Figure 3. 28 illustre le cheminement que nous proposons de faire suivre au concepteur pour écrire le code permettant la résolution numérique de son modèle.

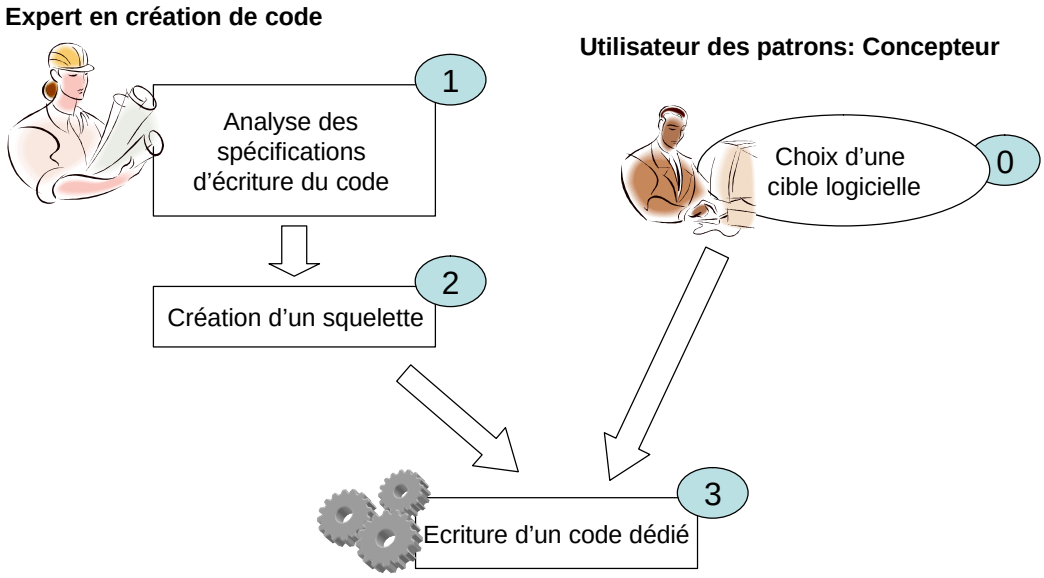


Figure 3. 28: Enchaînement des étapes pour la production d'un code de calcul

Le concepteur choisit un environnement logiciel pour l'utilisation de son modèle **(0)**. Si les travaux d'analyse de spécifications et de création de squelette sont déjà réalisés, il utilise un générateur de code pour produire le bloc de calcul désiré **(3)**. Dans le cas contraire, il fait appel à l'expertise d'un développeur professionnel pour l'analyse des spécifications du logiciel choisi **(1)** et créer un squelette **(2)**.

F - 3 Analyse des spécifications (1)

Afin de pouvoir « entrer dans le moule » imposé par l'environnement choisi, il est nécessaire d'en connaître les spécifications. Cette connaissance s'acquiert par l'analyse de la documentation couramment fournie ou l'expérience du développeur afin de permettre au concepteur d'augmenter la bibliothèque de blocs ou de composants de l'outil pour qu'elle corresponde à ses besoins propres.

Il s'agit d'un travail qui peut être effectué une fois par environnement visé et dont le résultat sera ensuite réutilisé. L'approche que nous proposons dans ce contexte est fondé sur la capitalisation de cette expertise d'écriture afin d'en faire profiter l'ensemble de la communauté dont le concepteur fait partie (entreprise, laboratoire, domaine de recherche, etc…)

F - 4 Création d'un squelette (2) : les patrons de génération

De nombreux environnements de calculs ou de simulation proposent des squelettes de code. En effet, il apparaît bien souvent qu'il existe de nombreux points communs entre les différents

codes développés pour un environnement choisi. Les modifications à introduire pour créer son propre modèle sont identifiées et bien localisées. Ainsi, par exemple, la fonction « **mdlInitialize() »** d'une S-Function de MATLAB/SIMULINK doit contenir les informations permettant l'initialisation du modèle, de la taille et la nature des entrées, des paramètres, etc… MATLAB/SIMULINK fournit par ailleurs des exemples de patrons à compléter par l'utilisateur.

Bien souvent, cette expertise peut être et est représentée sous la forme de squelette de code ; c'est-à-dire sous la forme d'un fichier contenant des informations permettant de reconstruire un code dédié par ajout d'informations spécifiques à des endroits identifiés. La création de ces squelettes, que nous appelons « patrons de génération », est le fait d'individus uniques dans leur communauté ayant consacré temps et énergie à leur construction. Ceux-ci sont ensuite exploités par les développeurs de modèles pour écrire leurs propres modèles dans le contexte de l'environnement. Les concepteurs disposent alors d'« une recette » pour l'écriture de leur code. Nous proposons de capitaliser ces patrons afin de pouvoir disposer d'une batterie de projecteurs vers des environnements différents remplissant des fonctions différentes. Ceci est illustré sur la Figure 3. 29.

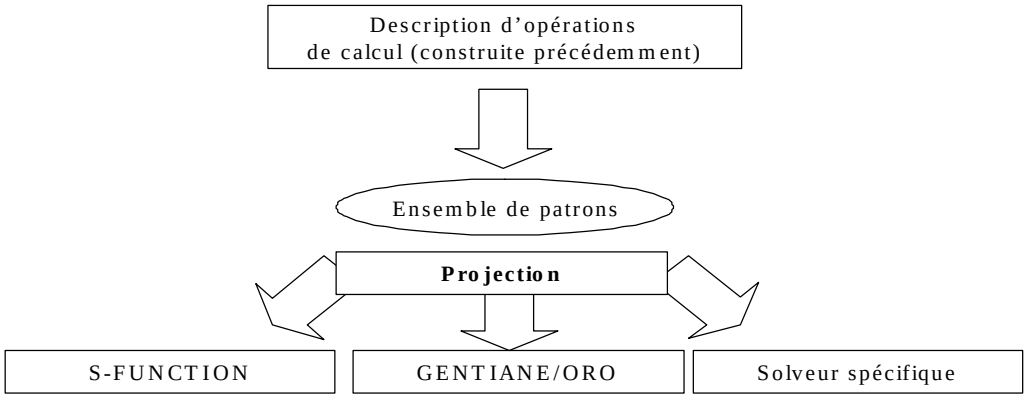


Figure 3. 29: Mécanisme proposé pour la génération de code dédié

A l'issue des opérations décrites au cours du paragraphe C, nous disposons de la description d'une (ou plus) séquence de calculs. Nous disposons par ailleurs d'une librairie de patrons qui vont nous permettre de projeter les modèles vers des environnements dédiés.

F - 5 Ecriture du code (3)

La dernière étape du processus global de contextualisation d'un modèle est la « *projection* » de code en utilisant ces patrons. Le résultat de la génération de code est un bloc de calcul (ou

de simulation, en fonction de l'objectif fixé) exploitable par tout concepteur qui n'est pas intéressé par la connaissance portée par ce bloc. En effet, soit parce que résultat d'un processus de compilation de programme, soit parce qu'écrit dans un formalisme obscur pour le non initié, l'expertise embarquée dans ces modèles n'est pas accessible, de sorte que l'entité de capitalisation à ce stade est une « ***boite noire*** » (par exemple, un COB ou une dll).
Afin de permettre l'intervention de l'expert à des fins de mise au point du modèle, nous utilisons une « **boite grise** », qui contient le code source correspondant au bloc de calcul.
Ainsi, nous facilitons l'intervention du concepteur sur cette boite, en structurant fortement le code dans la définition des patrons de génération.

G - Conclusion

Dans ce chapitre, nous avons défini trois moyens différents de capitaliser la connaissance des concepteurs :

- une capitalisation hors contexte de la connaissance, sous forme de « boules ».
- une capitalisation partiellement contextualisée, une orientation est définie pour les calculs
- une capitalisation en « boite noire », les modèles sont décrits et intégrés dans des bibliothèques dédiées d'outils de simulation ou de calcul.

Dans le même temps, nous avons présenté le travail que nous automatisons pour passer d'un mode de représentation à un autre. Comme nous l'avons exposé au cours des chapitres 2 et 3 de ce rapport, nous automatisons la plupart de tâches de traitements formels et de génération de code longues, fastidieuses et dans une large mesure source d'erreurs et donc de perte de temps lorsqu'elle sont réalisées « à la main ». Le chapitre suivant de ce rapport présente un outil informatique dont la fonction principale est de réaliser les différentes tâches automatisables présentées précédemment.

Chapitre 4

Mise en œuvre logicielle

Chapitre 4 Mise en œuvre logicielle

A - Introduction

Au regard de ces différents aspects, nous avons développé une suite d'outils logiciels permettant de remplir les fonctionnalités développées au cours des chapitres 2 et 3 :

- MAEL pour Module for Analysis of Equations and Logic, réalise les fonctions de capitalisation et de transformation des modèles
- HAGEL pour Hybrid Automaton for Generation of Equation and Logic, réalise les fonctions de génération automatique de code informatique

Dans ce chapitre nous décrivons les outils MAEL et HAGEL ainsi que les choix techniques pour leur réalisation.

Par ailleurs, nous présentons un utilitaire développé pour les besoins de manipulations symboliques : RAMA pour Rule Applicator for Mathematic Analysis. Nous présentons ici les spécifications de cet utilitaire qui est totalement décrit en annexe.

Avant de détailler ceci, nous présentons rapidement la structuration des modèles.

B - Structuration des modèles

B - 1 Des natures différentes

Nous avons mis en évidence la nécessité pour le concepteur de disposer de plusieurs représentations d'un même modèle sans devoir, pour autant, en gérer la compatibilité. Ainsi, le concepteur manipule des modèles :

- non orientés, les « boules »
- orientés, les boites blanches
- dédiés à un environnement, le code source du modèle
- dédiés et compilés pour un environnement : les blocs de calculs ou les composants
- sous la forme de composants informatique dédié

Nous fournissons au concepteur des outils informatiques qui lui permettent de transformer la représentation la plus générique qu'il manipule, les boules, en un autre des formats.

Il est néanmoins nécessaire de gérer la visualisation, et potentiellement, l'édition et l'utilisation de ces formats. En particulier, pour les codes sources des lois de commande qui

peuvent nécessiter des mises au point complexes et, parfois, le recours à l'utilisation d'outils de développement informatique, pour les activités de déboguage.
De plus, nous lui permettons de générer un document imprimable de son modèle au format pdf (Adobe Acrobat)[PDF].

Nous ne présentons, ici, que les aspects génériques des formats de capitalisation proposés, ceux-ci sont décrits en détail en annexe de ce rapport.

B - 2 Le format de capitalisation

i) les besoins

Pour la capitalisation des modèles nous définissons différents besoins :

- un support pour la capitalisation des données, c'est-à-dire leur conservation dans un but de réutilisation impose que le support de capitalisation soit, dans la mesure, du possible indépendant d'un outil logiciel et d'un système d'exploitation.
- le format de capitalisation doit reproduire la structuration des données que nous avons décrite dans le chapitre 3 : un arbre.
- La pérennité du support de capitalisation et sa réutilisabilité pour le concepteur, cela impose que la sauvegarde se fasse sous la forme d'un fichier texte.
- L'extensibilité, en effet, la pérennité d'un langage dépend de sa capacité à accepter des extensions pour répondre aux besoins non prévus à l'origine.

ii) la solution choisie

Nous avons donc choisi de sauvegarder les descriptions des modèles boules et boites en utilisant le format XML (eXtended Markup Language) [W3C]. Ce langage est simple, extensible et privilégie les structurations arborescentes des données. De plus, fondé sur une description textuelle de l'information, il est naturellement indépendant des outils et des systèmes d'exploitation. Outre ces avantages, il existe aujourd'hui de nombreux outils logiciels qui en permettent la manipulation, la transformation et la visualisation des données de ce format.

B - 2 - 2 La description des boules

Dans le chapitre 3, nous avons présenté une structuration des données pour les modèles non orientés. Cette structuration est retranscrite dans les fichiers XML décrivant les modèles non orientés.

La Figure 4. 1 illustre la description d'un modèle de résistance linéaire sous la forme d'un modèle non orienté.

```
<MODEL comment=« Modèle d un fil resistif" name= " Resistor" >
        <EQUATION value= "R =rho*L/S" />
        <VARIABLE comment= "  longueur l" unit= " me ter" name="L" visibility="intern" datatype="real" />
        <VARIABLE unit=" ohm"  name="R" visibility="extern" datatype="real" />
        <VARIABLE unit= "   m^2" name="S" visibility="intern" datatype="real" />
        <VARIABLE unit= "  ohm*m" name="rho" visibility="extern" datatype="real" />
</MODEL>
```

Figure 4. 1: Description d'un modèle de résistance non orienté (*.mo)

B - 2 - 3 La description des modèle boites

Dans le chapitre 3, nous avons défini une structuration de l'information pour les modèles orientés. Cette structuration est réalisée, également, sous la forme de fichiers XML qui reprennent l'arborescence proposée.

Nous illustrons cette implantation sur la Figure 4. 2 qui reprend le modèle de résistance présenté ci-dessus dans un contexte orienté.

```
<PROCESS comment= " description d'un conducteur" name="Resistor" >
        <EQUATION value= "R =rho*L/S" />
        <PARAMETER comment="longueur du conducteur"
                        unit= " meter" name="L" datatype="real" />
        <VARIABLE unit=" ohm" name="R" state="output"  />
        <PARAMETER unit= "  m^2" name="S" datatype="real" />
        <PARAMETER unit= " ohm*m" name="rho"  datatype="real" />
        <ZONE condition= "always" >
                <NOTTIME>
                        <RELATION expression="R =rho*L/S" order="0"
                                                resolution="explicit" >
                                <SOLVEDPARAMETER name="R" />
                        </RELATION>
                </NOTTIME>
                <TIME/>
        </ZONE>

</PROCESS>
```

Figure 4. 2: Exemple de description d'un modèle orienté : description d'une résistance (*.pro)

L'information représentée dans ces fichiers résulte de l'ensemble des opérations expliquées dans le chapitre 3. Il n'est, a priori, pas nécessaire d'en prévoir l'édition. Ce fichier est la source de l'information utilisée pour la génération des blocs de calculs ou des composants.

B - 2 - 4 Diffusion et documentation des modèles

Dans notre communauté, il est très intéressant de pouvoir diffuser et partager la connaissance. A ce titre, nous avons réalisé un format particulier de représentation des modèles qui en

permet le partage papier et électronique rapide. Il est ainsi possible de créer un fichier reprenant la vue totale du modèle sous une forme plus conviviale et plus exploitable, par exemple un fichier au format PDF, grâce à un outil libre développé par le groupe Apache : FOP [FOP].

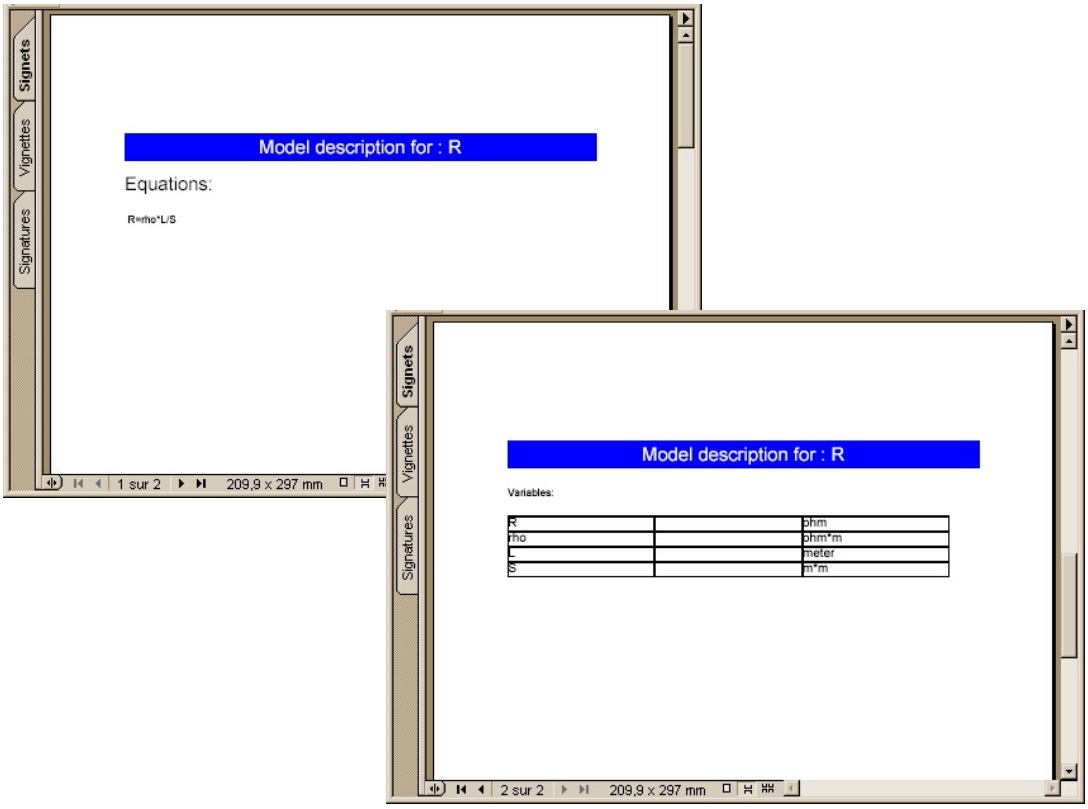


Fig. 4. 3: Diffusion des modèles sous la forme de fichiers au format PDF

Par ailleurs, nous avons mis en évidence l'intérêt de la documentation des modèles. Afin de satisfaire à cette obligation, nous offrons la possibilité de documenter les modèles par l'ajout de schémas de texte ou de toute autre information que le concepteur estime utile à la compréhension et à l'exploitation de son modèle. Ces documents sont des fichiers indépendants de la description du modèle mais qui y sont associés afin de conserver une entité de modélisation cohérente et qui contienne toute l'information que le concepteur y a ajoutée.

B - 2 - 5 Utilisation et tests des modèles

Le développement d'un modèle de calcul ou de simulation est souvent long, et la mise au point peut s'avérer fastidieuse si le concepteur est astreint à répéter un grand nombre de fois les opérations d'orientation et de génération du modèle.

Nous proposons donc qu'il lui soit permis de réaliser des calculs dans l'environnement de modélisation. Pour cela, un composant de calcul est générable et exploitable dans MAEL, le COB. Ce même composant informatique de calcul est utilisé pour le dimensionnement dans

des outils tels que CDI-Optimizer (développé par David Magot- doctorant dans les équipes CDI/EP du LEG) ou Pro@Design. Aujourd'hui, les tests des blocs de simulation ne sont pas disponibles dans notre environnement.

Ainsi l'environnement doit gérer 3 types d'entités différentes :

- les boites blanches pour la capitalisation de la connaissance, ces représentations sont des fichiers compressés qui contiennent la représentation mathématique des modèles ainsi que l'ensemble des documentation ajoutées par le concepteur :
    - les modèles non orientés : caractérisés par l'extension « ***.mo*** »
    - les modèles orientés, les boites : caractérisés par l'extension « .**pro** »
- les boites noires, utilisées pour les évaluations numériques ou pour la diffusion limitée du modèle. Celles-ci sont générées automatiquement à partir des représentations précédentes :
    - des composants de calculs, les COB
    - des fichiers au format pdf, pour la visualisation et l'impression des modèles
- les boites grises, utilisées pour la mise au point des modèles et les activités déboguage :
    - des fichiers « **.c** » et les dll (Dynamic Link Library) associées générés pour MATLAB/SIMULINK
    - des fichiers « **.cpp** » générés pour GENTIANE-ORO

C - Principe d'utilisation de l'outil

Au dessus de ces modèles, nous avons identifiés 2 niveaux d'actions :

- le premier est le niveau d'action des concepteurs qui décrivent des modèles de structures pour une utilisation en simulation ou en dimensionnement et font des choix d'outils, de structuration des données (découpage par blocs, …). Ce niveau d'action est celui qui correspond à la manipulation du modèle et sa transformation depuis la représentation hors contexte jusqu'au code de calcul. Il s'agit de l'utilisation de l'outil MAEL.
- le second est le niveau d'intervention des experts en traitements formels de modèles. Il s'agit d'un niveau de maintenance et d'extension des capacités du logiciel, par exemple l'ajout de squelettes de génération, qui se fonde sur l'utilisation de HAGEL.

Sur ces deux niveaux, il nous parait important d'offrir la plus grande souplesse possible.

Le concepteur intervient pour les tâches de (cf. Figure 4. 4):

- description de sa connaissance sous une forme mathématique (1)
- choix de l'orientation du modèle produit précédemment (2)
- le choix d'un environnement cible pour utiliser le modèle et faire sa conception (3)

De plus, un expert de l'outil intervient sur l'outil informatique pour augmenter, modifier ses capacités, par exemple de génération de code.

La Figure 4. 4 illustre les rôles des différents intervenants :

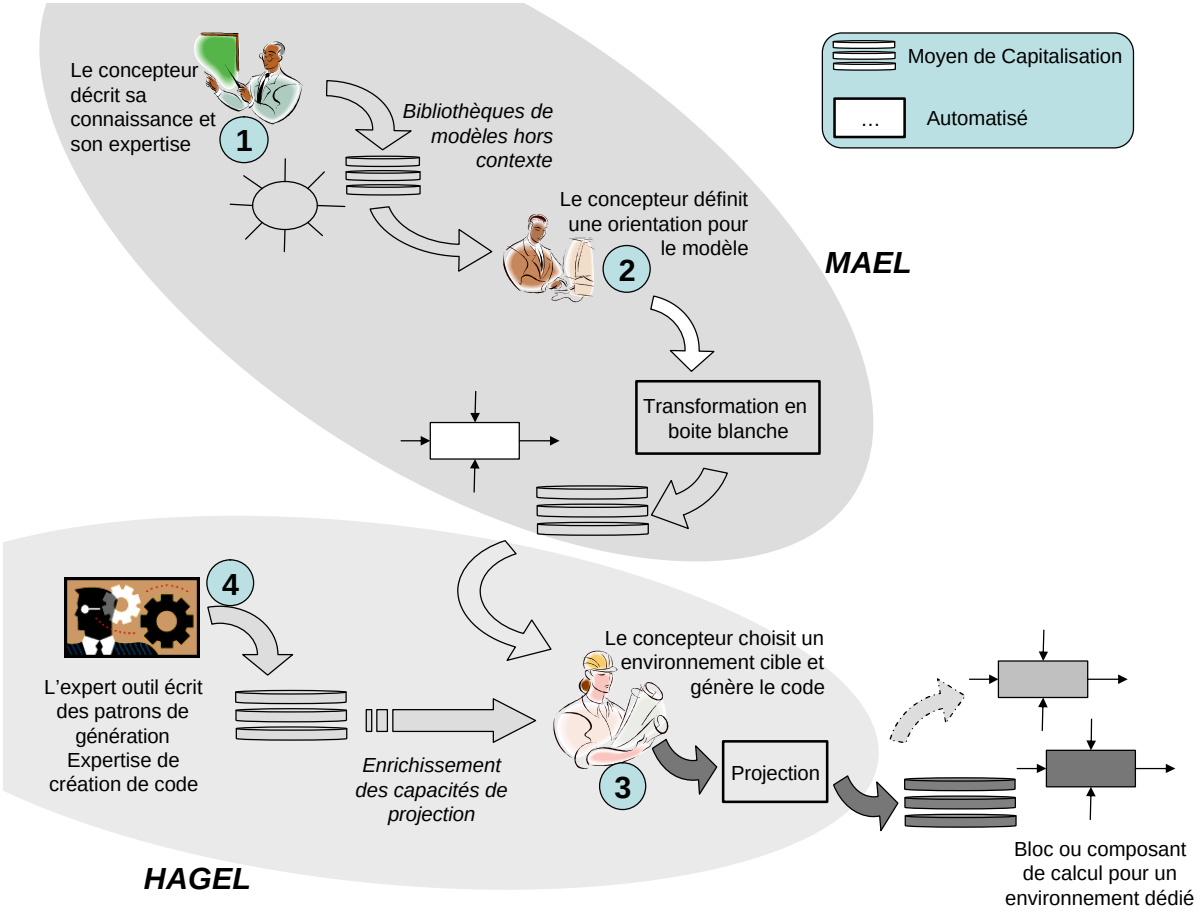


Figure 4. 4: Principe général de fonctionnement de MAEL et de HAGEL

D - Mise en œuvre

D - 1 Spécifications de l'environnement

L'environnement de modélisation développé doit, pour l'utilisateur :

- permettre l'édition des modèles non orientés

- supporter la transformation des boules en boites
- réaliser la génération de code de calcul
- permettre la validation des modèles développés
- offrir l'ensemble des outils nécessaires à la modification des modèles réalisés

En ce qui concerne la maintenance de l'outil, il y a des besoins :

- de modularité, chaque élément de l'architecture doit être indépendant des autres et pouvoir être modifié sans conséquence grave pour le reste de l'environnement
- d'extensibilité, par exemple pour l'ajout de nouveaux générateurs de code
- de documentation du code et des interfaces des modules

Enfin, l'environnement doit offrir une interface agréable et conviviale au concepteur.

Nous illustrons l'ensemble de ces spécifications sur la Figure 4. 5.

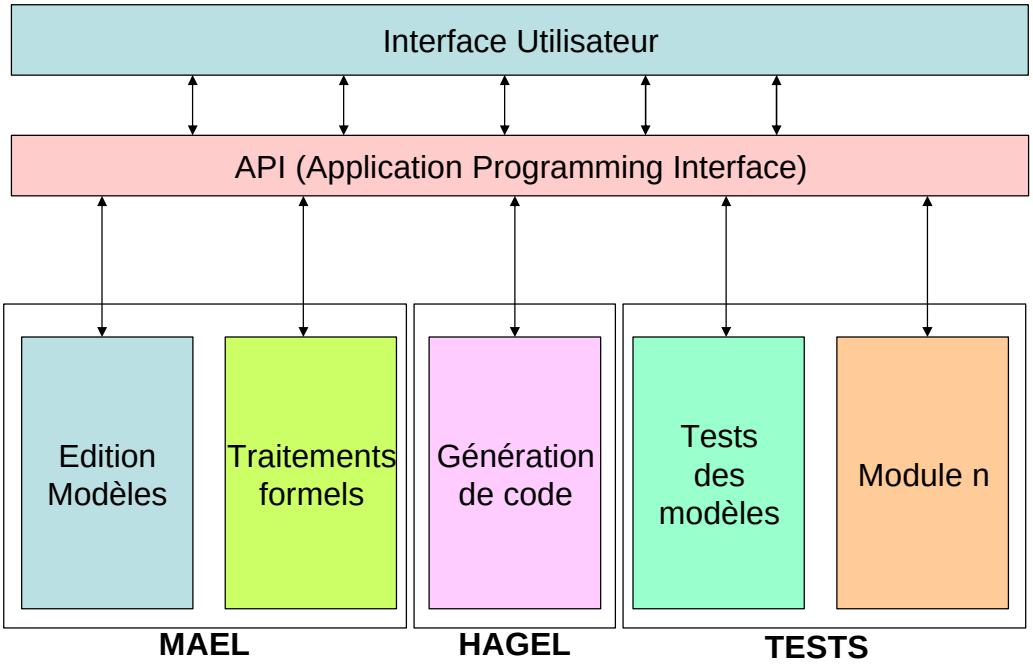


Figure 4. 5: Spécifications de l'outil

Puisque chaque module identifié peut lui-même être décomposé en sous-modules, chacun d'entre eux doit lui-même pouvoir être changé sans nuire au fonctionnement de l'ensemble. Ce changement doit se faire simplement et avec un investissement minimal de la part du développeur.

D - 2 Solution technique implantée

La Figure 4. 6 représente l'architecture générale mise en œuvre pour l'environnement.

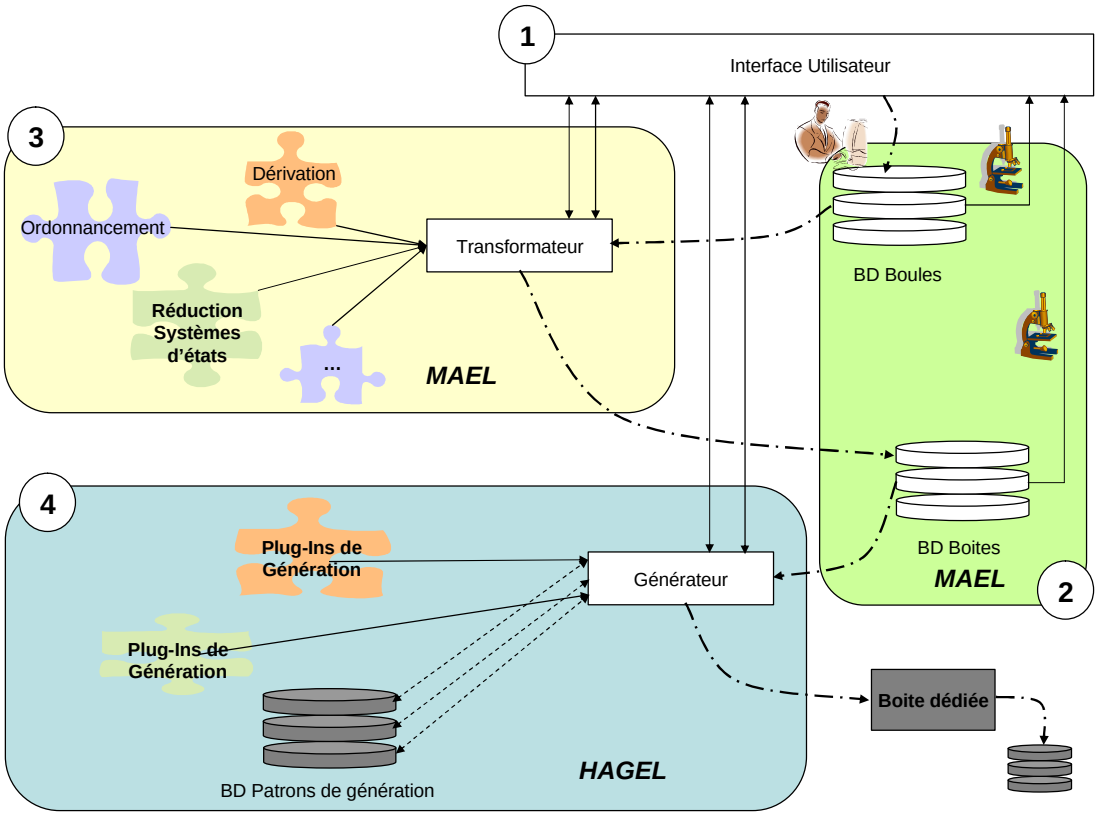


Figure 4. 6: Architecture de l'environnement

L'environnement dispose d'une interface utilisateur commune **(1)** qui offre des services pour :

- piloter les transformations effectuées sur les modèles par le biais de MAEL
- piloter la génération de code réalisée par HAGEL
- permettre l'édition des modèles hors contexte réalisée par l'environnement
- la visualisation des modèles hors contexte et des modèles orientés réalisée par le l'environnement.

Les deux modules d'édition et de visualisation ne portent pas de nom particulier, il ne s'agit que de facilités graphiques offertes à l'utilisateur.

Par ailleurs, MAEL gère les bases de données **(2)** des modèles orientés et non orientés (boites et boules).

MAEL a été développé pour aider le concepteur à mettre en œuvre la méthodologie de transformation des modèles que nous présentons dans le chapitre 3 de ce rapport. Il existe donc un module de MAEL dont la fonction principale est la transformation des modèles **(3)**. Ce module est conçu de telle façon qu'il est possible de changer simplement les étapes du processus de transformation.

Enfin, l'environnement offre un support à la création de code pour le calcul ou la simulation temporelle par le biais de l'outil HAGEL. Ainsi, un module de l'architecture de

l'environnement est consacré à la génération de code **(4)**. HAGEL gère une base de données des patrons de génération, et offre le moyen, par l'intermédiaire d'un mécanisme de « plug-ins », d'augmenter les capacités de générations de l'environnement.

D - 2 - 1 L'interface homme machine (IHM)

1 - a) L'architecture

Les besoins de convivialité de l'outil nécessitent le développement d'une IHM poussée, ouverte. Il existe actuellement sous les termes de la licence SPL (Sun Public Licence), un noyau d'interface graphique qui offre la quasi-totalité des services dont nous avons besoin.
Ce noyau d'interface graphique (Platform Netbeans) [NETB] est actuellement utilisé dans de nombreux environnement de développement logiciel, mais aussi pour des fonctions aussi diverses que l'édition musicale (Projet XEMO) ou de données géologiques. La plateforme offre un certain nombre de services de base pour la gestion de l'interfaçage graphique des applications logicielles. Le principe d'utilisation et de réutilisation de cette plate forme est illustré sur la Figure 4. 7.

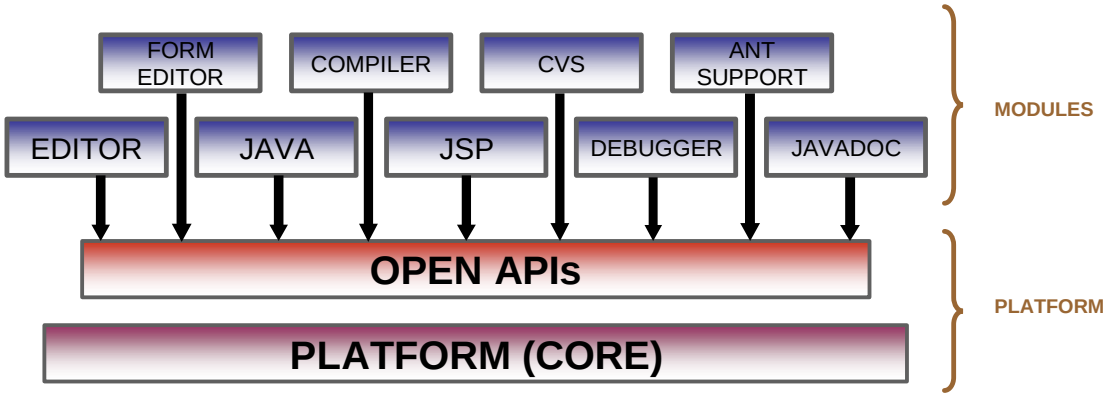


Figure 4. 7: La plate forme Netbeans (Image issue du site internet : http://www.netbeans.org)

La réutilisation de ce noyau n'a nécessité de notre part que la description d'un module chargé au lancement de l'application, et nous a ainsi offert un gain de temps considérable dans le développement de l'ensemble de l'outil.
La Figure 4. 8 est une présentation de l'ensemble de l'IHM de l'outil.

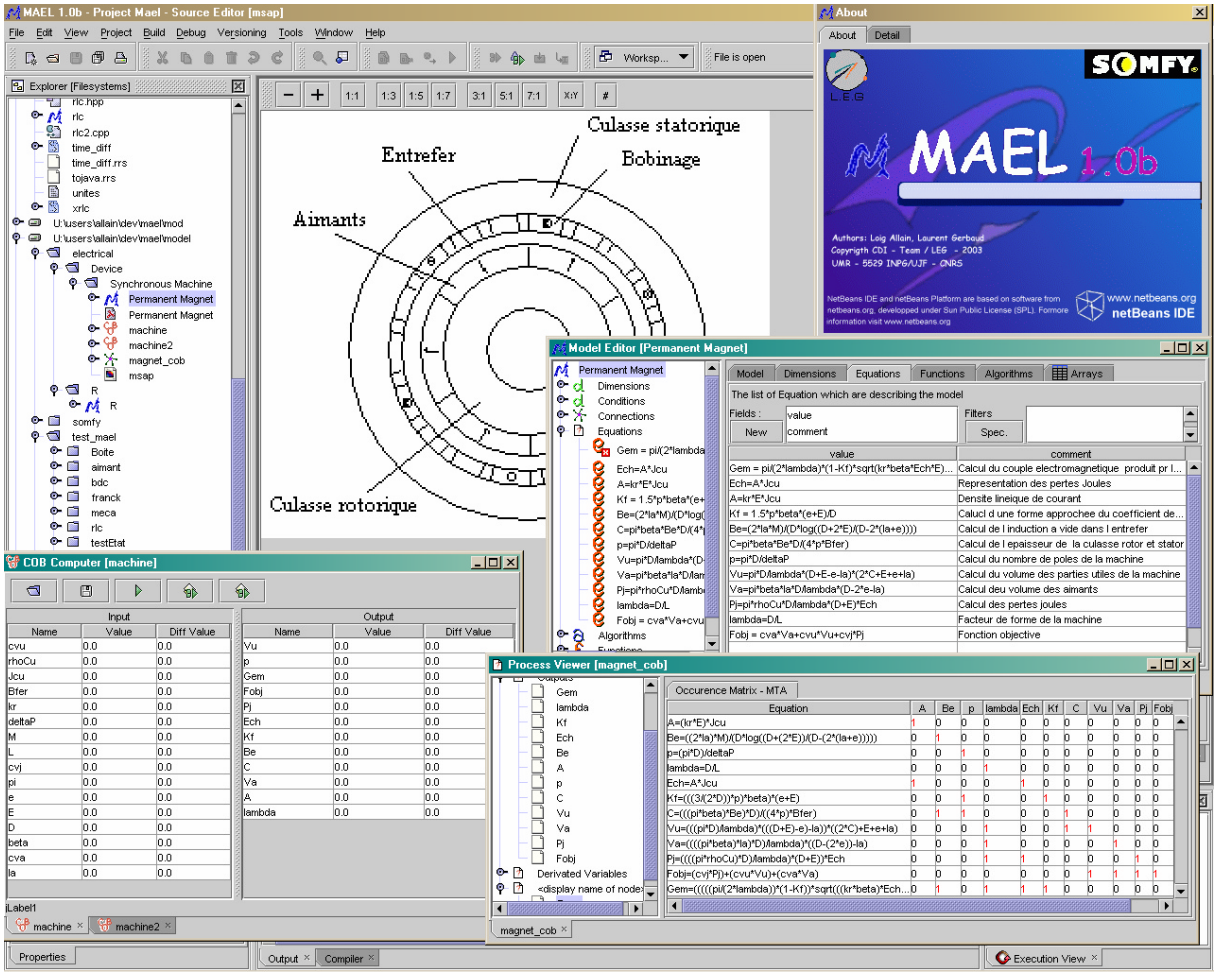


Figure 4. 8: IHM de MAEL

1 - b) La visualisation des modèles

L'IHM de l'environnement permet une visualisation et une édition simple de l'ensemble du modèle par le biais d'un module d'édition illustré sur la Figure 4. 9.

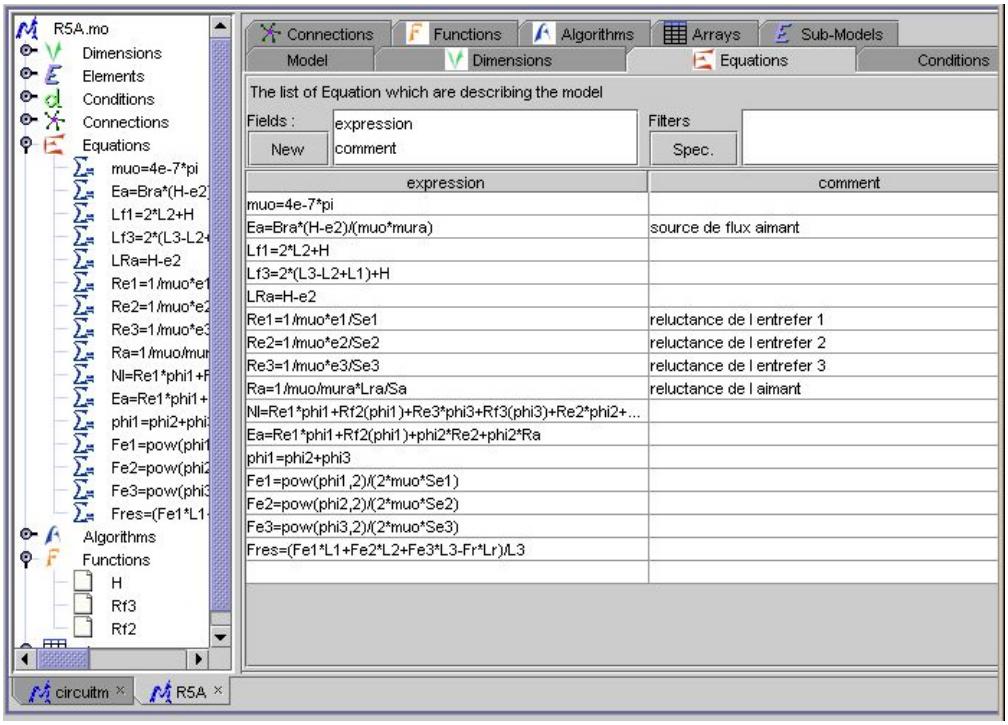


Figure 4. 9: Visualisation et édition du modèle

Le concepteur peut éditer son modèle hors contexte, c'est-à-dire qu'il ne se concentre que sur la connaissance qu'il décrit. Les modèles boules sont structurés en un arbre, que l'IHM permet de visualiser (Figure 4. 10) ; ce qui facilite également la navigation au sein des différents éléments du modèle.

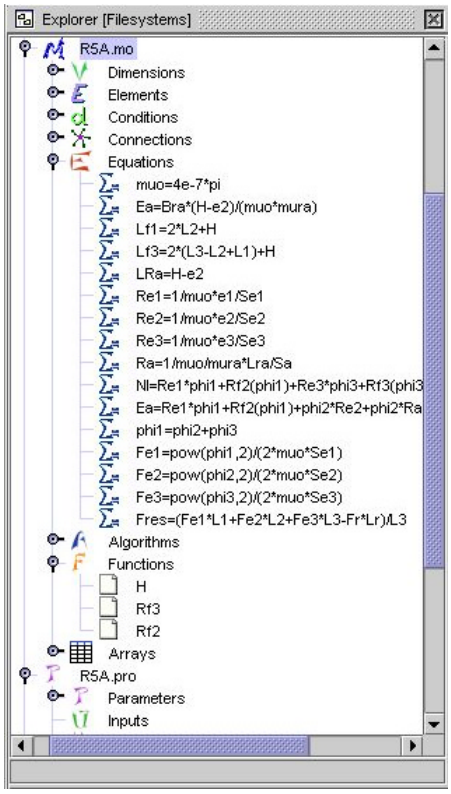


Figure 4. 10: Arborescence des modèles

1 - c) Test des composants de calculs

Afin de permettre la validation des modèles développés avant la phase de génération pour un environnement spécifique, un module permet de réaliser des évaluations sur ces modèles.

Ce module n'utilise que les composants informatiques de calcul COB générés dans l'environnement. Il permet de réaliser des calculs simples ou de tracer des courbes des grandeurs de sorties en fonction des paramètres d'entrée (Figure 4. 11).

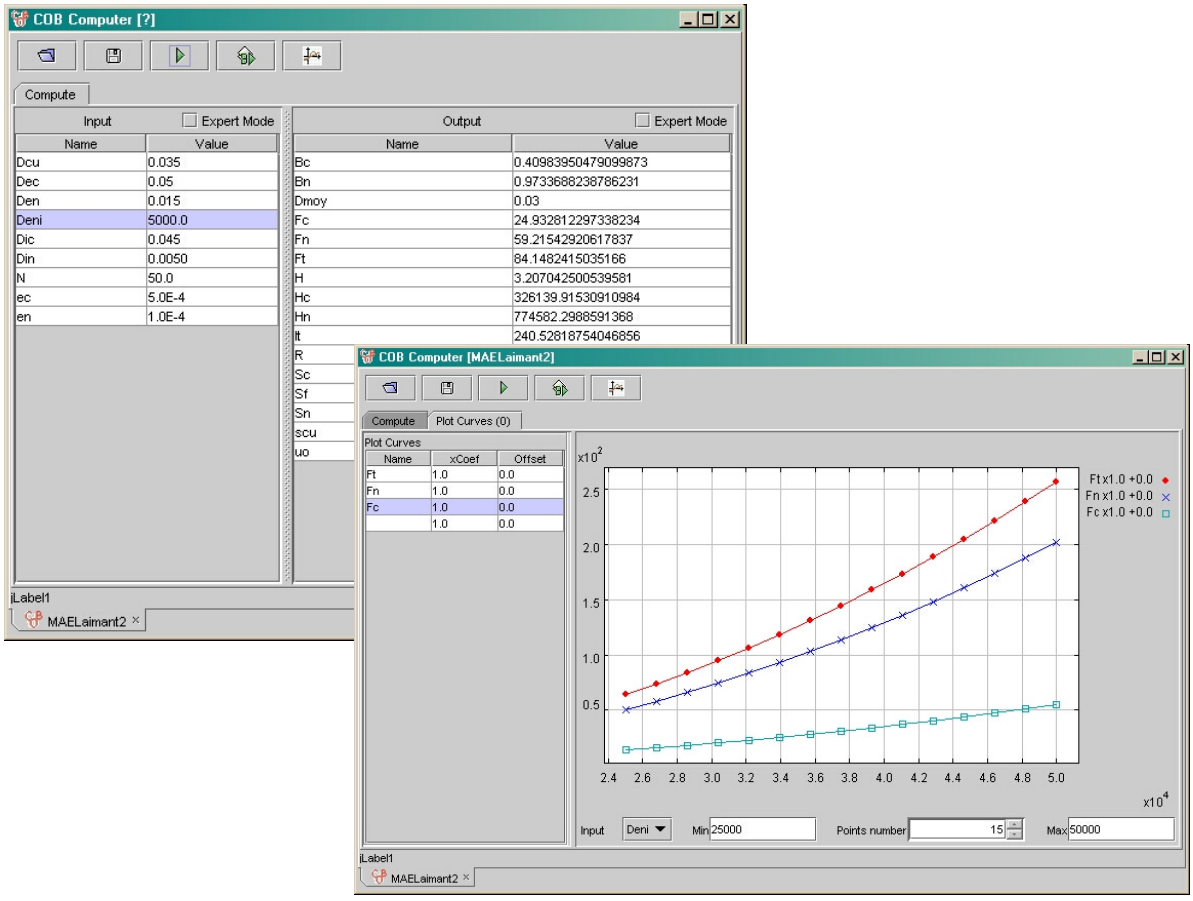


Figure 4. 11: Module de test des modèles

1 - d) Transformation des modèles (MAEL)

MAEL est architecturé autour d'un module essentiel qui réalise le processus de transformation des modèles : l'orientation et la création d'un processus de calcul.

Nous montrons le fonctionnement de ce module sur la Figure 4. 12.

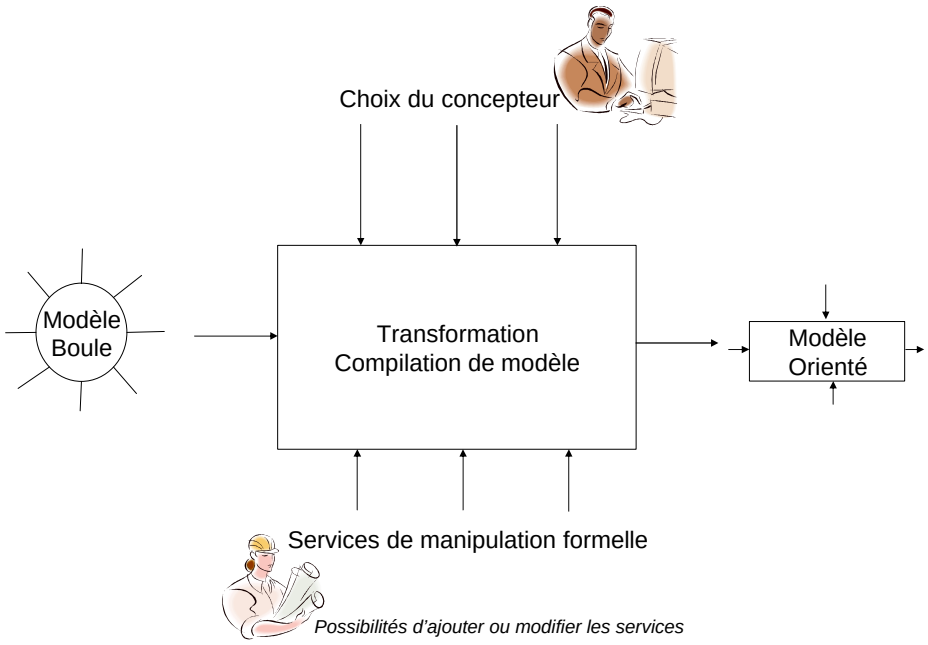


Figure 4. 12: Principe de fonctionnement du module de transformation des modèles

Le processus de transformation permet de produire la description d'une boite de calcul à partir de la description d'un modèle. Pour réaliser cette transformation, le processus s'appuie sur des choix du concepteur. Celui-ci choisit d'attribuer des rôles spécifiques à chaque grandeur : paramètre, entrée, sortie. Pour le guider et l'assister dans ses choix, un assistant lui offre des listes de choix, la Figure 4. 13 présente les différentes étapes de cet assistant.

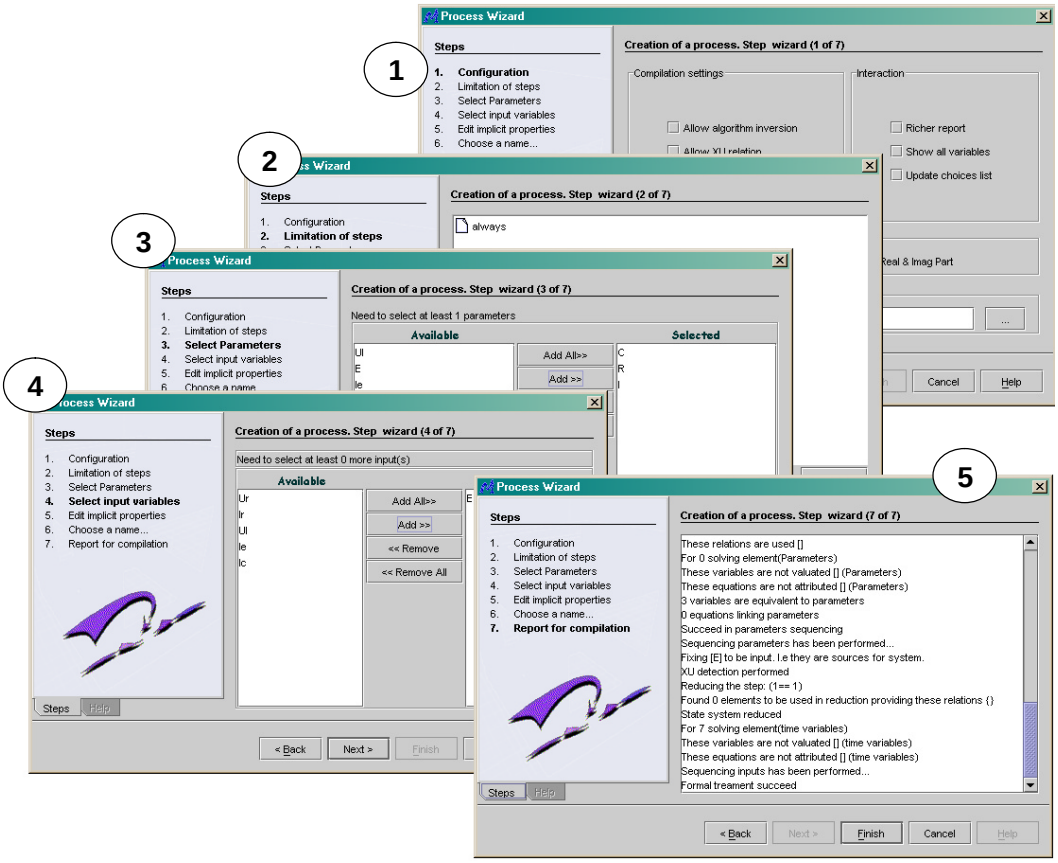


Figure 4. 13: Assistant de transformation / compilation

1 - e) Génération de code (HAGEL)

i) Spécifications de la génération

Ce module permet la génération d'un code de calcul dédié à un environnement de calcul. Afin de faciliter cette activité, nous proposons de l'automatiser. Cette automatisation s'appuie sur la capacité d'un expert en création de code à créer des patrons de génération.

Par ailleurs, il est nécessaire de pouvoir gérer une multitude de fichiers dans le cas de la génération de gros problèmes de simulation. Dans ce cas, il peut s'avérer nécessaire de fragmenter les fichiers de code ou certaines fonctions. Ainsi, des opérations de gestion des tailles de texte doivent être disponibles.

De plus, il est nécessaire de pouvoir développer à peu de frais :

- de nouveaux patrons de génération : dans un langage aussi proche que possible de la cible
- de pouvoir ajouter de nouvelles fonctionnalités pour pouvoir répondre à de nouveaux problèmes de génération

Enfin, il doit être possible de récupérer des informations concernant l'activité de génération, report d'erreur ou de succès, quantité et identités des fichiers générés.
Nous devons avoir une récursivité de la programmation afin de pouvoir développer des patrons de génération complexes.

ii) Les patrons de génération

Le développement des patrons de génération est une activité pointue et souvent longue. Nous offrons donc un moyen de créer des patrons de génération de la manière la plus simple possible. Nous avons développé un langage élémentaire qui permet de :

- décrire le code tel qu'il doit être créé
- définir les zones de modifications par le biais d'un ensemble de balises.

iii) Structure générale du module

Nous proposons un outil de génération de code nommé HAGEL (la description en est donné en annexe A, qui utilise ces patrons de génération. La Figure 4. 14 illustre le principe de fonctionnement de HAGEL.

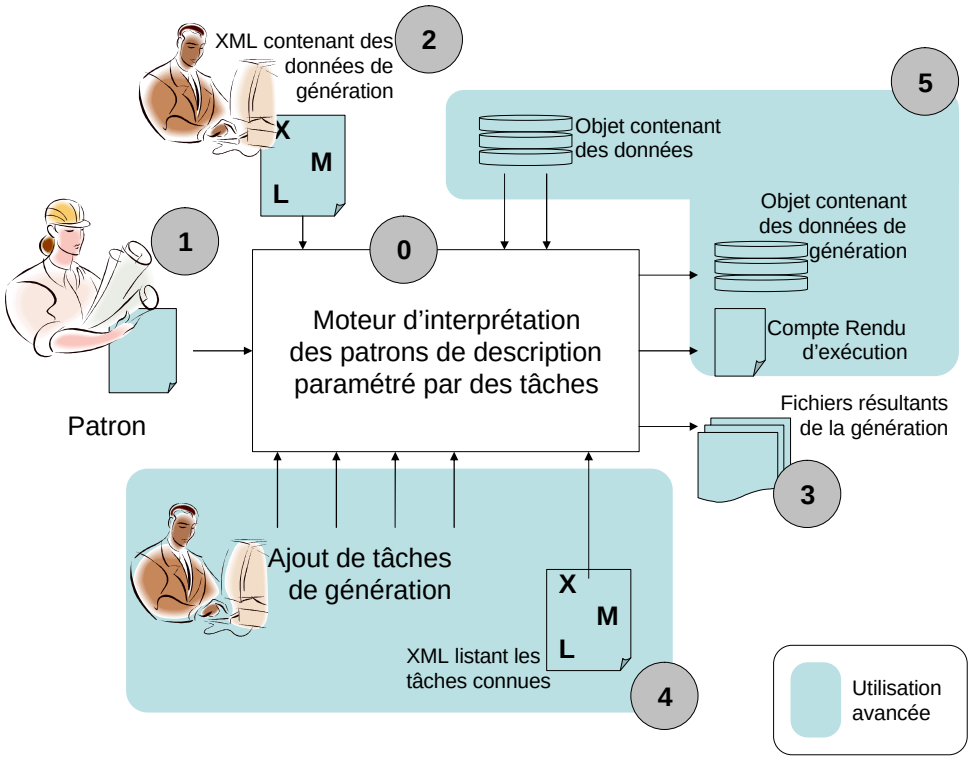


Figure 4. 14: Principe de fonctionnement de HAGEL

iv) Utilisation de HAGEL

Nous distinguons trois modes d'utilisation :

- un premier mode, classique, utilise le moteur d'interprétation **(0)** pour construire une liste d'actions élémentaires à réaliser. Cette liste est construite par interprétation d'un fichier de texte, le patron, **(1)** qui contient la description des fichiers de destinations. La génération de code est alors effectuée en récupérant les données décrites dans un ou plusieurs fichiers au format XML. Le résultat est une liste de fichiers **(3)** de code générés.
- Une deuxième possibilité d'utilisation se base sur l'ajout de tâches élémentaires **(4)** de génération qui sont utilisées dans la description du patron pour décrire des activités de génération plus complexes. Ces tâches sont des éléments de programmes ajoutés par le développeur et dont la nature est spécifiée dans un fichier XML. La suite de l'utilisation correspond au premier mode d'utilisation.
- la troisième utilisation possible du générateur repose sur la possibilité de manipuler des objets informatiques en cours de génération, soit en récupérant des données, soit en y ajoutant des informations de génération **(5)**, cette utilisation fait un usage intensif de tâches supplémentaires développées à cette occasion.

v) Des plug-ins de génération

Les besoins de génération peuvent évoluer très rapidement. En effet, l'ajout de fonctions spécifiques telles que les fonctions de Bessel, les calculs d'intégrales ou les évaluations de minima ou de maxima peuvent poser des problèmes de générations. Nous proposons donc une architecture à base de plug-ins de génération qui permettent d'ajouter à moindre frais de nouvelles fonctionnalités de génération.

Littéralement, les « plug-ins » sont des éléments à « brancher » sur un logiciel. Dans la pratique, les plug-ins sont des éléments de logiciels qui augmentent des fonctionnalités ou se substituent à l'existant et qui ne fonctionnent que dans le contexte de l'environnement dont ils sont des plug-ins.

Les plug-ins présentent un intérêt pour :

- l'ajout de la prise en compte de nouvelles fonctions mathématiques dans la description des modèles
- la possibilité d'augmenter les librairies de méthode de calculs numériques en génération, par exemple pour la résolution de problèmes linéaires et non linéaires.

Le principe des plug-ins de génération que nous avons implanté dans le module de génération répond au besoin de souplesse et d'ouverture de l'ensemble du logiciel.

vi) Projecteurs réalisés

Aujourd'hui des projecteurs ont été réalisé en utilisant ces techniques pour :

- générer des COB, composant de calcul utilisé pour le dimensionnement
- générer des S-Function pour MATLAB/SIMULINK, ce sont des blocs de calculs utilisables pour la simulation de structure
- générer une représentation VHDL-AMS des boites, ce qui permet d'utiliser les modèles de simulation dans des outils tels que Simplorer (travaux actuels de DESS de Hichem Chetouani)

D - 2 - 2 Un utilitaire pour les manipulations symboliques : RAMA

Nous avons également prévu que l'expertise en traitement formel des modèles soit amenée à se développer, et, dans cette perspective, il est possible de modifier les services de manipulation formelle pour les remplacer par d'autres. Cette architecture logicielle offre ainsi une grande souplesse pour la maintenance des activités essentielles de traitement formel. Dans ce contexte nous proposons un utilitaire pour les manipulations formelles des expressions mathématiques.

2 - a) Situation respective de RAMA par rapport aux modules de l'environnement

Les activités de transformation des modèles boules et de génération de code sont complexes mais, il est possible de regrouper des fonctionnalités similaire pour réaliser des outils permettant de faciliter les traitements.

Dans ce contexte, nous avons développé un outil informatique en collaboration avec Vincent Fischer, doctorant de l'équipe CDI du LEG qui travaille sur la prise en compte des équations différentielles pour le dimensionnement : RAMA, un moteur d'application de règles simples de calcul formel

Nous illustrons sur la Figure 4. 15 le positionnement de cet outil par rapport à MAEL et HAGEL.

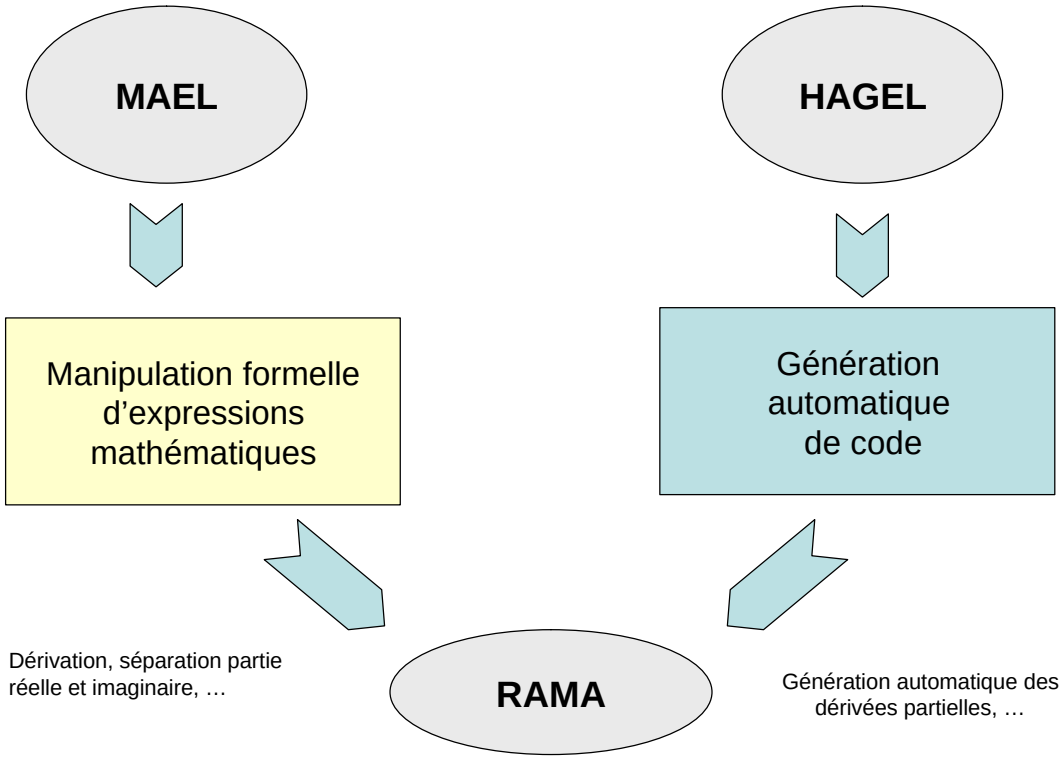


Figure 4. 15: MAEL et les outils RAMA et HAGEL

2 - b) RAMA : Rule Applicator for Mathematic Analysis

i) Spécifications

MAEL et HAGEL doivent être des outils indépendants. La conséquence de cette obligation est le souhait d'une indépendance vis-à-vis de logiciels de calcul formel tels que MAPLE ou MATHEMATICA pour les manipulations telles que la dérivation, les substitutions ou encore les séparations des parties réelles et imaginaires des nombres complexes. Cette indépendance est d'autant plus souhaitables que les opérations à réaliser sont simples et ne nécessitent pas un environnement de calcul formel lourd.

Nous avons donc spécifié les fonctionnalités d'un outil indépendant et léger de manipulation formelle sur les expressions mathématiques :

- l'indépendance par rapport à des outils de calcul formel externe
- l'extensibilité, nous devons pouvoir possible définir de nombreuses opérations mathématiques sans les connaître a priori, c'est-à-dire que nous n'avons pas connaissance de toutes les opérations formelles que nous serons amenés à réaliser
- souplesse, l'ajout de fonctionnalité se fait simplement et sans avoir besoin de modifier le code informatique

ii) Structure

Pour répondre à ce cahier des charges nous avons implanté RAMA, dont le fonctionnement est illustré sur la Figure 4. 16.

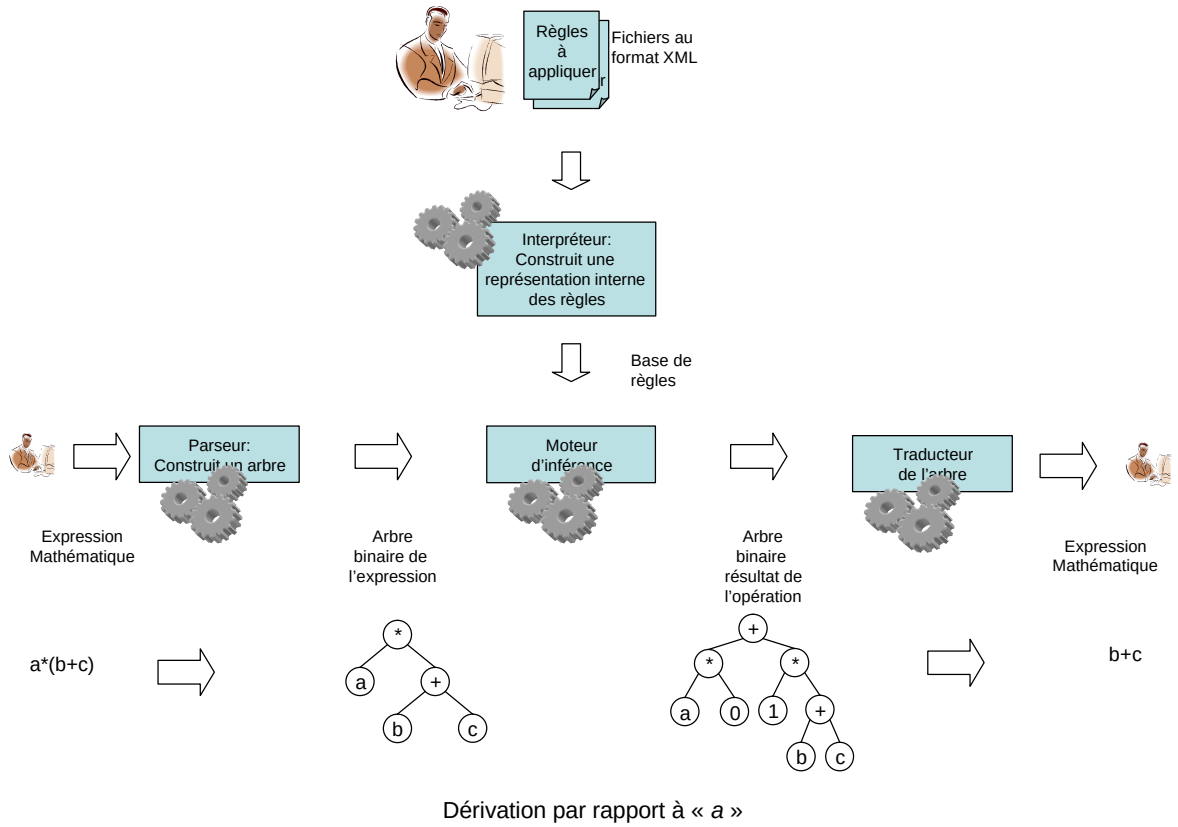


Figure 4. 16: Principe de fonctionnement de RAMA : Exemple de dérivation d'une expression

Chaque élément de RAMA est pilotable par le biais d'API, ce qui en fait un outil indépendant. L'utilisateur définit des règles de transformation de l'expression d'entrée en fonction de l'opération mathématique qu'il désire réaliser (dérivation, séparation complexe, …) dans un fichier au format XML. Ce fichier est lu et interprété par RAMA qui construit alors une représentation interne de ces règles. Puis l'utilisateur spécifie l'expression mathématique à traiter, elle est analysée pour construire un arbre binaire qui est utilisé par RAMA pour l'application des règles. L'application des règles de traitement permet de construire un second arbre qui est une image du résultat de l'opération mathématique. Cet arbre peut ensuite être retraité pour une nouvelle opération ou bien retranscrit sous la forme d'une chaine de caractères.

Nous détaillons en annexe E le fonctionnement de RAMA ainsi que son implantation et son utilisation.

iii) Utilisation

Aujourd'hui, MAEL utilise RAMA pour :

- la séparation des parties réelles et imaginaires des expressions complexes
- la dérivation temporelle des expressions

De même, HAGEL l'utilise pour

- la génération du code de calcul des dérivées partielles
- la génération automatique de code dédié par la reformulation des expressions mathématiques

***Remarque** : nous n'avons pas implanté de fonctionnalités de résolution formelle des équations. Ceci est actuellement réalisé par un outil développé par [DOL].*

E - Conclusion

Les travaux de formalisation du processus de modélisation suivi par le concepteur ont mené au développement et la mise au point d'un logiciel supportant cette méthode de travail. Par ailleurs, les besoins de capitalisation, mis en exergue dans le chapitre 3 de ce rapport, trouvent leurs solutions dans ce même logiciel qui supportent la gestion des bases de données contenant les différentes représentations de la connaissance que nous avons mises en évidence.

Chapitre 5

Exemples d’utilisation

Chapitre 5 Exemples d'utilisation

A - Introduction

La démarche que nous proposons a été appliquée sur des modèles de natures différentes :

- micro-systèmes (modèle de 300 équations)
- modélisation machine asynchrone en calcul et en simulation
- modélisation machine à courant continu pour le calcul
- électro-aimant
- …

Nous présentons dans ce chapitre deux exemples de mise en œuvre de la méthodologie de travail présentée. Nous appuyons notre propos par l'utilisation de l'outil logiciel développé. Nous présentons un exemple d'utilisation de l'outil pour le dimensionnement d'un électro-aimant rudimentaire. Puis nous présentons la mise en œuvre de l'ensemble de la méthodologie pour la création d'un bloc de simulation utilisable dans l'environnement MATLAB/SIMULINK.

B - Dimensionnement d'un électro-aimant

B - 1 Présentation de l'application

Nous nous intéressons ici au dimensionnement d'un électro-aimant dont la géométrie est donnée sur la Figure 5. 1.

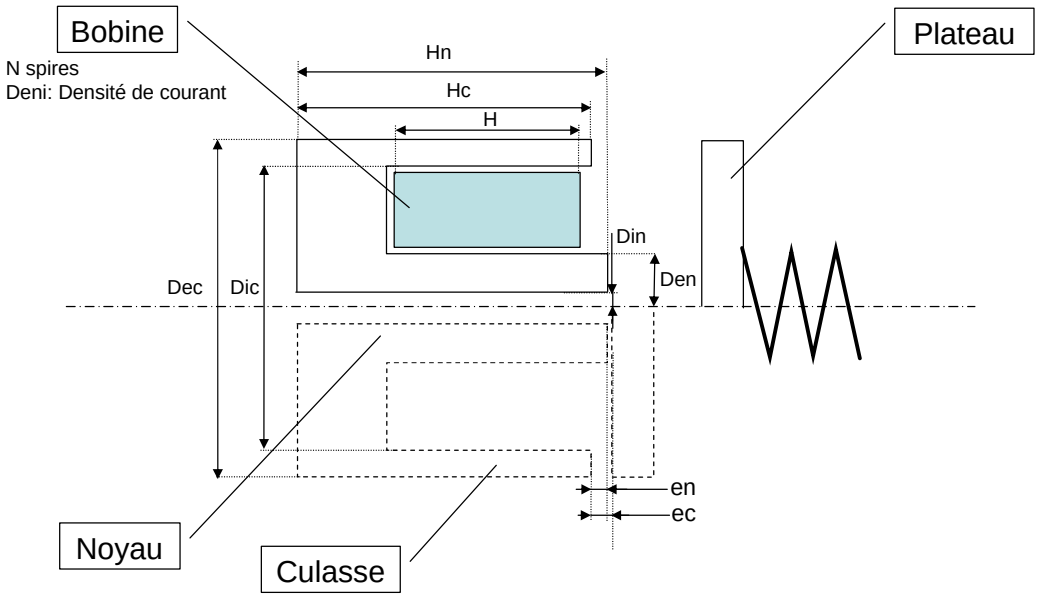


Figure 5. 1: Géométrie de l'électro-aimant

Cet électro-aimant est modélisé par un réseau de réluctance simple, nous ne prenons pas en compte la saturation des matériaux.

L'objectif de travail est de déterminer une valeur minimum du courant à injecter dans la bobine pour exercer une force de 3 newtons sur le plateau. Cette force doit être maintenue en tolérant un écart de fabrication sur les entrefers noyau et culasse. L'ensemble de l'électro-aimant doit être contenu dans un volume contraint :

- diamètre externe inférieur à D_{max}
- hauteur totale inférieure à H_{max}

Nous définissons donc le cahier des charges de l'application en ces termes :

- maximiser la force F sur le plateau
- contraindre les entrefers ***en*** et ***ec*** à être différents

B - 2 Présentation de la démarche suivie

Pour résoudre le problème qui se pose, ainsi, nous devons :

- décrire le modèle du dispositif
- documenter le modèle décrit
- choisir une orientation pour la boite de calcul
- générer un code de calcul utilisable dans l'optimisation
- vérifier la validité du modèle
- utiliser le composant informatique généré pour le dimensionnement contraint de l'actionneur

C - Démonstration de la démarche

C - 1 - 1 Saisie du modèle de l'électro-aimant

La première étape de ce travail est la description du modèle analytique du dispositif. Pour cela, nous utilisons MAEL pour éditer le modèle de l'électro-aimant.

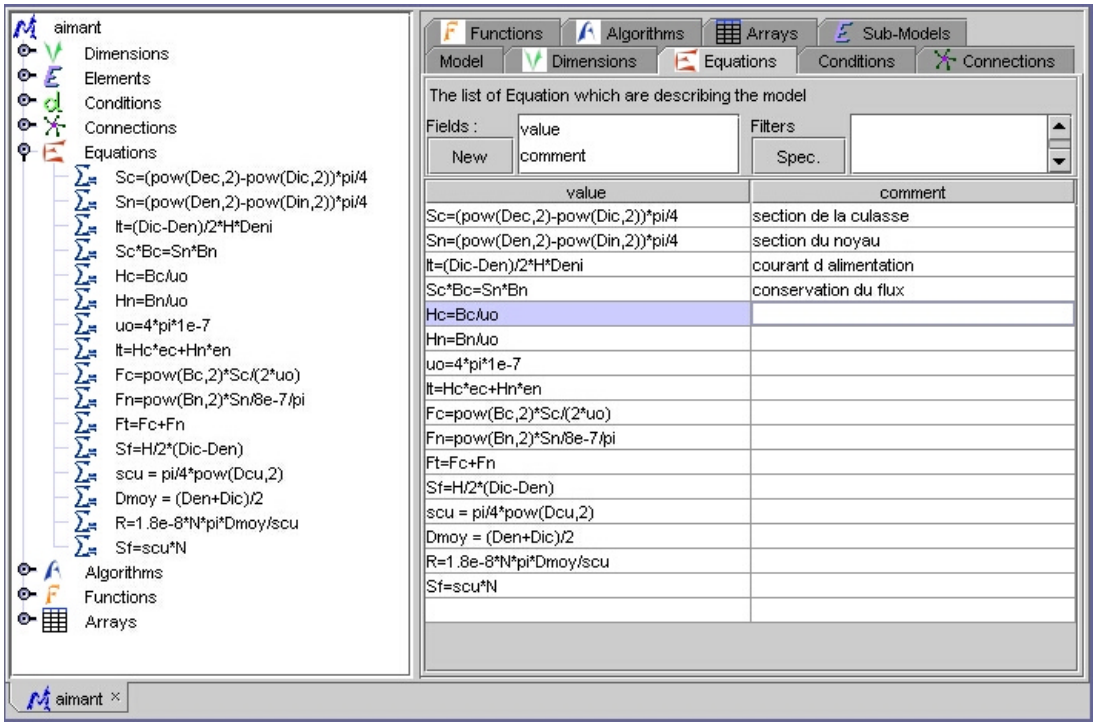


Figure 5. 2: Edition du modèle de l'électro-aimant dans MAEL

Nous pouvons retrouver à gauche de la zone d’édition, la représentation arborescente des modèles que nous avons proposée dans le chapitre 3.

La documentation du modèle est faite simultanément en ce qui concerne les équations. Pour la documentation et la spécification des variables, la liste des variables est automatique construite par analyse des relations. Puis, elle est proposée à l’utilisateur sous la forme d’un tableau qu’il complète (Figure 5. 3).

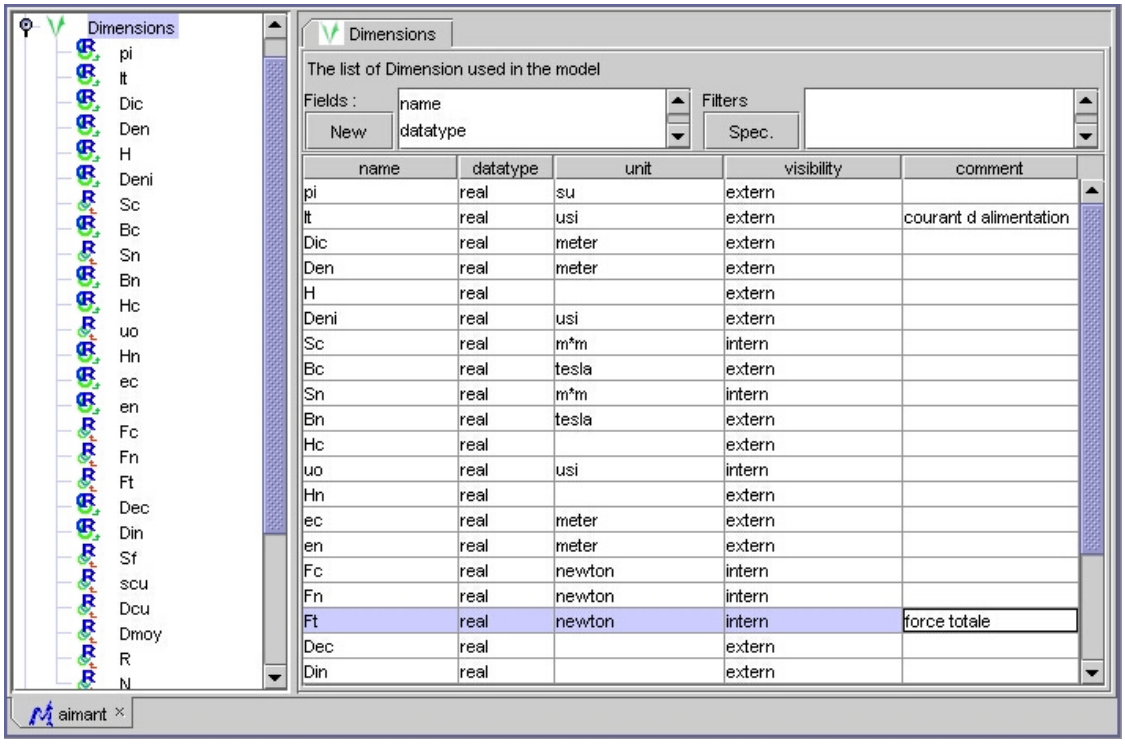


Figure 5. 3: Liste des grandeurs du modèle (boule)

Les tableaux de l'IHM de MAEL permettent de réaliser des opérations sur un jeu d'éléments de la modélisation de la même famille (VARIABLES, EQUATIONS, …). Il est également possible de faire une édition « élément par élément » en sélectionnant les nœuds de l'arbre du modèle (Figure 5. 4). La zone de droite de l'éditeur présente alors un ensemble de champ pour l'édition des propriétés de l'élément.

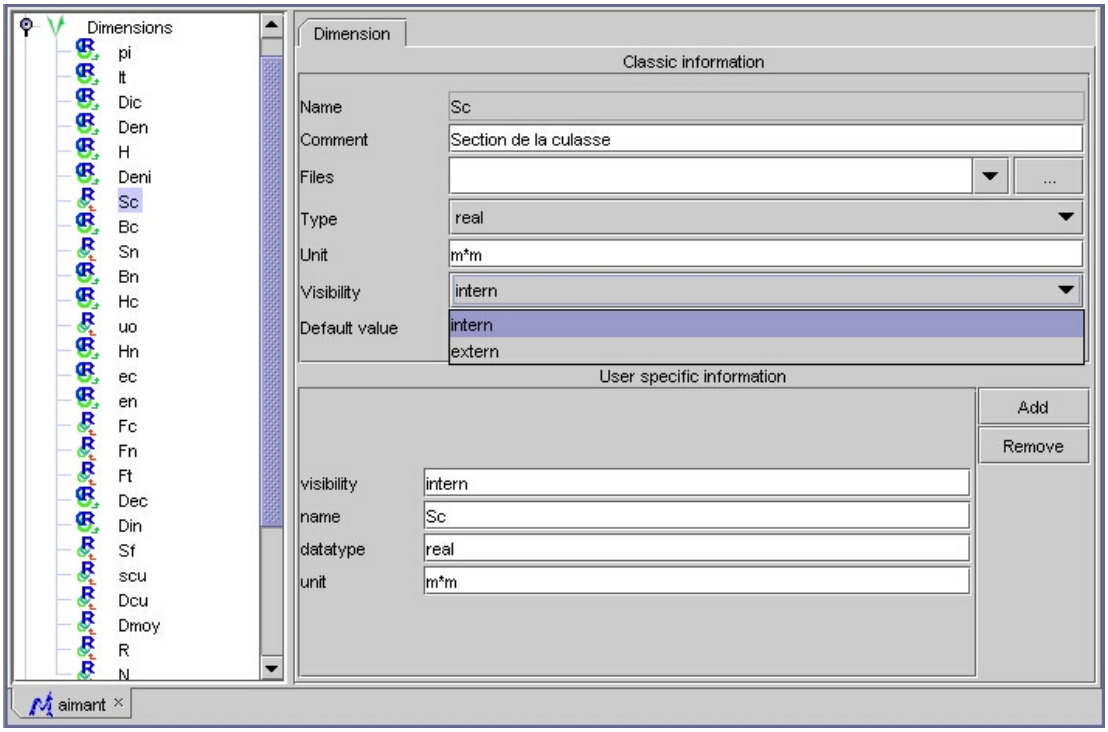


Figure 5. 4: Edition des propriétés avancées d'un nœud

Cette zone se divise elle-même en deux parties :

- la partie supérieure permet l'édition des propriétés génériques des éléments, celles que nous avons définies par défaut dans la structuration des données
- la partie inférieure qui offre à l'utilisateur la possibilité de définir des propriétés non prévues initialement mais qu'il juge nécessaire de fournir (auteur, versionnement du modèle, date de création, etc...)

A ce stade, nous avons décrit une « boule » qui contient les lois physiques régissant le fonctionnement de l'électro-aimant. Nous choisissons de conserver cette boule pour une utilisation ultérieure.

C - 1 - 2 L'orientation du modèle

L'étape suivant la description du modèle analytique est l'orientation du modèle. L'orientation que nous donnons à ce modèle est très dépendante de l'utilisation que le concepteur désire en faire. Dans notre cas, nous posons comme entrées de la boite de calcul l'ensemble des paramètres géométriques du dispositif, ainsi que la densité de courant dans les conducteurs.
Pour orienter le modèle, nous utilisons l'assistant d'orientation de MAEL (Figure 5. 5).

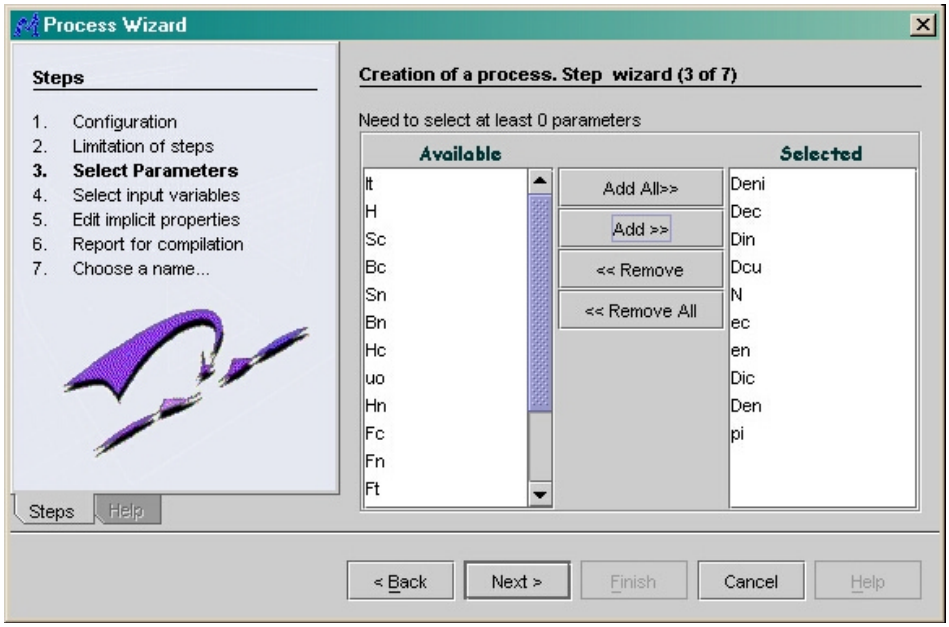


Figure 5. 5: Assistant d'orientation de MAEL (Choix des entrées)

La liste de gauche contient l'ensemble des grandeurs intervenant dans la description du modèle dont le choix en tant qu'entrées est possible. La liste de droite contient les grandeurs typées comme entrées de la boite.

L'étape suivante du processus d'orientation du modèle est entièrement automatisée, il s'agit de la construction d'une séquence de calcul optimale qui permet d'évaluer les sorties de la boite en fonction des entrées choisies.

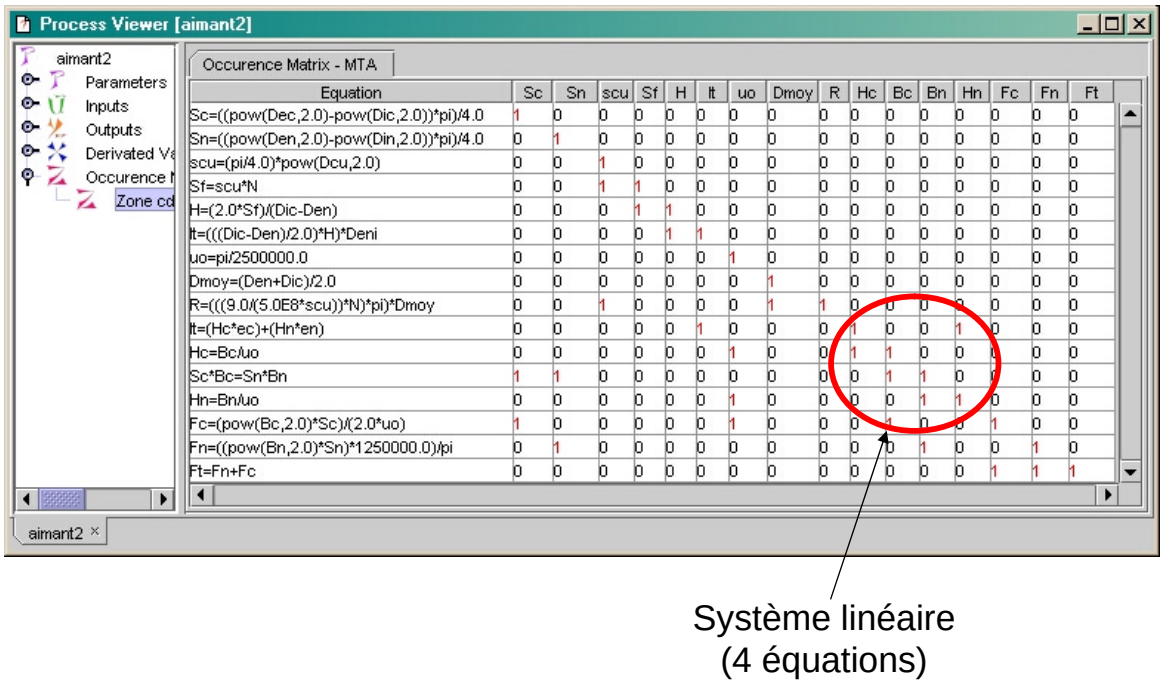
Nous choisissons alors un nom pour la boite qui vient d'être créée, et celle-ci est alors sauvegardée par MAEL.

A ce stade, nous disposons de deux formats de modèles :

- une boule, que nous avons entièrement éditée et documentée grâce aux fonctionnalités de MAEL
- une boite, dont nous avons choisi les entrées et les sorties

Nous avons la possibilité d'évaluer la pertinence de la séquence d'opérations construite automatiquement en observant la matrice d'occurrence du modèle orienté :

- les lignes sont les relations que nous avons données précédemment
- les colonnes correspondent aux sorties de la boite



Equation	Sc	Sn	scu	Sf	H	It	uo	Dmoy	R	Hc	Bc	Bn	Hn	Fc	Fn	Ft
Sc=((pow(Dec,2.0)-pow(Dic,2.0))*pi)/4.0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sn=((pow(Den,2.0)-pow(Din,2.0))*pi)/4.0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
scu=(pi/4.0)*pow(Dcu,2.0)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Sf=scu*N	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
H=(2.0*Sf)/(Dic-Den)	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
It=(((Dic-Den)/2.0)*H)*Deni	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
uo=pi/2500000.0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Dmoy=(Den+Dic)/2.0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
R=(((9.0/(5.0E8*scu))*N)*pi)*Dmoy	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0
It=(Hc*ec)+(Hn*en)	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0
Hc=Bc/uo	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0
Sc*Bc=Sn*Bn	1	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0
Hn=Bn/uo	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0
Fc=(pow(Bc,2.0)*Sc)/(2.0*uo)	1	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0
Fn=((pow(Bn,2.0)*Sn)*1250000.0)/pi	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0
Ft=Fn+Fc	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Figure 5. 6: Matrice d'occurrence du modèle orienté

Ainsi, nous pouvons constater que l'ensemble du problème de l'électro-aimant contient un système linéaire de quatre équations. Nous constatons également que le système linéaire ne fait intervenir que des relations concernant soit la conservation du flux, soit la définition des inductions aux différents endroits.

$$It = (Hc * ec) + (Hn * en)$$

$$Hc = \frac{Bc}{uo}$$

$$Hn = \frac{Bn}{uo}$$

$$Sc * Bc = Sn * Bn$$

La mise en évidence de ce système est normale et correspond à un « problème » fréquent de la modélisation en magnétostatique.

C - 1 - 3 Génération de code et validation du modèle

Nous disposons maintenant de la description d'une boite de calcul le second format de capitalisation, afin de pouvoir l'utiliser en calcul et en dimensionnement, nous devons encore générer le code de calcul correspondant.

Nous choisissons de générer un composant informatique utilisable dans un logiciel de dimensionnement : le COB.

La création de ce COB est entièrement automatique, un compte rendu d'exécution nous informe du bon déroulement des opérations (Figure 5. 7).

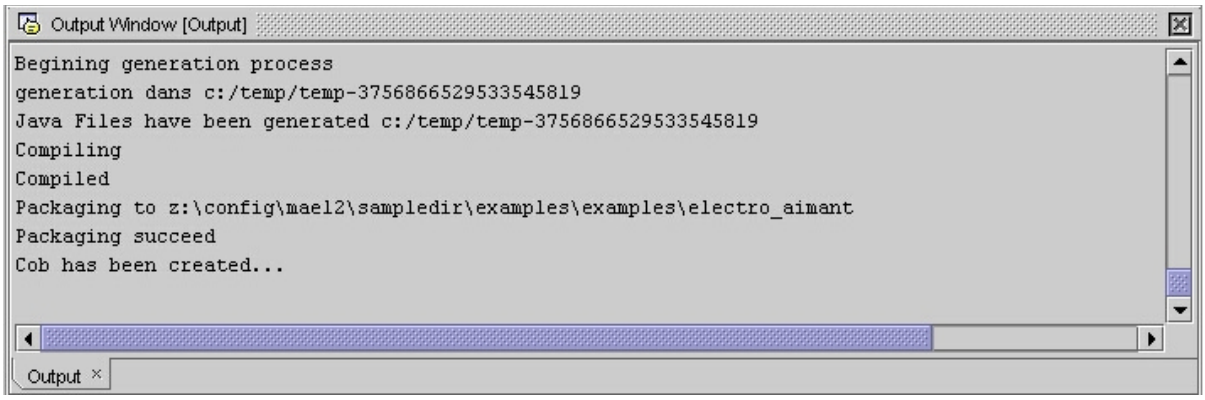


Figure 5. 7: Compte Rendu de génération

Nous utilisons ce composant de calcul pour valider le modèle décrit précédemment. Un traceur de courbe et un outil de visualisation des valeurs des sorties en fonction des entrées nous permet de tester le modèle (Figure 5. 8). Nous représentons ainsi les évolutions des forces s'appliquant sur le plateau (en newton) en fonction de la densité de courant dans la bobine (en A/m^2).

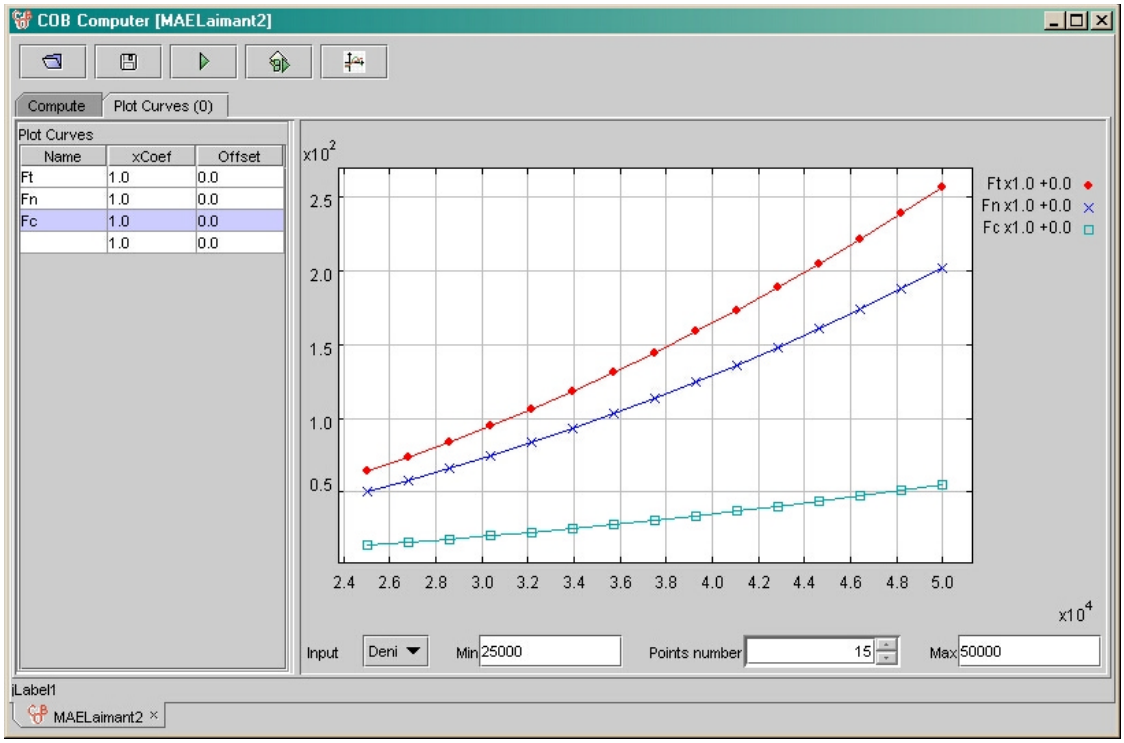


Figure 5. 8: Visualisation et validation du modèle

Le même composant de calcul est ensuite utilisé pour réaliser le dimensionnement contraint de la structure. Nous utilisons un outil réalisé dans le cadre des travaux de recherche de David Magot : le logiciel CDIOptimizer.

Nous définissons le cahier des charges qui correspond à notre besoin :

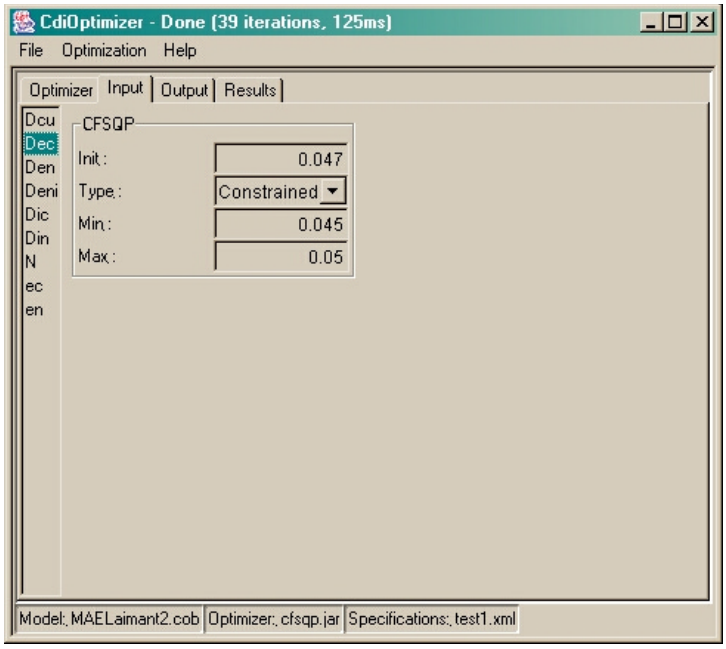


Figure 5. 9: Définition d'un cahier des charges pour le dimensionnement

Nous utilisons alors le composant de calcul généré pour réaliser l’optimisation de l’actionneur avec le cahier des charges suivant :

- Diamètre Fil contraint dans [0.2 mm ; 1 mm]
- Hauteur totale contrainte dans [100 mm ; 50 mm]
- Diamètres extérieur de la culasse contraint dans [45 mm ; 50mm]
- Diamètre intérieur du noyau contraint dans [5mm ; 10 mm]
- Diamètre extérieur du noyau contraint dans [10mm ;15mm]
- Le nombre de spire de la bobine contraint dans [500,5000]

On cherche à maximiser la force d’appel sur le plateau en limitant les inductions dans la culasse et le noyau pour éviter la saturation.

Les résultats sont alors donnés sur la Figure 5.10 :

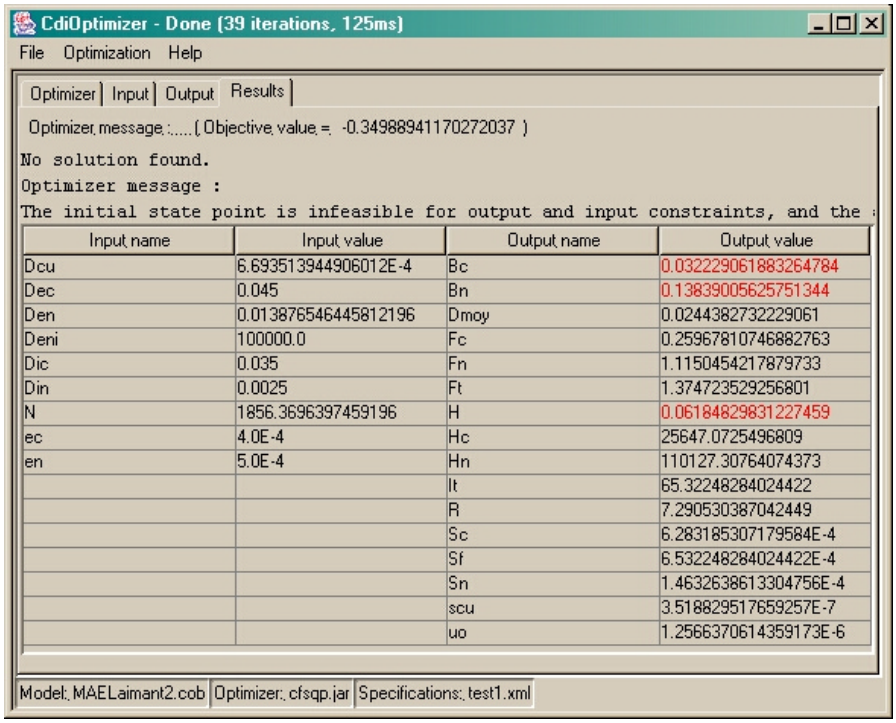


Figure 5. 10: Résultat de l'optimisation

Nous avons ainsi montré la validité de la démarche proposée et des outils mis en œuvre en ce qui concerne les aspects de dimensionnement de structure.

D - Réalisation d'un bloc de simulation

D - 1 Présentation de la machine

Nous réalisons un bloc de simulation, utilisable dans SIMULINK, contenant une modélisation de la machine asynchrone. La Figure 5. 11 illustre la modélisation de Park que nous utilisons dans ce cas.

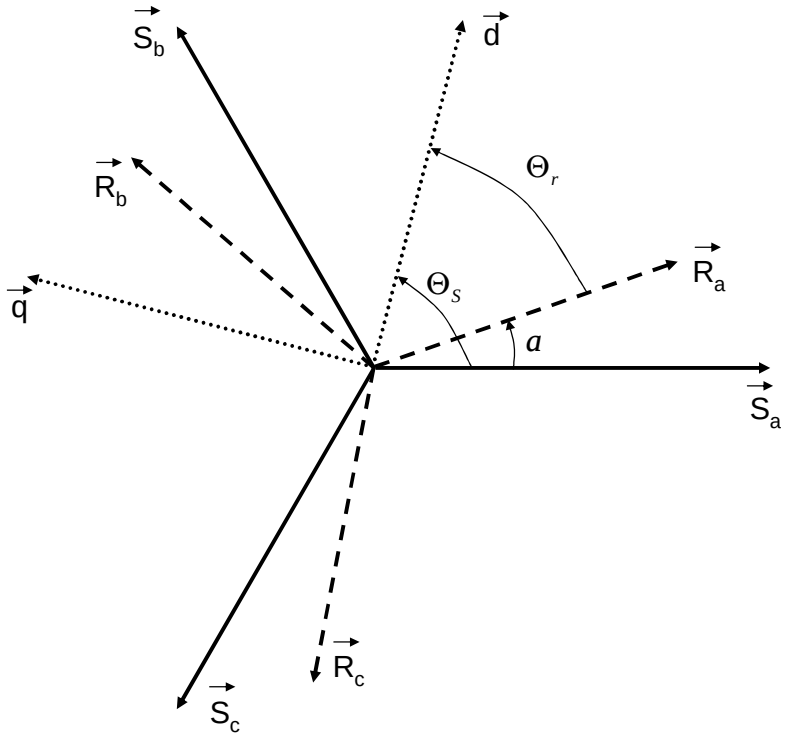


Figure 5. 11: Modélisation de Park de la machine asynchrone

Nous ne rappelons pas ici les équations de ce modèle de simulation de la machine asynchrone. Nous ferons une modélisation avec les axes liés au stator.

D - 2 Utilisation de MAEL pour décrire la boite de calcul

D - 2 - 1 Description du modèle

Comme pour le modèle de dimensionnement de l'électro-aimant, nous devons tout d'abord décrire la modélisation de Park sous forme de ses équations dans l'éditeur d'équations de MAEL (Figure 5. 12).

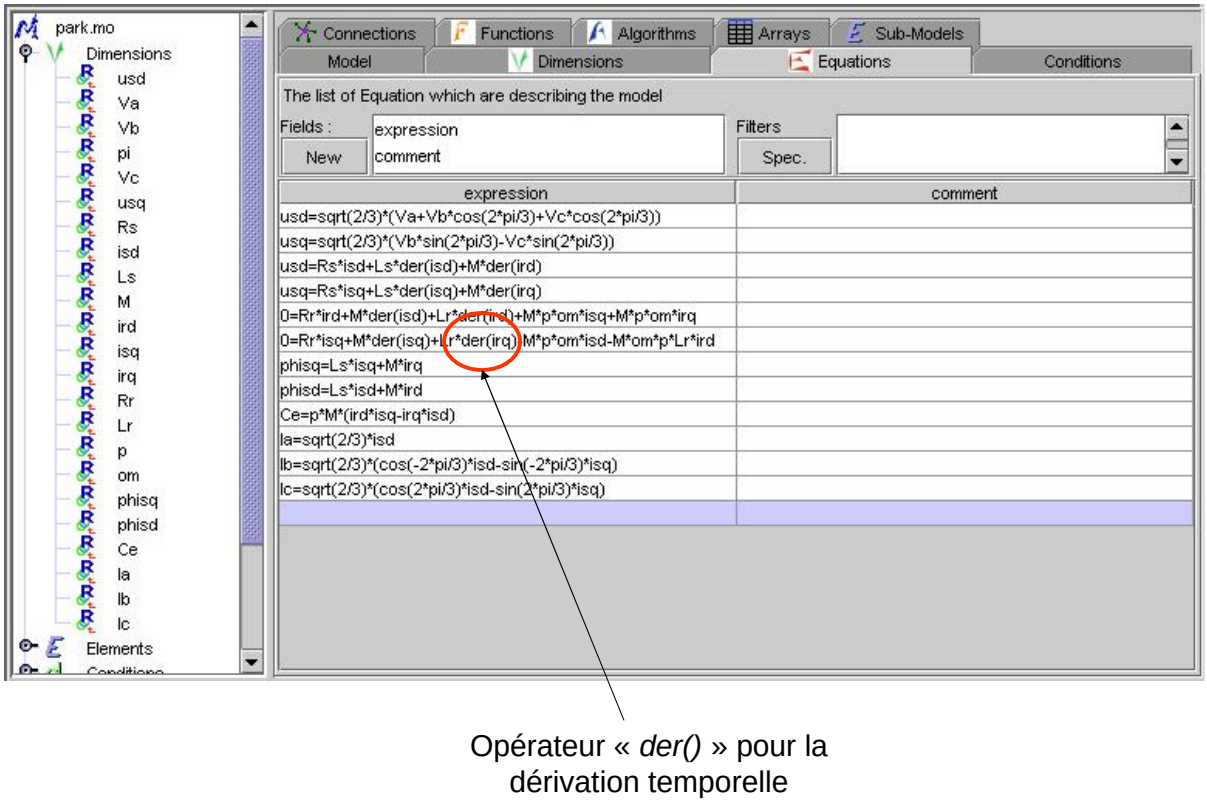


Figure 5. 12: Description du modèle de Park

Nous noterons ici l'utilisation de l'opérateur **der()** pour marquer la présence d'une dérivée temporelle.

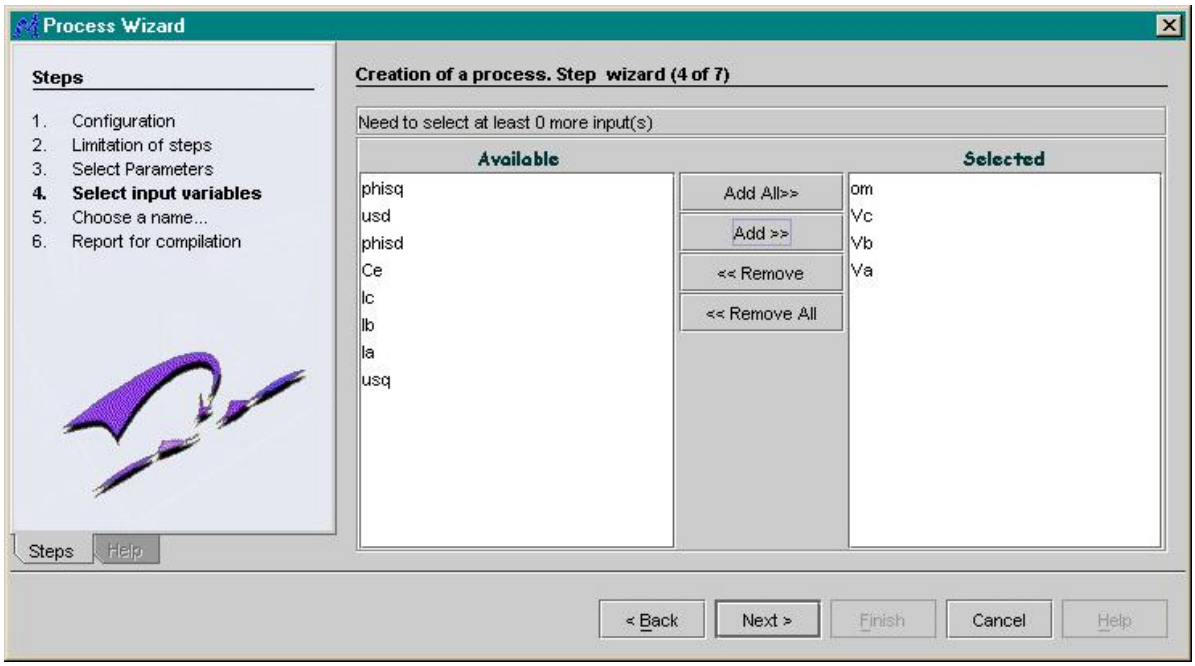
Puis nous documentons le modèle en attachant un fichier décrivant les notations choisies dans les équations. Ce fichier est produit par le concepteur dans un éditeur graphique classique ou bien utilisé depuis une librairie d'images.

D - 2 - 2 Choix d'une orientation

L'orientation de la boule obtenue après description du modèle est nécessaire pour la création d'un bloc de calcul. Dans le cas de la création d'une boite pour la simulation, l'orientation du modèle se compose de deux étapes :

- le choix des paramètres, les grandeurs constantes au cours du temps
- le choix des entrées

Le choix des paramètres est identique à celui des paramètres d'un modèle de dimensionnement. Nous avons rajouté une étape intermédiaire qui est le choix des entrées. Cette étape est semblable à celle du choix des paramètres (Figure 5. 13) :



Les grandeurs dérivées par rapport au temps n'apparaissent pas dans la liste des choix possibles

Figure 5. 13: Choix des entrées du modèle

Nous remarquons notamment que la liste du choix des entrées possible ne contient les grandeurs qui apparaissent dérivées par rapport au temps (1). En effet, nous avons automatiquement empêché le choix de grandeurs dérivées comme entrées du système. Il est néanmoins possible de désactiver ce filtre automatique à la demande. Nous renvoyons le lecteur au paragraphe D-2-3 du chapitre 3 pour de plus amples explications.

Le reste des opérations est réalisé automatiquement ce qui permet de construire une séquence optimale de calcul. La matrice d'occurrence de cette séquence de calcul fait apparaître 2 systèmes linéaires de deux équations Figure 5. 14, ce qui est un résultat classique pour cette application.

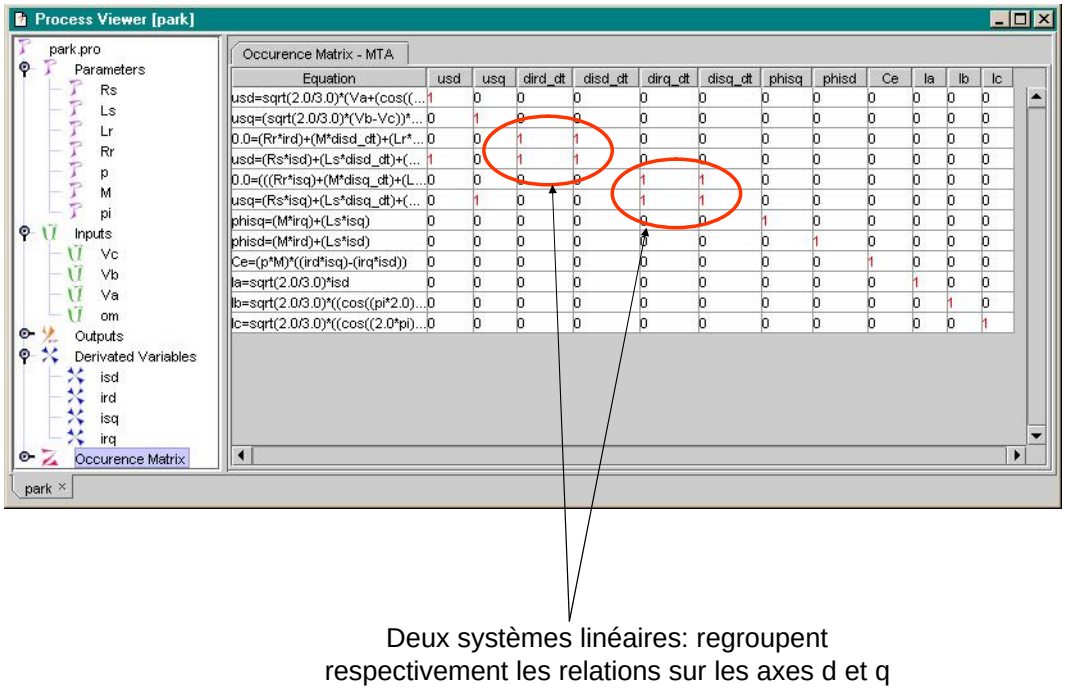


Figure 5. 14: Séquence de calcul pour le modèle de Park

D - 2 - 3 Génération de code et utilisation du bloc dans MATLAB/SIMULINK

Nous utilisons, ici, un générateur de code développé par Hichem Chetouani dans le cadre d'un DESS. Ce générateur de code s'appuie sur HAGEL et est utilisable pour la création des S-Functions de Simulink. La génération de la S-Function permet de créer un bloc de calcul utilisable dans une simulation d'un actionneur asynchrone alimenté par une source de tension triphasée. Ceci est illustré sur la Figure 5. 15. Cette figure présente également l'évolution obtenue pour le couple électromagnétique de la machine en fonction du temps.

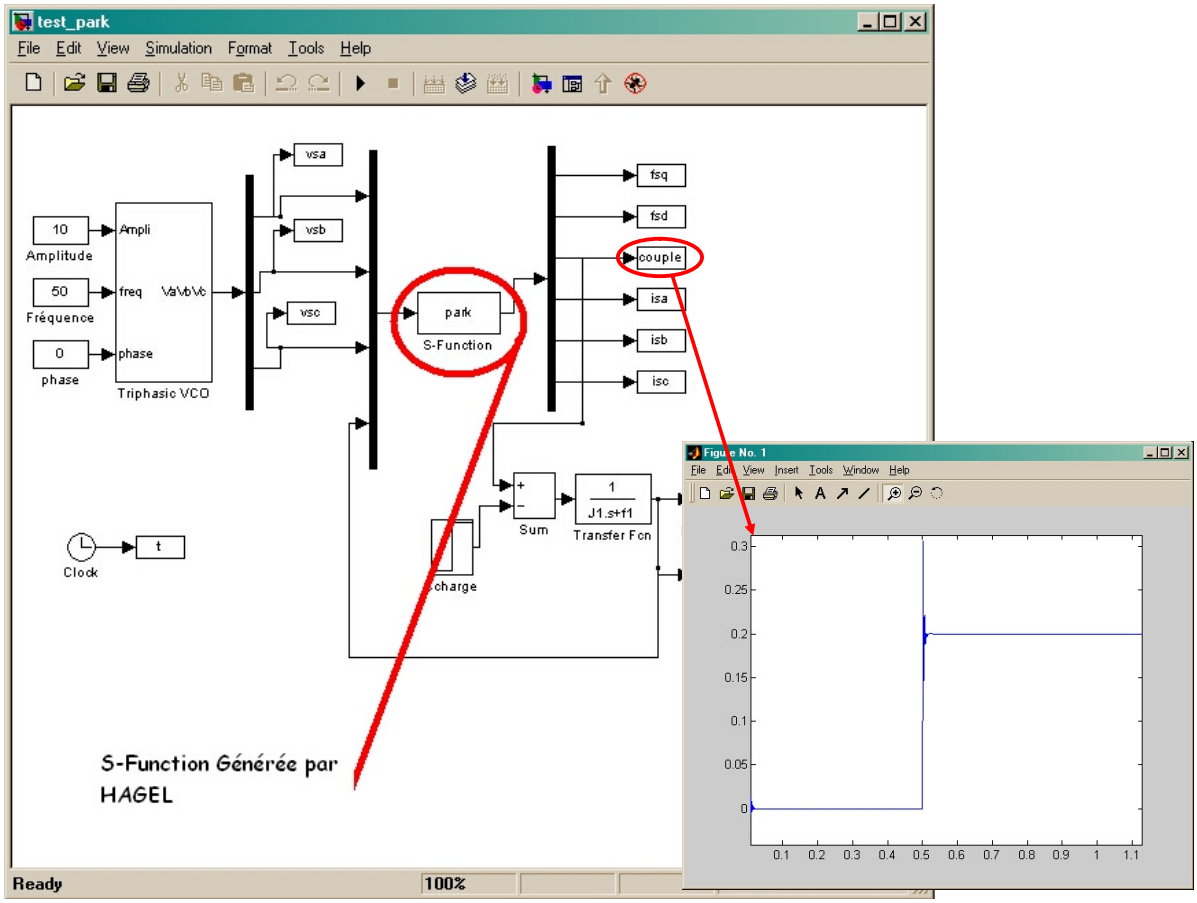


Figure 5. 15: Utilisation du bloc de simulation de la machine asynchrone – Visualisation du couple électromagnétique en N.m^{-1}

Les courbes montrées ici correspondent au comportement d'un moteur asynchrone d'une puissance 200 W utilisé dans les automatismes de l'habitat.

Nous avons ainsi montré la validité de notre approche dans le cadre de la construction de blocs de simulation.

E - Avantages et inconvénients de la démarche

La démarche proposée présente un certain nombre d'inconvénients :

- le concepteur est toujours tributaire d'un outil pour l'édition de ses modèles
- la mise au point des modèles dans l'environnement proposé est parfois fastidieuse car pour tester ceux-ci, il est nécessaire de rejouer le scénario d'orientation et de projection à chaque modification réalisée sur les équations
- le choix des méthodes numériques pour la résolution des problèmes non linéaires est encore un travail pénible qui demande une certaine expertise au concepteur

- le formalisme de description de l'expertise est essentiellement mathématique, sous la forme d'équations ou d'algorithmes. Ainsi, il ne prend pas en compte des modèles complexes de type éléments finis.

Mais, la démarche proposée présente également un certain nombre d'avantages :

- la possibilité de décrire le modèle une fois puis d'en disposer dans de nombreux environnements de calculs
- la capitalisation de l'expertise du concepteur
- une forte incitation à la documentation des modèles
- une grande souplesse d'utilisation, notamment dans la description de nouvelles cibles logicielle pour la génération de code
- les différents formats de représentation du modèle sont capitalisés pour le concepteur : la boule, les boites blanches, grises et noires.
- une démarche globale pour la modélisation statique et en transitoire des structures est proposée.

A cela subsiste la limitation que l'on rencontre pour certaines cibles, en terme de projection de modèles. Ainsi, on peut avoir besoin de voir ou modifier le code générer pour tester le modèle, par exemple des algorithmes de commande (déboguage).

F - Conclusion

Nous avons montré la faisabilité et la pertinence de notre approche et de nos outils dans les contextes de la simulation et du dimensionnement de structure. A l'heure actuelle, l'ensemble du processus de description des applications contextualisées n'est pas encore automatisé. Toutefois, les briques de bases de l'édifice sont posées et il ne reste désormais plus qu'à développer les automates pour la description de scénarii d'analyse, et d'encapsulation d'applications de simulation contextualisée. La faisabilité « manuelle » de telles études a déjà été démontrée [GER-2].

Conclusion et perspectives

Conclusions et perspectives

Au cours de ces travaux de thèse, nous avons proposé une méthodologie de travail pour le dimensionnement d'associations Machine Convertisseurs Commande. La nécessité de prendre en compte des critères issus de l'analyse d'une simulation temporelle nous a, en effet, poussé à mettre en place une méthode de travail pour le concepteur désirant décrire un problème de dimensionnement intégrant l'analyse de résultats de simulation temporelle complexe.

Les besoins que nous avons ainsi mis en évidence, en terme de modélisation ont amené un second problème : celui de la capitalisation des modèles du concepteur, et en particulier une capitalisation hors contexte d'utilisation. Nous avons ainsi proposé un double formalisme de capitalisation des modèles du concepteur :

- des boules, capitalisation hors contexte et documentée, reprenant les concepts de la Modélisation Orientée Objet
- des boites pour la représentation d'un processus de calcul informatique utilisable pour la génération de code.

Nous avons également formalisé et automatisé la démarche de transformation des boules pour créer des boites de calcul, cette démarche étant fastidieuse et productrice d'erreurs si elle est réalisée par le concepteur.

Dans la suite de cette démarche, une logique de génération automatique de code fondée sur la description de squelettes de code est proposée, qui permet de capitaliser et de partager une autre forme d'expertise : l'écriture du code informatique dédiée à un outil.

L'ensemble de ces propositions est implanté dans un environnement informatique constitué de deux modules : MAEL et HAGEL. MAEL contient un support de capitalisation des modèles sous les deux formats que nous avons proposés. Il contient également l'ensemble des automatismes nécessaires pour la transformation des modèles. HAGEL est, quant à lui, une architecture pour la génération automatique de code informatique.

L'environnement développé est aujourd'hui centré sur une représentation mathématique des modèles. Il faudrait se poser la question de l'intégration d'autres formalismes qui permettent eux aussi de représenter une certaine connaissance et apportent, entre autres, de la

complémentarité et des éclairages différents à l'activité de modélisation. Par exemple, comment intégrer des représentations telles que celle des Bond-Graph dans notre système ?

La réalisation de l'outil MAEL a confirmé la faisabilité et l'intérêt de l'automatisation des transformations formelles sur les modèles. Toutefois, les méthodes utilisées présentent quelques limites, notamment l'impossibilité, pour le concepteur, de choisir les grandeurs dérivées à conserver dans le vecteur d'état ou encore de choisir les grandeurs implicites de son problème. Or le choix de ces grandeurs peut avoir une influence sur les propriétés numériques des modèles. Il serait donc intéressant de donner au concepteur la liberté de choix pour ces variables clés.

Les expériences que nous avons pu mener en génération de code nous ont montré l'importance d'un choix judicieux des méthodes numériques. En effet, nous avons pu observer que la nature des modèles influe fortement sur les méthodes numériques à utiliser pour la résolution des problèmes. Ceci est une part de l'expertise du concepteur qui manipule ses modèles et connaît leurs capacités de convergence numérique. Nous pouvons envisager de fournir au concepteur un guide pour le choix des méthodes numériques à utiliser.

MAEL fournit un support de capitalisation des modèles et HAGEL un support de génération de code pour différentes cibles. Il manque, aujourd'hui, une étape dans le processus d'aide à la formulation des modèles pour le concepteur : des outils de génération de modèles métiers du concepteur. Ainsi, en électronique de puissance, il est intéressant de disposer de générateurs de modèles exploitables pour la conception [VER], [GER-1]. De même, des outils d'aide à la formulation de modèles magnétiques par réseaux de réluctances présentent des attraits forts séduisants pour le concepteur de systèmes électromécaniques.

Aujourd'hui, nous ne générons qu'un type de bloc de calcul : les COB pour le dimensionnement. Néanmoins, la génération de blocs d'une autre nature ouvre des horizons nouveaux pour le dimensionnement de structure. Dans le même état d'esprit, il faudrait créer des générateurs de modèles pour d'autres outils de simulation tels que DYMOLA ou 20-Sim.

Des développements, aujourd'hui en cours, mettent en exergue la limitation du langage algorithmique choisi dans l'environnement : le C. En effet, celui-ci répond à un certain nombre d'exigences initiales de l'environnement et de la méthodologie. Toutefois, il

n'apporte par de réponses pour une représentation générique des lois de commande. Cette limitation apparaît notamment avec la génération de code pour le langage VHDL-AMS. Ainsi, la gestion du pas de calcul en simulation, des évènements pour les systèmes hybrides nous posent des problèmes. Une solution pourrait être la création d'un langage de niveau supérieur pour la description des structures algorithmiques de tests. Le langage VHDL-AMS peut-il nous apporter une réponse ?

Dans la suite de cette réflexion, nous pouvons aujourd'hui envisager de réaliser des outils d'aide à la mise au point des modèles de calculs et de simulation. En effet, nous avons implanté des fonctionnalités de test des modèles de calculs, mais la réalisation des modèles de simulation, en particulier pour les lois de commande, reste une activité complexe et difficile à effectuer. Ainsi, un système d'aide à la mise au point par l'implantation de fonctionnalité de déboguage, par exemple, est une perspective intéressante pour la suite des ces travaux.

Dépassant le cadre du génie électrique et de la manipulation formelle, MAEL propose une structuration et une capitalisation de la ressource modèle de l'entreprise. A long terme, il serait envisageable de gérer cette information au même titre que les données techniques des dispositifs fabriqués (SGDT). Se pose alors la question de la place de la ressource de modèles dans le cycle de vie du produit. Comment exploiter les retours d'expériences sur les produits? Comment intégrer cette connaissance dans la capacité de diagnostic et de maintenance des applications ?

Enfin, l'architecture de MAEL, malgré le soin apporté, présente encore des limites. En effet, il semblerait plus judicieux de séparer la fonctionnalité de capitalisation de celle de traitement des modèles. Aujourd'hui, si l'un peut se faire sans l'autre, les deux éléments sont implantés dans le même module. Par ailleurs, la description des agrégations de modèles pourrait se faire par le biais d'une interface graphique plutôt que par la déclaration textuelle des équivalences.

Bibliographie

Bibliographie

[**20S**] 20-Sim, http://www.20sim.com

[**ALL**] B. Allard, H. Morel et J.P. Chante, "Power Electronic circuit simulation using bond graph and Pétri network techniques ", PESC'93, 24th annual Power electronics Specialists Conference, University of Washington, Seattle, USA, June 20-24

[**ATI**] E.Atienza, "Méthodologie et Outils pour le dimensionnement", Thèse de doctorat en génie électrique de l'Institut National Polytechnique de Grenoble, France, 4 juillet 2003

[**BEL**] B. Bel Habib, F. Wurtz, J. Bigeon, E. Atienza, "An integrated platform for electrical design", IEEE-IECON'99, San Jose, CA, USA, November 29th - December 3rd, 1999.

[**COU**] C. Coutel, "Contribution méthodologique à la conception sous contraintes de dispositifs électromagnétiques", Thèse de doctorat en génie électrique de l'Institut National Polytechnique de Grenoble, France, 20 octobre 2000

[DAV] J. Davenport, Y. Siret, E. Tournier, "Calcul formel. Systèmes et algorithmes de manipulations algébriques". Editions Masson 1993 275 pp., ISBN : 2 225 84200

[**DEL**] B. Delinchant, "Un Environnement a base de Composants Integrant le Concepteur et ses Outils pour de Nouvelles Methodes de CAO", Thèse de doctorat en génie électrique de l'Institut National Polytechnique de Grenoble, France, 10 juillet 2003

[**DOL**] U. Dolinsky, "Interpreter for symbolic manipulation of mathematical expression", http://www.mb.hs-wismar.de/Mitarbeiter/Pawletta/00Uwe/casE.html

[DUC] J.P. Ducreux, A. Castelain, C. Rombaut, G. Dauphin-Tanguy, "Application de l'outil Bond-Graph à la modélisation des convertisseurs statiques de l'électronique de puissance", Extrait du rapport des Journées EEA – électrotechnique, 26-27 Mars 1992, Conservatoire National des Arts et Métiers Electrotechnique, Ecole centrale de Lille

[**DUF**] I.S. Duff, J.K. Reid, "An implementation of Tarjan's Algorithm for the Block Triangularization of a Matrix", ACM, Transactions on Mathematical Software, Vol. 4, N°2, Juin 1978, p.137-147

[**DYM**], Dymola, http://www.dymola.com

[**ECO**] EcoSim-Pro, http://www.ecosimpro.com

[**ELM-1**] H. Elmqvist, F.E. Cellier, M.Otter, "Object oriented moddeling of hybrid system", ESS'93 European Simulation Symposium, Delft, Netherlands, 1993

[**ELM-2**] H. Elmqvist, S.E. Mattsson, M. Otter, "Modelica - An International Effort to Design an Object-Oriented Modeling Language", Computer Simulation Conference '98, Reno, Nevada, USA, 19-22 Juillet 1998

[**EPO**] Epogy, http://www.synaps-inc.com

[**FAB**] G.Fabian, D.A van Beek, J.E. Rooda, “Integration of the discrete and the continuous behaviour in the hybrid Chi Simulator“, Proceedings ESS 1998, European Simulation Symposium, Manchester, 1998 p.252-257

[**FOP**] Formating Object Processor, http://www.apache.org

[GER-4] L. Gerbaud, A. Bolopion, J. Bigeon, "Symbolic and object oriented approach for the simulation of electric drives", Simulation in Industry, 8th SCS-ESS’96 (Europeen Simulation Symposium), Genoa, Italy, October 24-26, 1996, pp 287-288

[**GER-1**] L. Gerbaud, C. Lechevalier, A. Bolopion, J. Bigeon, "Modélisation et simulation à topologie variable des convertisseurs statiques et des entraînements électromécaniques", The European Physical Journal - Applied Physics, Volume 2, Number 2, June 1998, pp 235-252

[**GER-2**]Gerbaud L., Atienza E., Bolopion A., Fandino J., "An Optimisation Process of Power Electronics applications", COMPEL : The International Journal for Computation and Mathematics in Electrical and Electronic Engineering, MBC University Press COMPEL, Vol. 20, N°3, 2001, pp 879-890

[**GER-3**] L. Gerbaud, J. Bigeon, G. Champenois, "Modular approach to describe electromechanical systems. Using Macsyma to generate global approach simulation software", Conference record of the IEEE PESC'92 (Power Electronics Society Conference), june 29 - july 3, 1992, Toledo, Spain, pp 1189-1196

[**HAU**] J.P. Hautier, J.P Caron, "Convertisseurs statiques: Méthodologie causale de modélisation et de commande", chez Technip, ISBN : 2710807459

[HAU-1] J.P. Hautier et G. Manesse, "Utilisation des réseaux de Pétri pour l’analyse des systèmes électrotechniques", Techniques de l’ingénieur D3740.

[**ISI**] iSIGHT, http://www.engineous.com

[JAN] R. Janssen, C.P. Riley, R.C.F. McLatchie, C. Glasgow, T. Gutierrez, J. Simkin, P. Brochet, C. Furmaniak, F. Gillon, G. Molinari, P. Alotto, J-F Lemoine, G. Drago “An environment for the optimization of electromagnetic design“, IEEE Trans. On Magnetics Vol. 36, N° 4, Juillet 2000p.1640-1644

[**KAR**] D.C. Karnopp, D.L. Margolis, R.C. Rosenberg, "System dynamics : a unified approach", 2nd edition, 1990, John Wiley and Sons

[**KRA**] Kragh H., Blaabjerg F., Pedersen J. K., "An advanced tool for optimised design of power electronic circuits", proceeding of IEEE-IAS'98, Saint Louis, Missouri, USA, october 12-15, 1998, pp 991-998

[**LAR**] C.Larouci, "Conception et optimisation de convertisseurs statiques pour l'électronique de puissance. Application aux structures à absorption sinusoïdale", Thèse en spécialité génie électrique, de l'Institut National Polytechnique de Grenoble, 13 mai 2002

[**LAV**] N. Laverdure, I. Valero, S. Bacha, D. Roye, L. Gerbaud, "Optimisation de l'interfaçage de puissance dans les systèmes éoliens", GEVIQ'2002, Colloque National Génie Electrique Vie et Qualité, Marseille, 12-13 juin 2002

[**M/S**] MATLAB, MATLAB/SIMULINK, http://www.mathworks.com

[**MAPLE**] MAPLE, http://www.maplesoft.com

[**MATH**] MathCAD, http://www.mathsoft.com

[**MATH-2**] MATHEMATICA, http://www.wolfram.com

[**MOD**] M.M. Tiller, "Introduction to physical modling with Modelica", Kluwer Academic Publishers, 2001, ISBN: 0792373677

[**NOR**] O. Normand, A. Bolopion, D. Roye, L. Gerbaud, "Object oriented simulation of electromechanical systems", European Simulation Symposium, SCS-ESS'94 (Europeen Simulation Symposium), Istanbul, Turkey, October 9-12, 1994, Vol III, pp 71-73 (3 pages).

[NOR-2] O. Normand, "Conception d'un outil général de simulation des systèmes de conversion d'énergie électrique et de leur commande", Thèse de doctorat en génie électrique de l'Institut National Polytechnique de Grenoble, France, Septembre 1992

[**NETB**] T. Boudreau, J. Glick, S. Greene, V. Spurlin, J.J. Woehr, "NetBeans: The Definitive Guide", Octobre 2002, O'Reilly, ISBN : 0-596-00280-7

[**NOU**] Nougier, "Méthodes de calcul numériques", Hermès, ISBN: 2746202786

[**PDF**] Portable Document Format, http://www.adobe.com

[**PRO**] Pro@Design, http://www.designprocessingtechnologies.com

[**SAB**] SABER, http://www.synopsys.com/

[**SAU**] C. Sauvey, "Contribution méthodologique à la modélisation pour le dimensionnement de moteurs à réluctance variable", Thèse de doctorat en génie électrique de l'Institut National Polytechnique de Grenoble, France, 8 septembre 2000

[**SIM**] Simplorer, http://www.ansoft.com

[**TAN**] G. Dauphin-Tanguy, "Les bond graphs et leur application en mécatronique", Techniques de l'Ingénieur, S7222

[**TAR**] R.E. Tarjan, "Depth first search and linear graph algorithms", SIAM Journal of. Computing, Vol 1, n°2, 1972, p.146-160

[**THE**] L. Thevenon, "Représentation des systèmes hybrids complexes par flux de données: développement d'un outil de modélisation et de simulation des procédés batch", Thèse de

doctorat en automatique et productique de l'Institut National Polytechnique de Grenoble, France, 16 octobre 2000

[**TKS**] TK-Solver, http://www.uts.com

[**TOU**] F. Tourkhani, "Environnement de CAO pour l'optimisation du dimensionnement global des convertisseurs statiques. Application à la CAO d'un onduleur multiniveaux". Thèse de l'université Laval, Québec, Dept. de génie électrique, Janvier 1996

[**VAN**] D.A. van Beek, J.E. Rooda, "Languages and application in hybrid modelling: positioning of CHI", Proceedings of the 9th Symposium on information Control in manufacturing, Nancy, Vol II, 1998, p.77-82

[**VER**] F. Verdiere, S. Bacha, L. Gerbaud "Automatic modelling of static converters averaged models", EPE'03 Toulouse, France, Septembre 2003

[**VHDL**] http://www.vhdl.org

[**W3C**] World Wide Web Consortium, http://www.w3c.org

[**WUR**] F. Wurtz, "Une nouvelle approche pour la conception sous contraintes de machines électriques". Thèse de doctorat en génie électrique de l'Institut National Polytechnique de Grenoble, France, 28 Mai 1996

En gras, nous marquons les éléments bibliographiques référencés dans le rapport.

Annexe A

Génération de code HAGEL

Annexe A Génération de code : HAGEL (Hybrid Automaton for Generation of Equations and Logic)

A - Besoins de génération de code

A - 1 Pourquoi générer du code ?

L'approche proposée dans cette thèse permet d'automatiser un certain nombre de traitements formels effectués sur un modèle dans le cadre de la contextualisation de la connaissance. La première phase de cette approche permet de créer une description d'un processus de calcul qui est ensuite utilisée pour écrire un code informatique destiné à être utilisé dans un environnement de calcul scientifique spécifique.

La seconde étape de cette approche est l'écriture d'un code informatique pour répondre aux spécifications d'un outil de calcul ou d'un composant informatique de calcul. Cette phase peut être, dans une large mesure automatisée. En effet, la compétence d'un modélisateur n'est pas dans l'écriture d'un code informatique, il s'agit d'un métier à part entière dont certaines activités sont extrêmement pénibles. Au nombre de celles-ci, la mise au point d'un code avec la correction des erreurs de « recopiage » ou de respect des normes est très pénalisante. Il est donc intéressant d'envisager un système de génération automatique de code basé sur la mise au point d'un patron de génération qui permet de remplir le patron par des données apportées par l'extérieur.

A - 2 Les besoins spécifiques de génération

Dans le cadre de ce projet, les données utilisées pour la génération de code sont stockées dans un fichier au format XML, il s'agit des fichiers PROCESS dont la description est donnée en Annexe B. Le générateur de code doit donc pouvoir manipuler des fichiers au format XML et retranscrire les données qui s'y trouvent pour créer un fichier de code compilable et exécutable dans le contexte d'un environnement dédié.

Outre les fichiers XML, les informations de génération pourront être disponible dans un Objet informatique, cette contrainte apportant souplesse et rapidité de génération.

Par ailleurs, il est nécessaire de pouvoir gérer une multitude de fichiers dans le cas de la génération de gros problème de simulation, il peut s'avérer nécessaire de fragmenter les fichiers de code ou certaines fonctions. Ainsi des opérations de gestion des tailles de texte doivent être disponibles.

De plus, il est nécessaire de pouvoir développer à peu de frais :

- de nouveaux patrons de génération : dans un langage aussi proche que possible de la cible
- de pouvoir ajouter de nouvelles fonctionnalités pour pouvoir répondre à de nouveaux problèmes de génération

Enfin, il doit être possible de récupérer des informations concernant l'activité de génération, report d'erreur ou de succès, quantité et identités des fichiers générés.

La figure Figure A. 1 présente un bilan des fonctionnalités que doit remplir le générateur de code.

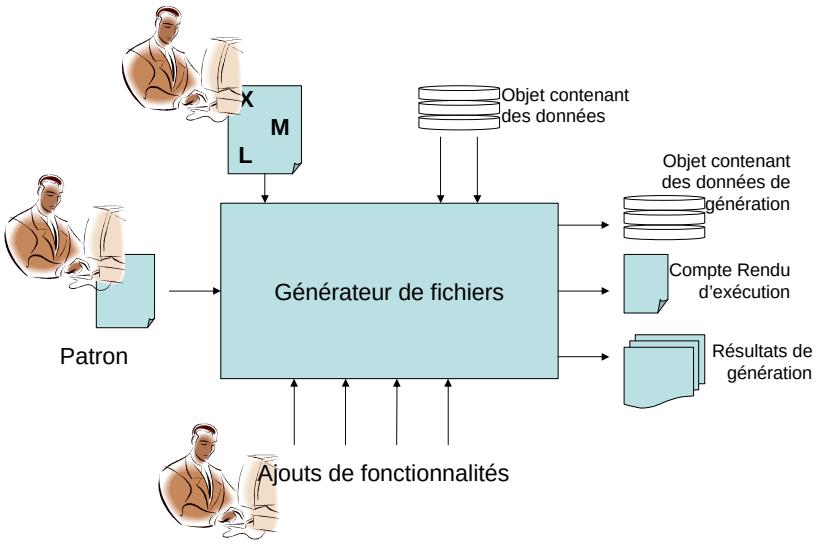


Figure A. 1: Illustration du cahier des charges du générateur de code

A - 3 Bilan des outils existants

Nous présentons ici un rapide bilan des outils informatique existants qui rentrent dans la problématique de génération de fichiers décrits par des templates (patrons). Ces outils sont disponibles en libre diffusion à l'exception de l'outil CDICC, développé dans le cadre de travaux antérieurs au sein du LEG.

Outils	Simplicité du langage de description des patrons	Ajouts de fonctionnalités supplémentaires	Données d'entrée Fichier XML	Données d'entrée Objet	Données de sortie Compte Rendu textuel	Données de sortie Objet encapsulant des données de génération
e-gen	-		✖			
XSLT	-	✖	✖			
CDICC	++			✖		

B - Solution proposée

B - 1 Présentation de l'architecture générale

Afin de remplir le cahier des charges que nous nous sommes fixé, il a été nécessaire de produire un outil permettant la génération de code dans les conditions explicitées ci-dessus. La Figure A. 2 illustre l'architecture du générateur développé dans le cadre de ce projet.

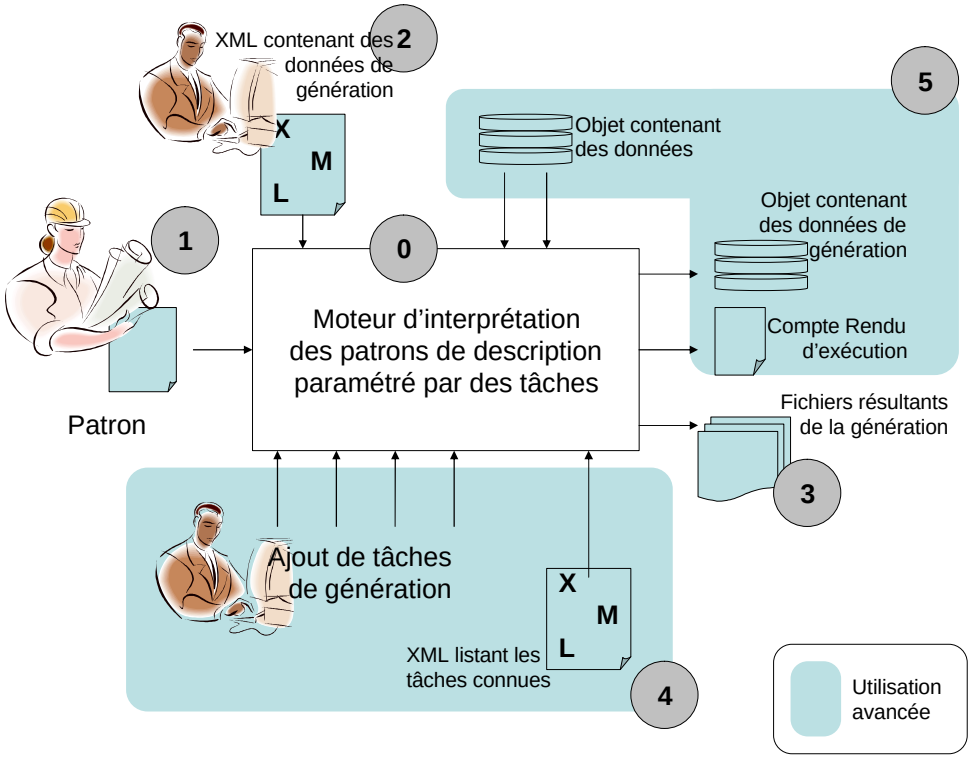


Figure A. 2: Solution logicielle proposée pour HAGEL

B - 2 Ajout de tâches

Le principe de base de HAGEL est l'évolutivité et l'extensibilité à toutes les fonctions voulues par le développeur. Pour réaliser cela, nous avons mis en place une structure Objet, basée sur le langage JAVA, afin de permettre l'ajout de fonctionnalités au langage.

Cette extensibilité est rendue possible par l'introduction d'une classe de base qui définit des actions à effectuer pour réaliser une génération : il s'agit de la classe abstraite **Task**. En créant un objet qui hérite cette classe, le développeur peut ajouter autant de fonctionnalité qu'il le souhaite aux capacités du générateur. L'ajout de tâche se fait donc en créant un nouvel objet et en spécifiant un mapping entre la déclaration informatique de cet objet et un mot clé choisi par le concepteur : cette association est réalisée en ajoutant une information dans le fichier XML qui contient la description des tâches connues par le moteur de génération. La figure

présente une vue UML de la classe Task dont les fonctionnalités additionnelles doivent hériter.

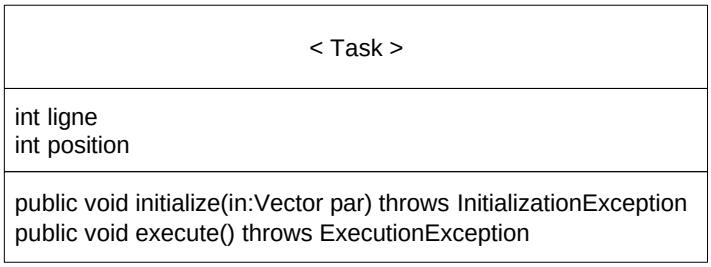


Figure A. 3: Vue UML de la classe abstraite *Task*

La classe Task est définie par deux méthodes à implémenter, deux indicateurs de position permettent d'effectuer une mise au point des patrons. Outre ces spécificités, toute classe héritant de Task doit décrire un constructeur sans paramètres.

B - 3 Fonctionnement de HAGEL

HAGEL est avant tout un langage de haut niveau dédié à la génération de code. Il s'agit d'un langage interprété car il n'exécute pas directement les opérations qui sont définies dans les descriptions de patron. La Figure A. 4 illustre le fonctionnement de HAGEL.

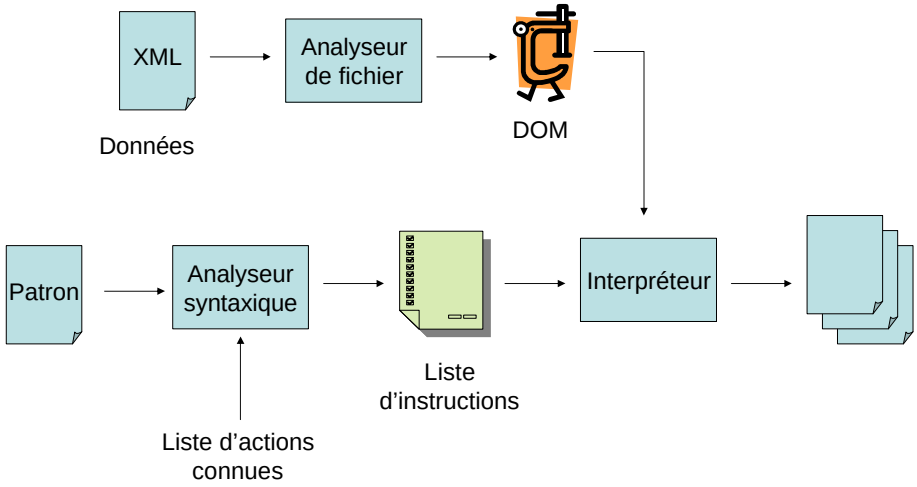


Figure A. 4: Fonctionnement de HAGEL

HAGEL fonctionne en deux temps :

- l'**initialisation** : il charge un patron, c'est-à-dire qu'il analyse un fichier texte qui contient la description de ce que le développeur désire générer. Un parser (analyseur syntaxique) est chargé de cette tâche, il identifie des marqueurs qui permettent de repérer des mots clés. Ces mots clés sont ensuite utilisés en

conjonction avec la liste des actions connues pour construire une liste d'instructions.

- L'**exécution**, il charge des données dans un fichier XML. Grâce à un analyseur syntaxique, une représentation arborescente interne des données est construite : le DOM (Document Object Model). Cette représentation est utilisée pour interpréter la liste d'instruction en lui permettant par le biais d'API XPath (langage propre au XML permettant d'évoluer dans le DOM) d'accéder aux données contenues dans le fichier XML. L'interprétation des données produit les fichiers résultats de la génération.

C - Exemple d'utilisation

C - 1 Définition de la structure des patrons

Nous utilisons, pour définir cette structure une notation classique en théorie des langages.

La structuration des fichiers de description de génération pour HAGEL utilise 4 entités de base :

- la **task** qui est la description globale d'une activité de génération
- le **template** qui est la description d'un patron de génération
- la **macro** qui est une série d'instruction paramétrée, c'est l'équivalent d'une fonction
- l'**option** qui est une sorte de variable globale utilisée dans une **task**

Une **task** est composée d'au moins un **template** et peut contenir 0 ou plus **macro** et **option**.

C - 1 - 1 Définition des entités syntaxiques

Caractères ignorés : " ", "\t", "\n", "\r", "\f"
Début de commentaire : "/§"
Fin de commentaire : "§/"
Séparateur de début de bloc d'instruction : "§{"
Séparateur de fin de bloc d'instruction : "}§"
Texte de commentaire : Tout ce qui est compris entre les séquences de début et de fin de commentaire
Mots clés :
"task" indique le début de la description d'une activité de génération
"§macro" indique le début de la définition d'une macro
"§template" indique le début de la définition d'un template
"§option" indique la déclaration d'une variable globale
"write" indique le début d'un zone dont le contenu doit être repris tel quel dans la cible, sans interprétation. Ceci est utilisable pour la construction de générateur de générateur.

"interp" indique que le jeu d'instruction qui suit est à interpréter. Ces instructions constituent des structures complexes de tests ou de boucles d'instruction
"call" mot clé pour marquer l'appel d'une macro
" ¤ :" début de la déclaration d'un chemin XPath
" :¤" fin de la déclaration d'un chemin XPath. Un chemin Xpath est défini en accord avec la spécification donnée par le « World Wide Web Consortium » : http://www.w3.org/TR/xpath.
Mots clés pour les instructions de macro (ou de bloc d'interprétation) :
"if"
"while"
"for"
"case"
"switch"
"else"
"default"
"do"
"return"
Caractères séparateurs ou spéciaux :
"="
"\""
" . "
" ; "
" : "
" ("
") "
Caractères classiques :
lettre : "a-z" , "A-Z"
chiffre : "0-9"
Identifiant :
lettre (lettre|chiffre)*
Chaîne de caractères :
- "\"" []"\"" : une chaîne de caractère comprise entre guillemets

C - 1 - 2 Description de la structure grammaticale

Unité de génération :

"task" Identifiant
(
DeclarationOption
|
DeclarationMacro
|
DeclarationTemplate
)*

DeclarationOption :

"§**option" Identifiant** "=" **Chaine de Caractères**

DeclarationOption :

"§**option" Identifiant** "=" **Chaine de Caractères**

DeclarationMacro :

"§**macro" Identifiant** " (" ListeDeParamètres ") "
Block

DeclarationTemplate :

"§**template**" **Identifiant** " (" ListeDeParamètres ") "
"**§{**"
(
Texte
|
InstructionDeTemplate
)*
"}**§**"

InstructionDeTemplate :

"**§**"
(
"**write**" "**(**" Texte "**)§**"
|
"**interp**" "**(**" Instructions "**)§**"
|
"**option**" "**(**" **identifiant** "**)§**"
|
"**call**" "**(**" **identifiant** "**(**" **identifiant** (" , " **identifiant**)* "**)**" "**)§**"
|
identifiant "**(**" ParametresDeTache "**)§**"
)

ParametresDeTache :

(
ChaineDeCaracteres
|
identifiant "**(**" ParametresDeTache "**)§**"
|
identifiant
|
" ¤ **:**" XPath "**:¤**"
)

(" **;**" (
ChaineDeCaracteres
|
identifiant "**(**" ParametresDeTache "**)§**"
|
identifiant
|
" ¤ **:**" XPath "**:¤**"
)
)*

ListeDeParametre :

"**(**"
(
identifiant identifiant
)
("**,**"**identifiant identifiant)***
"**)**"

Bloc :

"**§{**"
(Instructions)*
"**}§**"

Instructions :

Instruction
|
Bloc
|
InstructionDeSwitch
|
InstructionConditionnelle
|
InstructionWhile
|
InstructionBoucle
|
InstructionRetour

InstructionRetour :

"**return**"

InstructionDeSwitch :

"**switch**" "**(**" ParametresDeTache "**)**"
LabelSwitch Bloc
"**}§**"

LabelSwitch :

"**case**" ParametresDeTache " **:**"
|
"**default**" " **:**"

InstructionConditionnelle :

"**if**" "**(**" ParametresDeTache "**)**"
Bloc
["**else**" Bloc]

InstructionWhile :

"**while**" "**(**" ParametresDeTache "**)**"
Bloc

InstructionFor :

"**for**" "**(**" ParametresDeTache "**)**"
Bloc

Instruction :

identifiant "**(**" ParametresDeTache "**)**"
|
"**call**" "**(**" **identifiant** "**(**" **identifiant** (" , " **identifiant**)* "**)**" "**)§**"

C - 2 Structure du fichier XML de description des tâches connues

Le fichier XML permettant de définir les tâches connues par HAGEL contient une liste d'association entre des alias et les noms exacts des classes définissant les tâches utilisées.
La Figure A. 5 illustre la structure de ce fichier :

```
<HAGEL>
<TASK name="append" class="cdi.hagel.gener.task.elem.Append"/>
<TASK name="option" class="cdi.hagel.gener.task.elem.GetOption"/>
<TASK name="get" class="cdi.hagel.gener.task.elem.GetXMLValue"/>
<TASK name="interp" class="cdi.hagel.gener.task.elem.Interpreted"/>
<TASK name="for" class="cdi.hagel.gener.task.test.ForLoop"/>
<TASK name="newString" class="cdi.hagel.gener.task.elem.CreateNewString"/>
<TASK name="newInt" class="cdi.hagel.gener.task.elem.CreateNewInteger"/>
<TASK name="add" class="cdi.hagel.gener.task.elem.Add"/>
<TASK name="valueOf" class="cdi.hagel.gener.task.elem.GetStringValue"/>
< TASK name="newBuffer" class="cdi.hagel.gener.task.file.CreateNewBuffer"/>
<TASK name="setCurrentBuffer" class="cdi.hagel.gener.task.file.SetCurrentBuffer"/>
<TASK name="date" class="cdi.hagel.gener.task.elem.GetDate"/>
<TASK name="rama" class="cdi.hagel.gener.task.elem.RamaTask"/>
</HAGEL>
```

Figure A. 5: Exemple du fichier *XMLTask.xml*

L'attribut **name** correspond au mot clé utilisé dans la description des patrons de génération.

C - 3 Exemple d'utilisation de HAGEL

Nous donnons ici un exemple d'utilisation de HAGEL.

La Figure A. 6 présente l'exemple d'un patron simple.

```
task demo

§template essai(DATA dat) §{
Voici un essai de §get(¤: /dat/AUTEUR/@nom :¤)§

Bonjour lecteur. Es tu en vacance le §date()§

}§
```

Figure A. 6: Exemple de patron de génération

La Figure A. 7 présente un exemple de données au format XML

```
< DATA >
<AUTEUR nom=« Dupont » adresse=« Grenoble »/>
</DATA>
```

Figure A. 7: Exemple de données au format XML

La Figure A. 8 présente le résultat de l'utilisation de HAGEL avec ces fichiers :

```
Voici un essai de Dupont

Bonjour lecteur. Es tu en vacances le 13 july 2003
```

Figure A. 8: Résultat de l'exploitation de HAGEL

Annexe B

Structuration des modèles

Annexe B Structuration des modèles

A - Généralités

A - 1 - 1 Les fichiers conteneurs

MAEL manipule deux types d'entités, des représentations acontextuelles de la connaissance (les fichier *.mo) et des représentation contextualisées de cette connaissance (les fichiers *.pro). Toutes deux sont représentées sous la forme de fichiers informatique qui ont une structure commune.

Les modèles sont stockés sous la forme de fichiers informatiques qui rassemblent la totalité de l'information de description du modèle y compris les éléments de documentations graphique ou textuelle dont l'utilisateur désire disposer pour la compréhension du modèle. Il s'agit de fichiers compressés au format zip.

La structure interne de ces fichiers est très simple :

- le répertoire de base contient la description du modèle lui-même, dans un fichier au format XML, ainsi que les documents rattachés au modèle.
- Un sous répertoire (meta-inf) contient un fichier de description de l'information stockée dans le modèle : attributes.attr.

Les fichiers attachés en documentation au modèle peuvent être sous n'importe quel format image, texte ou son.

La description du modèle dans un fichier XML fait l'objet d'une description plus approfondie plus loin.

Constitution du fichier **/meta-inf/attributes.attr**

Le fichier de description de l'information sur le modèle est un fichier au format XML. Nous présentons ici un exemple du fichier :

```
<ATTRLIST name="component.info">
<ATTR name="model" value="model.mo"/>
</ATTRLIST>
```

La racine du document est constituée par un nœud du type **ATTRLIST**, le seul attribut de obligatoire pour le nœud est l'attribut **name="component.info".**

Les enfants de ce nœud doivent être du type **ATTR** et ils ont deux attributs qui sont un nom et une valeur.

Afin de pouvoir lire les modèles de MAEL, il est nécessaire de disposer de l'attribut

<ATTR name="model" value="model.mo"/>, par cette information, on est alors capable de charger le modèle. Cet attribut indique, lors de la lecture d'un modèle que le fichier XML de description du **model** est le fichier **model.mo**.
Ainsi le fichier de description de l'information du modèle contient au moins l'information permettant de trouver dans l'archive la description XML du modèle.

Remarques : le fichier compressé du modèle contient toutes les informations nécessaires pour la visualisation du modèle par le biais de l'IHM de MAEL. MAEL suppose que les fichiers de documentation, de description des algorithmes ou des fonctions sont à la racine de l'archive. Toutefois, rien n'empêche que cela ne soit pas le cas.

A - 2 Les fonctions reconnues dans le langage

Le langage de MAEL est extensible, en particulier au niveau de la nature des fonctions. Cela permet d'ajouter de nouvelles fonctions « built-in », toutefois à l'heure actuelle seul un certain nombre est reconnu.

A - 2 - 1 Une fonction particulière : la fonction der

Pour obtenir la dérivée temporelle d'une variable ou d'une expression, on utilisera la fonction der, avec comme seul paramètre la variable ou l'expression à dériver.

```
Ul=L*der(Il)
```

Il est également possible d'utiliser la fonction **derf** pour obtenir la dérivée formelle d'une expression par rapport à une variable donnée :

```
A=derf(a*pow(x,2),x) est traitée par A= 2*a*x
```

A - 2 - 2 La variable p

La variable **p** est accessible par le mot clé **pi.**

A - 2 - 3 Les fonctions mathématiques classiques

Nom de la fonction	Ce qu'elle fait	Exemple d'utilisation
cos(x)*	Cosinus d'une expression. Retourne un nombre compris entre -1 et 1	
sin(x)*	Sinus d'une expression, retourne un nombre compris entre -1 et 1	
tan(x)*	Retourne un nombre compris entre -1 et 1	
cosh(x)*	Retourne le cosinus hyperbolique de x	
sinh(x)*	Retourne le sinus hyperbolique de x	
tanh(x)*	Retourne la tangente hyperbolique de x	
exp(x)*	Retourne l'exponentielle de x	
log(x)*	Retourne le logarithme népérien de x	
exp(x)*	Retourne l'exponentielle de x	
sqrt(x)	Retourne la racine carrée de x	
pow(x)	Retourne un nombre élevé à la puissance n	Pow(x,2) *Retourne x élevé au carré*
arccos(x)	Arc-cosinus, retourne un nombre compris entre –**p**/2 et + **p**/2.	
arcsin(x)	Arc-sinus retourne un nombre compris entre 0 et **p**	
arctan(x)	Arc-tangente	
arccosh(x)	Arc cosinus hyperbolique	
arcsinh(x)	Arc-sinus hyperbolique	
arcctanh(x)	Arc-tangente hyperbolique	

*Remarque : les fonctions marquées d'un astérisques acceptent des expressions ou des variables dont le type est complexe.

A - 2 - 4 Les fonctions avancées

Nom de la fonction	Ce qu'elle fait	Exemple d'utilisation
Interp1D(x,tab1,tab2)	Réalise une interpolation une dimension entre deux tableaux de valeurs. Ces valeurs sont définies dans des tableaux et sont accessibles par colonnes	Interp1D(3.0, A.X, A,Y) Interp1D(x , ColonneX , ColonneY) Retourne la valeur interpolée pour x, en réalisant l'interpolation sur les vecteur de données définis dans les colonnes X et Y du tableau A.
Interp2D(x,y,tab1,tab2,tab3)	Réalise une interpolation 2D entre les donnés en entrée	Interp2D(3.0 , 4.0, A.X ,A.Y , A.Z) Interp2D(x,y, ColonneX, ColonneY, ColonneZ) Retourne la valeur interpolée pour le couple (x,y) en réalisant l'interpolation sur les données du tableau A.
Map(x , tab1 , tab2)	Effectue un mapping de données, c'est-à-dire qu'elle retourne la valeur équivalente à celle que l'on a donnée	Map(0.15 , Diametre.Nu , Diametre.Isole) Map(x, col_1 , col_2) Retourne la valeur correspondante à la donnée x prise dans la colonne 2

A - 2 - 5 Les fonctions complexes

Puisque MAEL permet de définir des grandeurs de type complexes, il est naturel de prévoir un certain nombre de fonction permettant la manipulation de variables de ce type.

Nom de la fonction	Ce qu'elle fait	Exemple d'utilisation
Arg(z)	Donne l'argument d'un nombre complexe	Arg(3+5*I) Retourne Arctan2(5/3)
Mod(z)	Donne l'argument d'un nombre complexe	Mod(3+5*I) Retourne sqrt(9+25)=sqrt(34)
Im(z)	Donne la partie imaginaire d'un complexe	Im(3+5*I) Retourne 5
Re(z)	Donne la partie réelle d'un complexe	Re(3+5*I) Retourne 3
Conj(z)	Donne le conjugué d'un complexe	a=Conj(3+I*5) retourne 3-5*I
CompMA(a,b)	Crée un nombre complexe à partir de la valeur de son module et de son argument	a=CompMA(1,0) crée le complexe de module 1 et d'argument 0
CompRI(a,b)	Crée un nombre complexe à partir des parties réelle et imaginaire	a=CompRI(3,5) crée le complexe : 3+5*I

A - 3 La syntaxe pour les équations

Nous donnons ici la description de la syntaxe pour les équations des modèles. Il s'agit d'une syntaxe proche de celle spécifiée par la norme ANSI-C.

Certains éléments sont rajoutés, afin notamment de prendre en compte les « accès » aux variables des éléments ou encore aux éléments des tableaux.

Pour la description des structures grammaticales, les conventions suivantes sont utilisées :

[ ] : signale que la structure enclose est optionnelle

{ } : signale que la structure enclose n'est pas présente ou bien répétée un nombre infini de fois.

| : offre le choix entre deux structures

Nous donnons ici la description des TOKEN (mots) des équations :

Les symboles :

Les opérateurs classiques :

"+": pour les additions, ou pour signer une valeurs

"-": pour les soustractions ou pour signer les valeurs

"*": pour les multiplications

"/": pour les divisions

L'opérateur d'égalité :

"=": égalité de deux expressions, définition d'intervalles ou de points de départ…

Les séparateurs :

",": sépare les éléments de tables, les paramètres des fonctions.

"(" ")": les parenthèses, pour les paramètres de fonctions ou les priorités des opérations

"[" "]": pour la definition de tables ou d'intervalles.

Les noms de fonctions ou de variables : (identificateurs)

IDENT : LETTRE { LETTRE | CHIFFRE }
LETTRE : "_" | "a" - "z" | "A" - "Z"
CHIFFRE : "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"| "0"
ENTIER_NON_SIGNE: CHIFFRE {CHIFFRE}

NOMBRE_NON_SIGNE : ENTIER_NON_SIGNE ["." [ENTIER_NON_SIGNE]]
[("e" | "E") ["+"|"-"] ENTIER_NON_SIGNE]

Les structures grammaticales:

L'égalité (Equation)

Expression "=" Expression

Exemple

```
(a+b)*c=cos(om*t)
ou
f(a,b+c)=0
```

Expression

Multiplication { ("+" | "-") Multiplication }

Exemple

```
(a+b)-(t*m)
ou
L1*g+p-(t+m)
```

Multiplication

Unitaire { ("*" | "/") Unitaire }

Exemple

```
A*h*l
ou
3*m
ou
pi/2
```

Unitaire

"(" Expression ")"
|
(("+"|"-") Unitaire)
|
Primaire

Exemple

```
(a+b)
ou
+15.4
ou
–f(t)
```

Primaire

NOMBRE_NON_SIGNE
|
Tableau
|
AppelDeFonction
|
Nom

Exemple

```
1
ou
1.45
ou
a
ou
cos(om*t)
ou
interp1D([1,2,3],[t,h,o],2)
```

Nom: utilisé pour les noms de variables, les références à des colonnes des tableaux, des variables des sous-modèles

IDENT {"." IDENT}

Exemple

```
a
ou
longueur_A
ou
flux
ou
ELEM.u
```

Tableau

"[" ListeExpression "]"

Exemple

```
[1,2]
ou
[a+b,g(h)]
```

AppelDeFonction

Nom "(" ListeExpression ")"

Exemple

```
Cos(a)
ou
f(a+b,j*m)
```

ListeExpression

Expression { "," Expression }

Exemple

```
a+b , cos(t) , k*l
ou
a
```

A - 4 Les unités

Il est possible d'attribuer des unités à chacune des dimensions physiques qui interviennent dans la description du système.

Nous donnons ici la syntaxe générale pour la description des unités. Elles peuvent être décrites sous la forme d'un produit d'unités élémentaires qui sont données plus bas. Chacune de ces unités.

[Réel " *"] (unite^un_nombre) [" *" (unite^un_nombre)]

Par exemple une force pourrait s'exprimer sous la forme du produit d'une accélération par une masse, une accélération étant elle-même définie comme une distance divisée par le carré des secondes.

Ainsi, une force en Newton s'exprime sous la forme suivante, dans les unités fondamentales du système international :

N = kg*m*s^-2

Outre, une certaine quantité d'unité fondamentale, il est possible d'attribuer des multiples classiques à ces unités, pour obtenir des kilomètres (km) par exemple.

Il existe deux unités particulières utiles pour la description des variables. Pour une grandeur qui est un rapport de transformation par exemple, l'unité **su** (sans unité) est préconisée, lorsque l'on ne connaît pas l'unité d'une grandeur, il est possible de lui attribuer l'unité **usi** (unité du système international) qui assure une compatibilité avec toutes les unités utilisées dans le modèle.

A - 4 - 1 Unités reconnues

Abbréviation reconnue	Nom de l'unité	Passage au système international
m	Mètre : unité de longueur	m
kg	Kilogramme : unité de masse	kg
A	Ampère : unité de mesure de courant	a
s	Seconde : unité de mesure du temps	s
mol	Mole : unité de mesure de quantité d'atome	mol
cd	Candela : unité de mesure de la luminosité	cd
K	Kelvin : unité de mesure de la température	K
usi	Unité spéciale pour assurer la compatibilité des expressions	usi

su	Sans unité pour les coefficients de transformation	1
C	Coulomb : unité de mesure de charge électrique	s*A
N	Newton : unité de mesure des forces	m*kg*s^-2
sr	Steradian : unité de mesure des angles spatiaux	1
Hz	Hertz : unité de mesure de la fréquence	s^-1
Pa	Pascal : unité de mesure de la pression	N*m^-2
J	Joule : unité de mesure de l'énergie	m^2*kg*s^-2
W	Watt : unité de mesure de la puissance	m^-2*kg*s^-3
V	Volt : unité de mesure de la tension	m^2*kg*s^-3*A^-1
ohm	Omh : unité de mesure des résistances	V*A^-1
F	Farad : unité de mesure de capacité	C*V^-1
S	Siemens :unité de mesure des perméance	A*V^-1
Wb	Weber : unité de mesure des flux magnétiques	V*s
T	Tesla : unité de mesure d'induction	Wb*m^-2
H	Henry : unité de mesure des inductances	Wb*A^-1
dC	Degré Celsius : unité de mesure de la température	K
lm	Lumen : densité lumineuse	cd
lx	Lux : densite de flux lumineux	cd*m^-2
Bq	Becquerel	s^-1
Gy	Gray	J*kg^-1
Sv	Sievert : unité de mesure de la radioactivité	J*kg^-1

*A - 4 - 2 **Coefficients multiplicateurs***

Multiplicateur reconnu	Nom reconnu	Coefficient
y	yocto	1E-24
z	zepto	1E-21
a	atto	1E-18
f	femto	1E-15
p	pico	1E-12
n	nano	1E-9
u	micro	1E-6
m	milli	1E-3
c	centi	1E-2
d	deci	1E-1
da	deca	10
h	hecto	100
k	kilo	1E3
M	mega	1E6
G	giga	1E9

T	Tera	1E12
P	Peta	1E15
E	Exa	1E18
Z	Zetta	1E21
Y	Yotta	1E24

B - Les modèles non-orientés : les boules

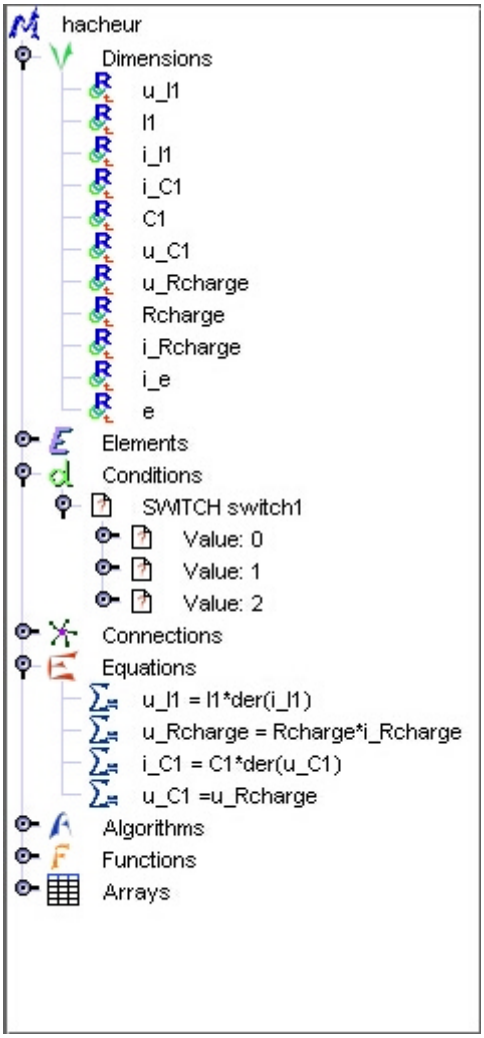
B - 1 Description XML du Modèle

Comme nous l'avons plus haut, les modèles sont stockés dans un fichier XML. Ce format de stockage offre la possibilité de structurer l'information sous la forme d'un arbre.
Afin de mieux comprendre la structure du fichier XML, nous donnons ici la structuration des modèles Objets que nous utilisons.

B - 1 - 1 L'arbre d'un modèle :

Nous ne décrivons ici que la structure générale d'un modèle. Les éléments de cette structure sont détaillés plus bas.

L'IHM de MAEL présente une vue de l'arbre d'un modèle tel que nous les avons développés.



Nous y reconnaissons des structures de bases pour la description de l'arbre.
Un modèle analytique est composé de Variables qui sont mises en relations par des Relations.

Les Variables sont l'équivalent des variables d'instances de la programmation orientés objet, et les Relations sont l'équivalent des méthodes.

Nous réservons un statut particulier pour les fonctions et les tableaux qui sont des données du modèle. Il s'agit en quelque de sorte des variables de classes, c'est-à-dire qu'elle ne sont pas propre à une instance de modèle.

Les variables sont les éléments de la modélisation qui sont mis en relation par la description du modèle. Les « variables » peuvent être des Dimensions, ce sont alors des grandeurs physiques qui interviennent dans la description du dispositif (une longueur, une résistivité, une résistance, un courant…), mais les variables peuvent aussi être des éléments de la modélisation : il s'agit alors de sous-modèles utilisés pour la description d'un modèle complexe par connexions des variables des sous-modèles utilisés.

Les relations sont les éléments de description du dispositif qui permettent de décrire les liens entre les différentes variables. Les équations sont des relations évidentes, mais les algorithmes sont eux aussi des relations puisqu'ils décrivent les relations entre les grandeurs. La description de dispositif par connexions de sous-modèles se fait par le biais d'un autre de relation : les « Connections ».

Enfin, dernier élément que l'on rencontre dans la description d'un modèle, les conditions : il s'agit de l'élément de base pour la description d'un système hybride. Une condition est un test réalisé sur un entier ou un booléen qui détermine l'activation d'un jeu de relations ou d'un autre.

Ainsi la structure générale d'un modèle est la suivante :
MODEL :
Est constitué de « variables » :

- *DIMENSION*
- *ELEMENT*

D'« utilitaires » de description

- *FUNCTION*
  - *DIMENSION*
  - *PORT*
- *ARRAY*
  - *COLUMN*
    - *ELEM*

De « relations » :

- *EQUATION*
- *ALGORITHM*
  - *DIMENSION*
  - *NUMERICALP*
- *CONNECTION*
  - *PORT*

De conditions pour les descriptions de modèles d'états hybrides :

- *CONDITION*
  Elle-même constituée d'une liste d'états (suivants les valeurs d'un test)
  - *STATE*

Dans ces états on retrouve des relations et des conditions qui permettent la description du dispositif.

- *EQUATION*
- *ALGORITHM*
  - *DIMENSION*
  - *NUMERICALP*
- *CONNECTION*
  - *PORT*
- *CONDITION*
  - …

B - 1 - 2 <u>Les éléments de l'arbre</u>

Un arbre est constitué de nœuds, qui ont, ou non, des fils. Lorsqu'un nœud n'a pas de fils, on dit qu'il est terminal.

Dans cette présentation des éléments de description du modèle, nous ne décrivons que les éléments de bases standard de la description telle qu'elle acceptée dans MAEL. Il faut savoir que le formalisme XML permet une extensibilité quasiment infinie autour d'un schéma de base. C'est cette capacité qui est utilisée pour permettre l'ajout d'informations spécifiques par l'utilisateur.

Nous ne décrivons donc ici que les attributs des différents éléments imposés ou prévu par le système. Les attributs décrits peuvent être obligatoires ou non et éventuellement avoir une valeur par défaut. Il n'est pas possible de modifier leur nom et les valeurs possibles peuvent être définies dans une liste.

Dans la suite de ce document, les différents éléments de l'arbre sont décrits, les informations qui leur sont attachées peuvent ou non être obligatoire. Dans ce cas, elles seront marquées d'un astérisque (*).

B - 1 - 3 L'élément MODEL

Le modèle est la racine d'une description, il englobe la description d'un dispositif. Un modèle peut hériter d'un autre modèle, sa clause d'héritage est un attribut.

Attributs	Signification	Domaine des valeurs
name*	Le nom du modèle	Chaîne de caractères. Contient tous les caractères alphanumériques classique (sauf accent) et commence par une lettre.
comment	Un commentaire décrivant la nature du modèle	Chaîne de caractères
extend	Le modèle dont celui-ci hérite. La syntaxe du nom donné doit suivre la convention de nommage donnée plus bas	Chaîne de caractères
file	La liste de fichiers attachés à cet élément pour la documentation du modèle. Les fichiers sont séparés par des « ; ».	Chaîne de caractères

L'élément MODEL est l'élément racine de la description d'un modèle, à ce titre il peut avoir dans parmi la liste de ses enfants tous les éléments de description excepté les états des conditions.

Exemple de syntaxe :

```
<MODEL name="rlc" comment="description d'un circuit RLC série" file="schema.bmp">
.
.
.
</MODEL>
```

3 - a) L'élément DIMENSION

L'élément DIMENSION représente les grandeurs qui apparaissent dans la description du dispositif. Les DIMENSIONs sont les points d'entrée offerts par le modèle à l'extérieur.
Lors de la description d'un dispositif par le biais d'un grand nombre de relations, il peut être intéressant de limiter la visibilité de certaines dimensions. Il n'est pas intéressant de « voir » toutes les grandeurs de la description depuis l'extérieur, il est donc possible de limiter cette visibilité (on retrouve ici l'équivalent des notions de variables privée ou publiques de la Programmation Orientée Objet).
Ces grandeurs sont typées, réelles, complexes ou entières et afin de permettre un contrôle d'homogénéité des relations construites, il est possible de donner une unité à chaque grandeur. La syntaxe pour ces unités est explicitée dans la suite du document.

Attributs	Signification	Domaine de valeur
name*	Le nom de la dimension	Chaîne de caractères
comment	Un commentaire décrivant signification de la variable	Chaîne de caractères
unit*	L'unité de la grandeur concernée. La syntaxe des unités est donnée plus bas	Suivant la convention donnée plus haut. Par défaut usi ; Exemple m, kg ou m^2
datatype*	Le type de la variable, utilisée pour la gestion des grandeurs complexes	Par défaut {real, integer, complex}
file	Le lien vers un fichier de documentation dans lequel on trouvera des explications sur cette grandeur	Chaîne de caractères
visibility*	Détermine la visibilité de la variable. Intern pour interdire la visibilité à l'extérieur, extern dans l'autre cas.	Par défaut inter {intern ou extern]

Les éléments de type DIMENSION sont des éléments terminaux de l'arbre, ils n'ont pas d'enfant.

Exemple de syntaxe :

```
<DIMENSION name="R" comment="valeur de la résistance" unit="ohm" datatype="real"/>
```

3 - b) L'élément ELEMENT

Un ELEMENT est un sous modèle utilisé dans le cas de la description d'un modèle complexe par agrégations de modèles élémentaire.
Un ELEMENT est caractérisé par un type, le modèle auquel il fait référence et un nom qui permet de l'identifier de façon unique dans le système. Ces deux informations sont exploitées lors de la construction d'un PROCESS.

Attributs	Signification	Domaine de valeur
name*	Le nom de l'élément	Chaine de caractères
comment	Un commentaire décrivant la place de cet élément dans la modélisation	Chaine de caractères
datatype*	Le type de l'élément considéré. Il suit la même convention de nommage que la clause extend du model.	Chaine de caractère, permettant de retrouver la définition de l'élément (*electrical.component.R*)
file	Un lien vers un fichier de documentation pour cet élément	Liste de fichiers séparés par des « ; »

Les éléments peuvent être réutilisés dans la description du dispositif par le biais des EQUATION mais aussi par le biais des CONNECTION.

Comme les DIMENSION, les ELEMENT sont des nœuds terminaux de la description de l'arbre, ils n'ont pas d'enfant.

Exemple de syntaxe :

```
<ELEMENT name="R" comment="resistance du modele" datatype="elec.R"/>
```

3 - c) L'élément EQUATION

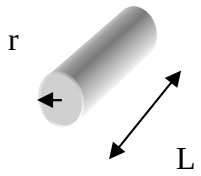
L'équation est l'élément central de la description d'un dispositif physique sous une forme mathématique. La syntaxe des équations manipulées est une syntaxe très proche de la Norme ANSI-C.

Les équations mettent en relation les dimensions du modèle sous une forme naturellement non orientée. C'est-à-dire que la syntaxe même des équations est telle qu'aucune orientation n'est donnée a priori.

Comme les méthodes de la P.O.O, les équations peuvent être surchargées. Lors de la construction d'un modèle qui hérite d'un autre modèle il est possible de remplacer une équation par une autre.

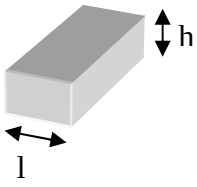
Par exemple, lors de la définition d'un conducteur, il sera possible de remplacer la définition de la résistance du conducteur en fonction de la géométrie de la section du conducteur.

```
Model : Conduc_cyl
Dimensions : rho, L, r, S, R
Equations :
R = rho*L/S
S=pi*pow(r,2)
```



On peut retrouver un conducteur de section carré dont le modèle serait alors le suivant :

```
Model : Conduc_Rect extends Conduc_cyl
Dimensions : l, h
Equations :
S=l*h remplace S=pi*pow(r,2)
```



La syntaxe des équations est détaillée plus bas ainsi que les fonctions propres à MAEL pour la description des dispositifs.

Attributs	Signification	Domaine de valeur
Expression*	L'expression de l'équation en respectant la syntaxe détaillée plus bas	Suivant la convention d'écriture des équations donnée plus haut
comment	Un commentaire donnant la signification de l'équation	Chaine de caractère
replace	L'intitulé exact de l'équation à remplacer.	Chaine de caractère
file	Un lien vers un fichier de documentation pour cette équation	Chaine de caractère

Les EQUATIONS sont des éléments terminaux de l'arbre, ils n'ont pas d'enfant.

Exemple de syntaxe :

```
<EQUATION expression="U=R*I" comment="loi d'ohm"/>
```

3 - d) L'élément ALGORITHM

Un ALGORITHM est similaire à une équation car il définit les relations entre deux jeux de variables, ses entrées et ses sorties. Si un algorithme est naturellement orienté, un sens de calcul est imposé par l'explicitation des entrées et des sorties, il reste possible d'inverser la signification des grandeurs.
Un ALGORITHM est décrit dans un fichier distinct et la syntaxe utilisée est similaire à celle du langage C. Ce fichier est inclus dans l'archive du modèle de la même façon qu'un élément de documentation graphique ou textuelle.
On distingue trois types d'algorithmes, suivant la nature des variations des paramètres de sortie décrites. Ils peuvent être **discret** ou **continu_0** ou encore **continu_1**. Par défaut, nous les considérerons comme discret et la localisation de leur évaluation dans un processus de résolution numérique sera fixée par cette condition.
Les ALGORITHM sont des fonctions de $\Re_n$ dans $\Re_m$. Les définitions des trois types d'ALGORITHM identifiés s'en suivent.

Un ALGORITHM est **continu_0** si la fonction qu'il représente est continue de classe 0 (les dérivées d'ordre 1 ne sont pas continues).
Un ALGORITHM est **continu_1** si la fonction qu'il représente est continue de classe 1 (les dérivées d'ordre 1 sont continues).
Enfin, un ALGORITHM est **discret** si la fonction représentée n'est pas continue de classe 0. La majorité des algorithmes sont de cette sorte. Dans la plupart des cas, un test utilisé dans une description provoque une discontinuité des sorties. Ainsi, en cas d'hésitation, il est conseillé de déclarer l'ALGORITHM dans le type discret.

Attributs	Signification	Domaine de valeur
name*	Le nom de l'algorithme	Chaine de caractères
comment	Un commentaire donnant des explications sur la fonction de l'algorithme	Chaine de caractère
datatype*	Le type de l'algorithme.	Par défaut : discret {discret, continu_0, continu_1}
file*	Contient au moins la référence du fichier source de l'algorithme.	Le nom du fichier contenant l'algo

Les fils d'un ALGORITHM : DIMENSION et NUMERICALP

Les Algorithmes peuvent utiliser des variables internes ou externes. Dans un cas comme dans l'autre, ceci est spécifié dans la liste des dimensions de l'algorithme. Dès lors qu'une variable est donnée comme **extern,** on considère qu'elle est partagée par l'ensemble du modèle. Dans l'autre cas, les grandeurs sont internes et ne sont pas connues par hors de l'algorithme.
Enfin, les dimensions externes de l'algorithmes doivent être définies comme des entrées (input) ou des sorties (output). Dans le cas où l'algorithme n'est pas inversé, les grandeurs d'entrées doivent être évaluées avant l'appel à l'algorithme.
Un algorithme peut être paramétré par un certain nombre de paramètres qui ne sont pas des grandeurs physiques. Il s'agit, par exemple, d'une précision numérique, d'un critère d'arret, d'un paramètre numérique qui est utilisé dans la description de l'algorithme.
Les attributs des dimensions d'un algorithme :

Attributs	Signification	Domaine de valeur
name*	Le nom de la dimension	Chaine de caractère
comment	Un commentaire décrivant signification de la variable	Chaine de caractère
io_status*	Détermine si la grandeur est une entrée ou une sortie.	Par défaut : input {input, outpu}
file	Le lien vers un fichier de documentation dans lequel on trouvera des explications sur cette grandeur	Chaine de caractère
visibility*	Détermine la visibilité de la variable. Intern pour interdire la visibilité à l'extérieur, extern dans l'autre cas.	Par défaut : extern {intern, extern}

De nombreux algorithmes utilisent des paramètres numériques, ce sont des grandeurs qui ne sont pas réelles pour le dispositif. Il s'agit d'artifices de calculs pour les résolutions telles que des pas minimums ou des erreurs.

Ils sont identifiés que par leur nom et une valeur par défaut.

Les attributs des paramètres numériques d'un algorithme :

Attributs	Signification	Domaine de valeur
name*	Le nom du paramètre	Chaine de caractère
comment	Un commentaire pour ce paramètre	Chaine de caractère
default	La valeur par défaut du paramètre	Les réels

Exemple de syntaxe :

```
<ALGORITHM name="etat_hacheur" comment="fixe l'etat du hacheur" file="etat_hacheur.alg" datatype="discret">
.<DIMENSION name="switch1" comment="choix de la config" visibility="extern" io_status="ouput"/>
.
.          <NUMERICALP name="prec_pdi" />
</ALGORITHM>
```

NB : L'inversion d'un algorithme reste un choix lourd de conséquence pour la stabilité numérique du modèle.

3 - e) L'élément CONNECTION

Les connections sont les éléments de bases utilisés pour la composition des modèles. Elles mettent en relation des DIMENSIONS des modèles considérés par le biais de sommes ou d'égalités.

Dans le cas où la connexion est une somme, une relation est ajoutée entre les variables concernées : la somme de ces variables est égale à 0. C'est le cas par exemple pour les courants à un nœud de circuit électrique.

Dans le cas où la relation est IDENTIQUE, on impose à toutes les variables connectées de prendre la même valeur. Lors du traitement formel du modèle, un jeu d'égalité est ajouté aux équations du modèle.

Une CONNECTION est une « relation » et à ce titre on peut la trouver aux mêmes endroits que des ALGORITHM ou des EQUATION.

Attributs	Signification	Domaine de valeur
name*	Un identifiant unique pour la connexion	Chaine de caractère
comment	Un commentaire sur la signification de cette connexion	Chaine de caractère
datatype*	IDENTITY ou SUM=0, pour déterminer le type de relation induite par cette connexion	Par défaut : IDENTITY {IDENTITY, SUM=0}
file	Un schéma ou un texte de documentation	Chaine de caractère

Les CONNECTION doivent être faites entre des PORTS des ELEMENT utilisés dans le modèle. Le nœud associé a donc des enfants qui sont ces ports.

Ces PORT n'ont qu'un seul attribut obligatoire qui est leur nom. La syntaxe de ce nom est la suivante :

<NOM de l'ELEMENT>.<NOM de la DIMENSION>

Le premier identifiant rencontré est le nom de l'élément, et le second est celui de variable à connecter.

Par exemple, pour un modèle de Résistance, ayant comme variable U, R, I, utilisé dans un modèle plu complexe, si l'élément utilisée porte le nom R1 ; on accède au courant par la notation **R1.I**.

Exemple de syntaxe :

```
<CONNECTION name="nœud_1" comment="une source et une resistance en serie"
datatype="SUM=0">
<PORT name="R1.I"/>
<PORT name="Source.I"/>
.
</CONNECTION>
```

NB : il y a un risque en utilisant des connections d'éléments. Il peut manquer des équations qui permettent la description du modèle. C'est par exemple le cas lors de la description d'un circuit électrique, il est nécessaire de rajouter les équations des mailles.

3 - f) L'élément CONDITION

La condition est l'entité de base pour la description d'un modèle hybride. Elle permet de donner une description d'un système multi-état. Une condition est un switch qui peut être de deux types, soit entier, auquel cas l'état actif est fixé par la valeur prise par un entier, soit par un booléen, qui peut alors représenter deux états : vrai ou faux.

Les conditions ne sont pas des nœuds terminaux, elles ont des enfants du type STATE. Ces enfants caractérisent l'activation d'un état en fonction de la valeur de la condition qui les contient.

Les attributs de la CONDITION

Attributs	Signification	Domaine de valeur
name*	Un identifiant unique pour la condition	Chaine de caractère
comment	Un commentaire sur la signification de cette condition	Chaine de caractère
datatype*	integer ou boolean, pour déterminer le type de tests à faite pour cette condition.	Par défaut : boolean {Integer, boolean}
file	Un schéma ou un texte de documentation	Chaine de caractère

Les attributs des STATE

Attributs	Signification	Domaine de valeur
value*	La valeur que la condition doit prendre pour activer cet état	Chaine de caractère
comment	Un commentaire sur la signification de cet état	Chaine de caractère
file	Un schéma ou un texte de documentation	Chaine de caractère

Un état contient alors un jeu de Relations telles que décrites plus haut. Lorsqu'un état est validé, il « active » le jeu de relations qu'il contient mais aussi les tests à faire pour les éventuelles conditions qu'il peut lui-même contenir.

Exemple de syntaxe :

```
<CONDITION name="test_1" comment="" datatype="boolean">
<STATE value="true">
        <EQUATION expression="a=b"/>
</STATE>
<STATE value="false">
        <EQUATION expression="a=-b"/>
</STATE>
</CONDITION>
```

3 - g) L'élément FUNCTION

Une fonction est un artifice mathématique utilisable dans tous les états. Par exemple la fonction f(x)=x^2+3*x, peut être définie dans un modèle pour une utilisation dans l'ensemble du modèle.
Les fonctions sont définies de la même façon que les algorithmes, elles sont écrites dans un fichier annexe qui est ajouté à l'archive du modèle.

La déclaration d'une fonction doit impérativement contenir une clause de retour de valeur. Contrairement à un algorithme, une fonction est utilisable à plusieurs reprises dans la description d'un modèle, il ne s'agit pas d'une relation mais d'un élément utilisable dans une relation. Ainsi, on peut définir la fonction f(x) et l'utiliser dans la déclaration d'équations et d'algorithmes.
Une fonction peut utiliser des éléments du modèle (variables) dans sa déclaration, les algorithmes de séquencement doivent alors prendre en compte l'initialisation de ces variables avant tout appel à la fonction.

Une fonction ayant une valeur de retour, celle-ci doit avoir une unité (afin de permettre les tests d'homogénéité).

Attributs	Signification	Domaine de valeur
name*	Un identifiant unique pour cette fonction	Chaine de caractères
comment	Un commentaire sur la signification de cette fonction	Chaine de caractères
unit*	L'unité de la veleur de retour	usi Suivant la convention donnée plus haut
file*	Un schéma ou un texte de documentation	Chaine de caractères

La signature d'une fonction est définie par une liste de VARIABLE, il s'agit de la liste de toutes les variables de la fonction. Ils doivent être définis par leur nom et leur position dans la signature. Afin de permettre les contrôles d'homogénéité ces VARIABLE doivent également donner l'unité de la variable qu'ils attendent.

Attributs	Signification	Domaine de valeur
name*	Le nom de la variable dans la description de la fonction	Chaine de caractères
comment	Un commentaire sur la signification de ce port	Chaine de caractères
unit*	L'unité de la variable à connecter au port.	Par défaut : usi Suivant la convention donnée plus haut
file	Un schéma ou un texte de documentation	Chaine de caractères
order*	Donne la position de la variable dans la signature de la fonction. -1 pour indiquer une variable globale	Par défaut : -1 Entier : {-1,0,…n-1}

Exemple de syntaxe : pour la fonction f(x) = x^2+3*x

Le fichier de description de la fonction doit contenir l'instruction suivante (on utilise la même syntaxe que pour les descriptions des algorithmes : du C). Le fichier f.fun contient :

```
return pow(x,2)+3*x ;
```

Et la fonction est décrite de la façon suivante dans le fichier XML :

```
<FUNCTION name="f" comment="" unit="usi" file="f.fun">
<VARIABLE name="x" unit="usi" order="0"/>
</FUNCTION>
```

3 - h) L'élément ARRAY

Un tableau peut parfois être utile pour la description de modèle. Par exemple, pour réaliser un mapping de variable (diamètre de fil nu <-> diamètre de fil isolé), ou une interpolation (courbe B(H), …), il est confortable de définir des tableaux de données numériques.

L'élément ARRAY apporte cette souplesse. Il est constitué de COLUMN elle-même constituée d'ELEM.

Un tableau a un nom qui doit être un identifiant unique et les attributs classiques de la description des éléments de modélisation.
Les COLUMN ont-elles aussi un nom. Les ELEM ont une valeur et un indice de position dans la colonne.
Pour la description d'un tableau : ARRAY

Attributs	Signification	Domaine de valeur
name*	Un identifiant unique pour ce tableau	Chaine de caractères
comment	Un commentaire sur la signification de ce tableau	Chaine de caractères
file	Pour la documentation du tableau	Chaine de caractères

La description d'une colonne : COLUMN

Attributs	Signification	Domaine de valeur
name*	Un identifiant unique pour la colonne dans le tableau	Chaine de caractères

La description d'une valeur : ELEM

Attributs	Signification	Domaine de valeur
value*	La valeur de l'élément	réel
index*	La position (ligne) de l'élément	De 0 à n-1

Exemple de syntaxe :

Pour la description du tableau suivant :

X	Y
0.1	0.15

0.12	0.17

```
<ARRAY name="map1" comment="" >
<COLUMN name="X" >
        <ELEM value="0.1" index="0"/>
        <ELEM value="0.12" index="1"/>
</COLUMN>
<COLUMN name="Y" >
        <ELEM value="0.15" index="0"/>
        <ELEM value="0.17" index="1"/>
</COLUMN>
</ARRAY>
```

B - 2 Exemples de description

B - 2 - 1 Description par les équations pour un circuit RLC série

Le jeu des équations décrivant le modèle du circuit RLC est le suivant :

$$
\begin{aligned}
&E = U_R + U_L + U_C \\
&I_E = I_R \\
&I_R = I_L \\
&I_L = I_C \qquad (1) \\
&U_L = L \times \frac{d}{dt} I_L \\
&I_C = C \times \frac{d}{dt} U_C \\
&U_R = R \times I_R
\end{aligned}
$$

La description XML du circuit sera alors la suivante :

```
<MODEL name="RLC circuit" >
<EQUATION expression="Ie=Ir" />
<EQUATION expression="Ir= Il" />
<EQUATION expression="Il=Ic" />
<EQUATION expression="Ur = R*Rr" />
<EQUATION expression="Ic = C*der(Uc)" />
<EQUATION expression="Ul=L*der(Il)" />
<EQUATION expression="E=Ur+Ul+Uc" />
<DIMENSION datatype="real" visibility="intern" name="Ie" unit="A"/>
<DIMENSION datatype="real" visibility="intern" name="Ir" unit="A"/>
<DIMENSION datatype="real" visibility="intern" name="Il" unit="A"/>
<DIMENSION datatype="real" visibility="intern" name="Ic" unit="A"/>
<DIMENSION datatype="real" visibility="intern" name="Ur" unit="V"/>
<DIMENSION datatype="real" visibility="intern" name="R" unit="ohm"/>
<DIMENSION datatype="real" visibility="intern" name="C" unit="F"/>
<DIMENSION datatype="real" visibility="intern" name="Uc" unit="V"/>
<DIMENSION datatype="real" visibility="intern" name="Ul" unit="V"/>
<DIMENSION datatype="real" visibility="intern" name="L" unit="H"/>
<DIMENSION datatype="real" visibility="intern" name="E" unit="V"/>
</MODEL>
```

B - 2 - 2 Description par composition d'un circuit RLC

Il est également possible de décrire le même circuit par la description d'un système réalisant la connexion de sous-modèles.

On suppose que l'on dispose des éléments de modélisation suivants (tous dans un répertoire nommé « electrical »).

Une résistance

```
<MODEL name="R" >
<EQUATION expression="Uc-Uf=I*R" />
<EQUATION expression="Uc-Uf=U" />
<DIMENSION datatype="real" visibility="extern" name="Uc" unit="V"/>
<DIMENSION datatype="real" visibility="extern" name="Uf" unit="V"/>
<DIMENSION datatype="real" visibility="extern" name="U" unit="V"/>
<DIMENSION datatype="real" visibility="extern" name="I" unit="A"/>
<DIMENSION datatype="real" visibility="extern" name="R" unit="ohm"/>
</MODEL>
```

Une inductance

```
<MODEL name="L" >
<EQUATION expression="Uc-Uf=L*der(I)" />
<EQUATION expression="Uc-Uf=U" />
<DIMENSION datatype="real" visibility="intern" name="Uc" unit="V"/>
<DIMENSION datatype="real" visibility="intern" name="Uf" unit="V"/>
<DIMENSION datatype="real" visibility="extern" name="U" unit="V"/>
<DIMENSION datatype="real" visibility="intern" name="I" unit="A"/>
<DIMENSION datatype="real" visibility="intern" name="L" unit="H"/>
</MODEL>
```

Une capacité

```
<MODEL name="C" >
<EQUATION expression="I=C*der(Uc-Uf)" />
<EQUATION expression="Uc-Uf=U" />
<DIMENSION datatype="real" visibility="intern" name="Uc" unit="V"/>
<DIMENSION datatype="real" visibility="intern" name="Uf" unit="V"/>
<DIMENSION datatype="real" visibility="extern" name="U" unit="V"/>
<DIMENSION datatype="real" visibility="intern" name="I" unit="A"/>
<DIMENSION datatype="real" visibility="intern" name="C" unit="F"/>
</MODEL>
```

Dans ces conditions la description du circuit RLC prend la forme suivante

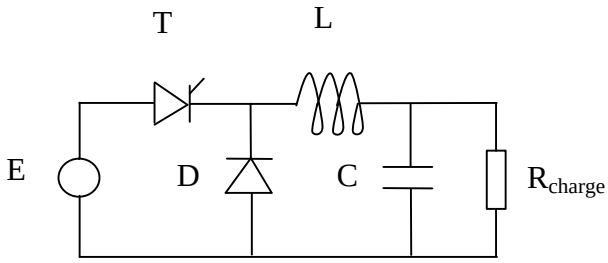
```
<MODEL name="RLC circuit" >
<ELEMENT datatype="electrical.R" name="R1"/>
<ELEMENT datatype="electrical.L" name="L1"/>
<ELEMENT datatype="electrical.C" name="C1"/>
<ELEMENT datatype="electrical.SourceV" name="E"/>

<EQUATION expression="E.U=R.U+L.U+C.U" />
<DIMENSION datatype="real" visibility="intern" name="E" unit="V"/>
<CONNECTION name="nœud_1" datatype="IDENTITY">
        <PORT name="R.I"/>
        <PORT name="E.I"/>
        <PORT name="C.I"/>
        <PORT name="L.I"/>
</CONNECTION>

</MODEL>
```

B - 2 - 3 Description d'un hacheur série

Soit le hacheur série suivant :



Sa description sous forme topologie variable est la suivante :

```
<MODEL name="buck" >
<EQUATION expression="u_l1 = l1*der(i_l1)" />
<EQUATION expression ="u_Rcharge = Rcharge*i_Rcharge" />
<EQUATION expression ="i_C1 = C1*der(u_C1)" />
<EQUATION expression ="u_C1 =u_Rcharge" />
<DIMENSION datatype="real" visibility="intern" name="u_l1" />
<DIMENSION datatype="real" visibility="intern" name="l1" />
<DIMENSION datatype="real" visibility="intern" name="i_l1" />
<DIMENSION datatype="real" visibility="intern" name="i_C1" />
<DIMENSION datatype="real" visibility="intern" name="C1" />
<DIMENSION datatype="real" visibility="intern" name="u_C1" />
<DIMENSION datatype="real" visibility="intern" name="u_Rcharge" />
<DIMENSION datatype="real" visibility="intern" name="Rcharge" />
<DIMENSION datatype="real" visibility="intern" name="i_Rcharge" />
<DIMENSION datatype="real" visibility="intern" name="i_e" />
<DIMENSION datatype="real" visibility="intern" name="e" />

<CONDITION datatype="integer" name="switch1" >
      <STATE comment="discontinuous condition" value="0" >
            <EQUATION expression ="i_e=0" />
            <EQUATION expression ="i_Rcharge =- i_C1" />
            <EQUATION expression ="i_l1 =0" />
     </STATE>

      <STATE comment="continuous condition T-on/D-off" value="1" >
            <EQUATION value="i_e=i_l1" />
            <EQUATION value="e=u_l1 +u_C1" />
            <EQUATION value="i_l1=i_Rcharge+i_C1" />
      </STATE>
      <STATE comment="continuous condition T-off/D-on" value="2" >
            <EQUATION value="i_e=0" />
            <EQUATION value="u_l1=-u_C1" />
            <EQUATION value="i_l1=i_Rcharge+i_C1" />
     </STATE>
</CONDITION>
</MODEL>
```

C - Les modèles orientés : les boites

Les modèles orientés sont issus de l'activité de traitement formel d'un modèle boule. Il s'agit de la description factuelle de la résolution numérique à effectuer pour calculer un jeu de sortie en fonction d'un jeu de paramètres et d'entrées.
Cette description est valable pour des modèles de simulations temporelles comme pour des modèles de régime permanent. Cette description est du type « boite blanche » et est destinée à être utilisée dans un processus informatique de génération de fichiers.

C - 1 La nature d'un « processus de calcul »

Un processus de calcul est une liste d'opérations à réaliser qui conduisent à la résolution d'un problème numérique. Dans cette optique, un certain nombre d'informations sont supposées connues : les paramètres (grandeurs constantes) ainsi que les entrées (excitations du systèmes) sont définies.
Cette connaissance permet de définir une séquence optimale de calcul qui mène à la résolution numérique du système. Le processus de calcul est l'ensemble des informations que l'on a pu construire de cette façon, nature des différentes grandeurs du système, séquences de calculs, proposition de méthode numérique de résolution.

C - 2 La description d'un processus

Deux grandes structures sont apparentes dans la description d'une boite :

- la description des interfaces de la boite de calcul
- la description des états d'un système hybride

La description des interfaces correspond à la définition des grandeurs qui sont en entrée en sortie ou paramètres du modèle.
La description des états du système hybride correspond à la description des séquences de calculs pour les différents états de ce système. Ces différents états sont appelés « ZONE » et contiennent l'information nécessaire à la résolution numérique du système. Ils sont activés par

la validation d’une condition booléenne construite à partir des conditions décrites dans le modèle boule.

Le processus de calcul est décrit dans un fichier qui prend la même structure que les descriptions de boules. C'est-à-dire un fichier compressé au format ZIP qui contient outre un fichier au format XML l’ensemble des fichiers sources nécessaires à la création du code de résolution (fonctions et algorithmes, fichier annexes pour les informations complémentaires).

C - 3 L'arbre d'un processus de calcul

Le fichier au format XML est le fichier central de la définition du processus. Il est structuré sous la forme d'un arbre.

La description d'un processus de calcul prend l'allure suivante :

PROCESS est constitué de

- PARAMETER(*)
- VARIABLE(*)
- FUNCTION(*)
- ARRAY(*)
    - COLUMN(1+)
    - ELEM(1+)

- ZONE(1+) est constituée de
    - STATESYSTEM(0-1) : la description du système d'état
        - A_MATRIX : matrice A de la représentation d'état
        - B_MATRIX : matrice B de la représentation d'état
        - C_MATRIX : matrice C de la représentation d'état
        - D_MATRIX : matrice D de la représentation d'état
    - NOTTIME (0-1): la description d'un ensemble de relations dont les variables ne dépendent pas du temps.
        - RELATION(*) : des équations simples ou des invocations d'algorithmes
            - SOLVEDPARAMETER(1+) : la liste des paramètres évalués par cette relation
        - SYSTEM(*) : un ensemble de relations indissociables, qui forment un bloc de résolution
            - RELATION(2+) : les relations prises en compte dans ce système
            - SOLVEDPARAMETER(2+) : les grandeurs évaluées dans le système
    - TIME(0-1) la description d'un ensemble de relations dont les variables dépendent du temps
        - RELATION(*)
            - SOLVEDPARAMETER(1+)
        - SYSTEM(*)
            - RELATION(2+)
            - SOLVEDPARAMETER(2+)

(*) l'élément peut ne pas être présent, ou alors un nombre non limité de fois
(n+) l'élément est présent au moins n fois
(n-m) l'élément est présent de au moins n fois et au plus m fois.

Comme pour la description des modèles boules, le stockage au format XML des informations permet d'enrichir l'information sur les modèles.

Dans ce document nous ne présentons que les informations minimales de description des différents éléments du fichier XML. Il reste entendu qu'une extension est possible dès lors que les attributs de base, ici présentés, ne sont pas surchargés.

C - 3 - 1 L'élément PROCESS

Le PROCESS est la racine de la description du processus de calcul. Il contient l'ensemble de la description.
Il n'a qu'un seul attribut : son nom, qui est obligatoire.

Attributs	Signification	Domaine de valeur
name*	Un identifiant pour le processus	Chaine de caractères

Exemple de syntaxe :

```
<PROCESS name="rlc">
…
</PROCESS>
```

C - 3 - 2 L'élément PARAMETER

L'élément PARAMETER contient au moins une information : son nom. Il marque les grandeurs du modèle qui sont des paramètres, c'est-à-dire qu'ils ne sont pas variables dans le temps (dans le cas de la description d'un système ayant une évolution temporelle). Les paramètres sont nécessairement connus du modèle.

Attributs	Signification	Domaine de valeur
name*	Le nom du paramètre	Chaine de caractères

Exemple de syntaxe :

```
<PARAMETER name="R"/>
```

C - 3 - 3 L'élément VARIABLE

Les variables sont les toutes les grandeurs physiques du modèle qui ne sont pas des paramètres.

L'élément variable contient au moins quatre informations :

- Un nom qui permet une identification unique de la variable
- La description du statut dans la représentation d'état du système (input, state ou output)
- Une information pour déterminer si la variable doit être « sortie » du modèle.
- Une valeur initiale pour les grandeurs d'état. Par défaut, elle est nulle.

Attributs	Signification	Domaine de valeur
name*	Le nom de la variable	Chaine de caractères
state*	Statut de la variable dans la représentation d'état du système : input : pour une excitation output : pour une sortie state : pour une grandeur d'état	Par défaut : output {input, output, state}
out*	La variable doit être sortie du modèle (true) ou non (false)	Par défaut : true {true, false}
init	Valeur initiale de la grandeur. Indispensable dans le cas d'un variable d'état	Par défaut : 0.0 Un réel Obligatoire pour les grandeurs d'états uniquement
discret*	Signale si une grandeur d'état est discrète (true) ou continue (false).	Par défaut : false {true, false}

Exemple de syntaxe :

```
<VARIABLE name="Ir" state="output" out="false" init="0.0" discret="false"/>
```

Les éléments ARRAY et FUNCTION

Ils reprennent la même structure que pour la description d’une boule.

C - 3 - 4 L'élément ZONE

Les zones sont des descriptions de séquence de calcul qui sont activées en fonction de la validation d'un ensemble de critères représentés par une expression booléenne.

Une zone a un caractère spécifique, elle est dite discrète et son évaluation numérique a une place bien fixée dans le processus d'intégration numérique des équations différentielles. Cette zone contient l'ensemble des algorithmes qui sont discrets, ainsi que ceux qui permettent de définir l'état du système hybride. Cette zone n'est pas obligatoire, mais sa présence dans la description d'un processus de calcul invite à prendre des précautions particulières pour la génération du code de résolution numérique.

Attributs	Signification	Domaine de valeur
condition*	Décrit les conditions de calcul de la zone. Dans le cas de la zone discrète, cet attribut doit prendre la valeur **« discret »**. Dans les autres cas, il s'agit de l'expression de la condition d'activation de la zone.	Par défaut : 1==1 L'explicitation de la condition d'activation de la zone

Exemple de syntaxe :

Description de la zone discrète

```
<ZONE condition="discret">
…
</ZONE>
```

Description d'une zone « classique »

```
<ZONE condition="switch1==1">
…
</ZONE>
```

C - 3 - 5 Les éléments NOTTIME et TIME

L'élément NOTTIME marque une section de calculs numériques dans laquelle l'ensemble des relations n'est pas dépendante du temps. Cette séparation est intéressante dans le cadre de la simulation temporelle car elle permet d'éviter la répétition d'un certain nombre d'opérations lors des passages successifs dans les boucles mineures du processus numérique.

L'élément TIME par opposition au précédent marque une section de calculs numériques dans laquelle les RELATION sont dépendantes du temps. Les grandeurs évaluées dans ce cadre sont des fonctions du temps.

La structure des ces deux éléments est similaire, ils contiennent tous les deux les descriptions de séquences d'opérations numériques.

C - 3 - 6 L'élément RELATION

La RELATION est une équation ou un algorithme qui permet de connaître la valeur d'une plusieurs grandeurs.
Elle a une place dans la séquence de résolution et un type de résolution numérique. Une relation permet d'évaluer une ou plusieurs variables du modèle.
L'évaluation peut être explicite lorsque l'équation le permet ou lorsque l'algorithme n'est pas inversé, ou implicite dans le cas contraire.

Attributs	Signification	Domaine de valeur
expression*	L'équation à résoudre ou encore une invocation d'un algorithme.	Suivant la syntaxe des équations décrite plus haut
order*	Un entier qui indique la position de cette relation dans la séquence de résolution (de 0 à n-1)	Un entier de [0 ; n-1]
resolution*	Le type de résolution possible. Lorsque la résolution d'une équation est explicite, un simple « copier-coller » pourrait être suffisant pour la génération de code.	Par défaut : explicit {implicit, explicit}

Exemple de syntaxe :

Résolution d'une équation

```
<RELATION expression="Ur=R*Ir" order="2" resolution="explicit">
…
</RELATION>
```

Invocation d'un algorithme

```
<RELATION expression="invoke(algo1)" order="2" resolution="explicit">
…
</RELATION>
```

C - 3 - 7 L'élément SOLVEDPARAMETER

Fils de l'élément RELATION et de l'élément SYSTEM, l'élément SOLVEDPARAMETER est utilisé pour donner les identités des paramètres évalués numériquement par la relation ou par la résolution d'un système d'équations.

Cet élément inclue également des informations pour la résolution numérique des problèmes, type de méthodes à utiliser de façon préférentielle, intervalle de recherche de solution…

Attributs	Signification	Domaine de valeur
name*	Le nom de la variable	Chaine de caractères

Exemple de syntaxe :

```
<SOLVEDPARAMETER name="Ur" min="2" max="11" method="Safe-Newton Raphson"/>
```

C - 3 - 8 L'élément SYSTEM

Un système est bloc indissociable de relations. Elles doivent être évaluées simultanément. Un système est caractérisé par le type de résolution que l'on doit utiliser pour le résoudre.

Il contient un jeu de relations ainsi qu'un jeu de paramètres résolus par ce système.

Attributs	Signification	Domaine de valeur
resolution*	linear ou implicit suivant la nature du système	{linear, implicit}
order*	Un entier qui indique la position de cette relation dans la séquence de résolution (de 0 à n-1)	Entier dans [0, n-1]

Annexe C

Algorithmes utilisés pour le traitement des modèles

Annexe C Algorithmes utilisés pour le traitement des modèles

Dans le corps de ce rapport, nous n'avons pas explicité tous les algorithmes utilisés pour le traitement formel des modèles. En particulier, la détection des systèmes d'états, la détection des grandeurs fixées par les paramètres ou encore l'ordonnancement ne sont pas complètement donnés dans le corps du rapport.

Cette annexe présente les algorithmes que nous avons utilisés et dans certains cas mis au point.

A - Détection des grandeurs équivalentes à des paramètres

Cet algorithme est également utilisé pour la détection des grandeurs équivalentes à des grandeurs d'états.

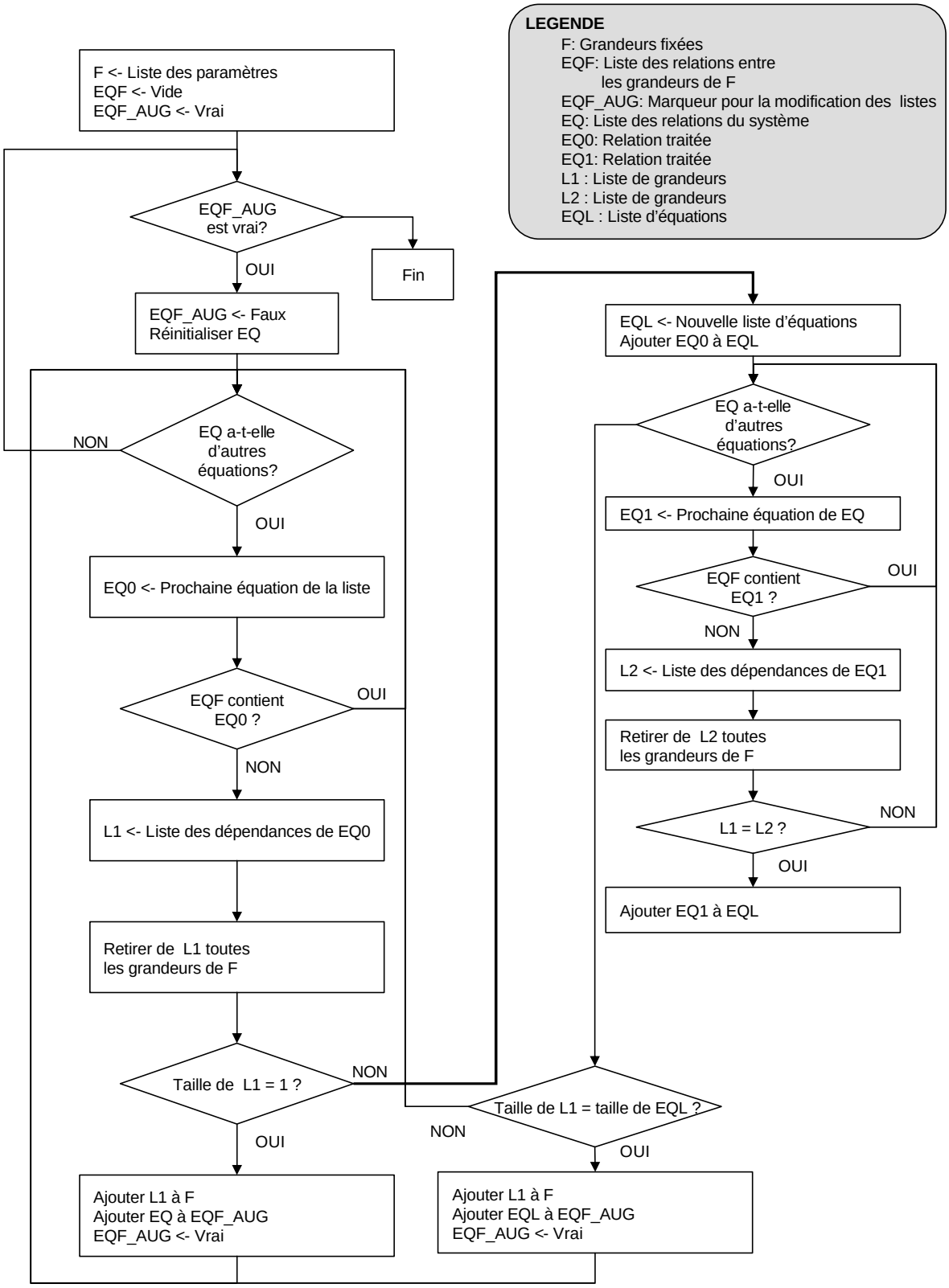


Figure C. 1 : Détection des variables équivalentes à des paramètres

B - Construction du vecteur d'état

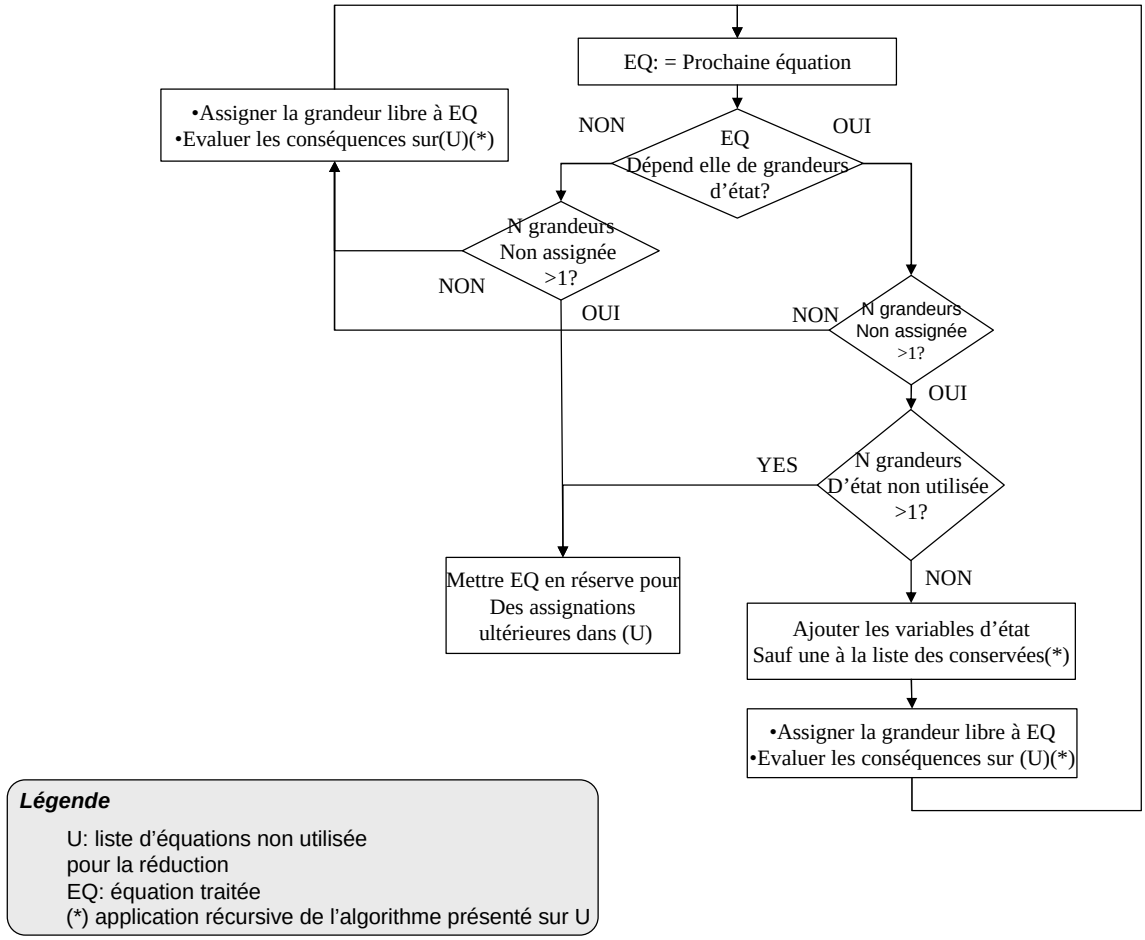


Figure C. 2: Construction du système d'état

C - Ordonnancement

Nous présentons ici une reproduction de l'adaptation de l'algorithme de Tarjan [TAR] par I.S. Duff [DUF] pour la construction d'une matrice triangulaire par bloc.

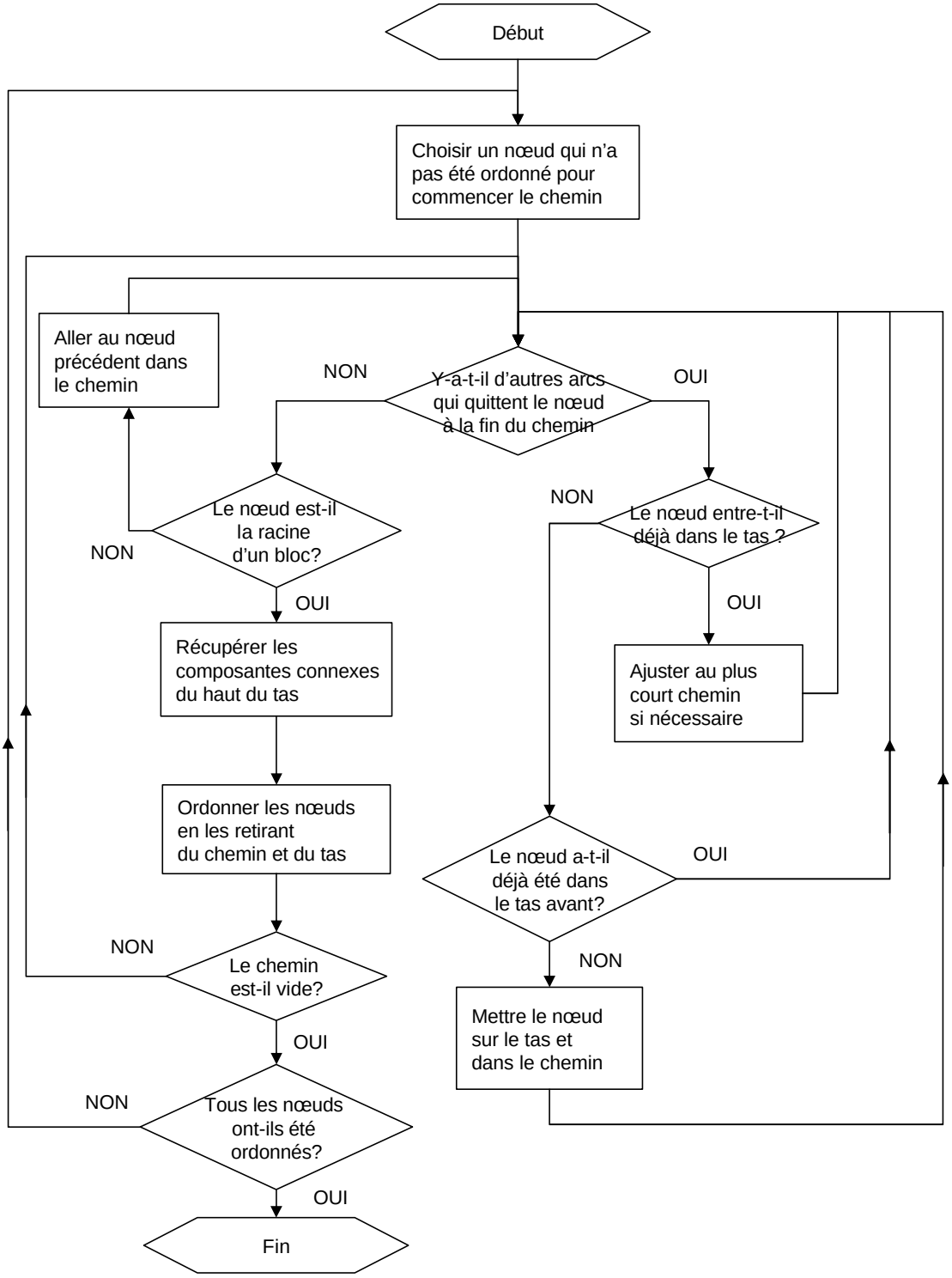


Figure C. 3: Reproduction de l'algorithme de Tarjan pour la séparation des composantes connexes d'un graphe

Nous ne commenterons pas cet algorithme ici. Les références citées plus haut décrivent en détail son fonctionnement.

Annexe D

Reformulation des modèles

Annexe D Reformulation des modèles

Dans le chapitre 3 de ce rapport, nous proposons une démarche de traitement formel des modèles pour transformer la représentation « boule » des modèles en une représentation « boite » plus à même de permettre la génération de code.

La première étape de ces actions est la reformulation du modèle dans une forme sur laquelle le traitement automatique est possible.

Cette annexe détaille les étapes de ce processus de reformulation qui concernent l'implantation des concepts de Modélisation Orientée Objet :

- l'héritage sans et avec surcharge
- l'agrégation, ou composition des modèles

A - L'héritage sans surcharge

Dans ce cas, le modèle résultant se compose :

- des relations décrites par le concepteur dans ce modèle
- des relations décrites dans le modèle père.

Cette construction se fait récursivement. La figure Figure D. 1 présente le cas d'un héritage sans surcharge.

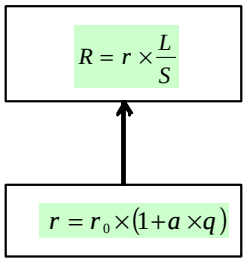


Figure D. 1: Description d'une résistance dépendant de la température par héritage et sans surcharge

Dans ce cas, le modèle résultant se compose des deux relations suivantes :

$$r = r_0 \times (1 + a \times q)$$

$$R = r \times \frac{L}{S}$$

B - L'héritage avec surcharge

Pour la construction du jeu de relations résultant de la description de l'héritage, le concepteur doit, récursivement,

- construire la représentation du modèle père
- remplacer les relations qui sont surchargées dans le modèle fils.

La figure Figure D. 2 présente l'exemple d'une surcharge de relations.

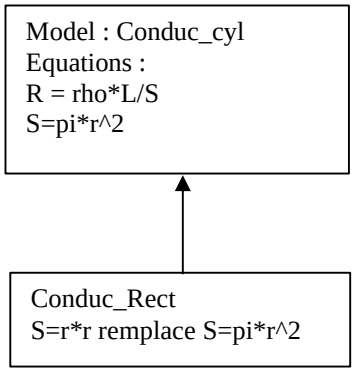


Figure D. 2: Exemple de modèle dont une relation surcharge une relation du modèle père

Dans ce cas, le modèle résultant pour un conducteur de section carrée est le suivant :

$$R = rho \times \frac{L}{S}$$
$$S = r \times r$$

C - La composition d'un modèle

La figure Figure D. 30 présente un exemple de description d'une résistance linéaire dont la longueur est issue d'un calcul externe.

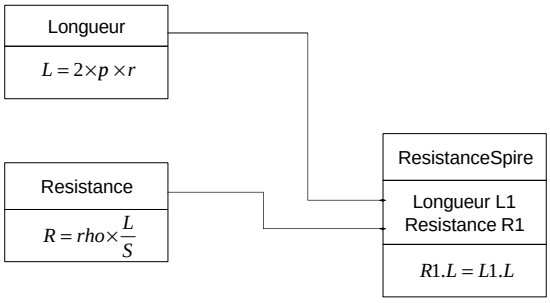


Figure D. 30: Exemple de composition de modèle - Calcul d'une résistance linéaire dont la longueur dépend d'un modèle

Cet exemple permet de réaliser un modèle de résistance linéique par associations de deux éléments de bases pris dans une bibliothèque pour, par exemple, définir un calcul de résistance pour une spire.

Le concepteur qui désire utiliser ce type de modèle doit créer une information qui regroupe l'ensemble des relations sous une forme différente. Lors de l'association des éléments entre eux, il est nécessaire de :

- renommer les variables
- réécrire les équations pour obtenir les relations voulues par le concepteur.

La relecture du modèle permet de créer les équations suivantes qui permettent de décrire le modèle global :

$$L1\_L = 2 \times p \times L1\_R$$

$$R1\_R = R1\_rho \times \frac{R1\_L}{R1\_S}$$

$$R1\_L = L1\_L$$

Annexe E

Traitement formel RAMA

Annexe E Traitement Formel : RAMA – Rule Applicator for Mathematical Analysis

A - Manipulations formelles

Pour automatiser les transformations d'une boule en une boite, nous avons besoin d'un ensemble d'opérateurs formels tels que la dérivation ou encore la séparation des parties réelles et imaginaires des expressions complexes. Par ailleurs, l'utilisation de moteur de calcul formel est impossible étant donné le besoin d'indépendance de l'outil final.

Dans ce contexte, nous avons besoin d'un outil léger, c'est-à-dire rapide et de faible occupation mémoire.

L'outil développé doit :

- permettre la dérivation automatique des expressions mathématiques classiques
- permettre la séparation des parties réelle et imaginaire des expressions complexes
- être rapide, étant donné le grand nombre d'utilisation nécessaires, il ne doit pas être gourmand en temps
- être indépendant
- être extensible, l'ensemble des besoins en traitement n'est a priori pas connu, nous devons donc pouvoir étendre ses capacités à peu de frais
- être pilotable depuis un autre logiciel

B - Architecture et fonctionnement de RAMA

B - 1 Architecture de RAMA

La solution au problème d'extensibilité soulevé dans le cahier des charges est d'implanter un système à base de règles.

Ces règles sont définies dans des fichiers textes et sont donc modifiables au gré des besoins de l'utilisateur, ce qui permet de disposer d'un ensemble extensible et modifiable d'opérations formelles applicables sur une expression.

Nous illustrons sur la Figure E.1 le principe de fonctionnement de RAMA sur un exemple de dérivation d'une expression (***a*(b+c)***) par rapport à une variable (***a***).

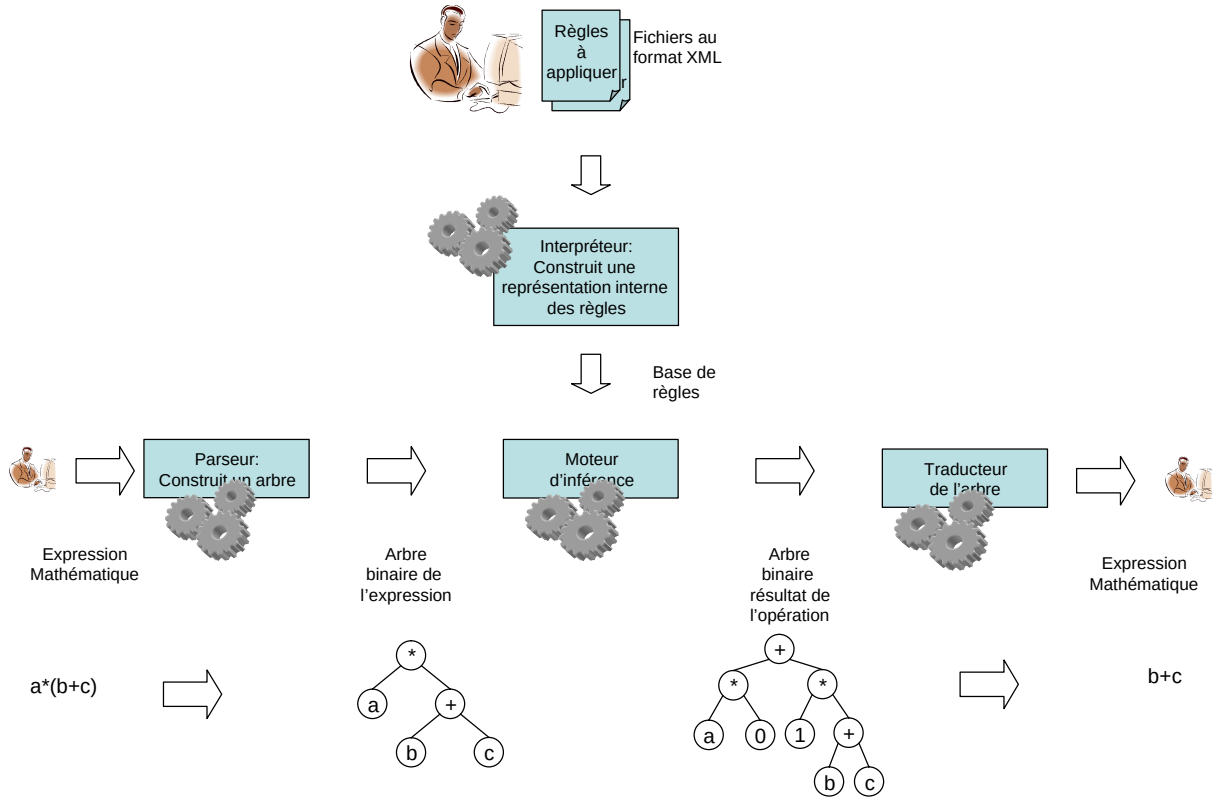


Figure E.1: Principe de fonctionnement de RAMA

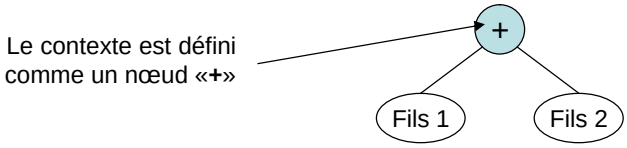
Nous voyons ainsi que la solution technique que nous avons réalisée permet d'obtenir une grande souplesse dans la définition des opérations formelles simples à réaliser sur des expressions mathématiques.

Nous avons choisi un formalisme proche de celui imposé par la norme ANSI-C pour la grammaire des expressions mathématiques à analyser et traiter.

Ces expressions mathématiques sont analysées pour construire un arbre binaire correspondant à la représentation des expressions. Il s'agit d'une structure classique en traitement formel sur les expressions. Cet arbre binaire porte le nom de MOM (Mathematical Object Model), et il s'agit de la base de travail pour les manipulations formelles. En effet, les règles décrites dans les fichiers textes sont constituées de deux éléments :

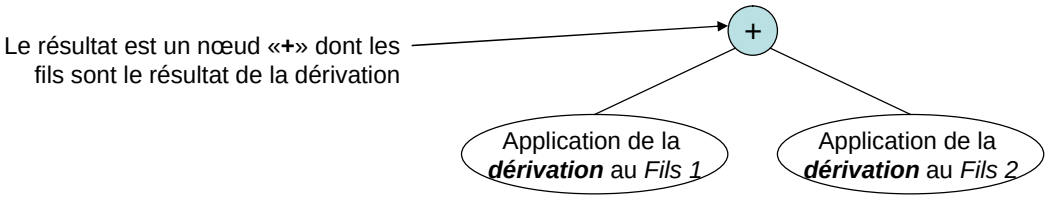
- un contexte d'application qui est une description de la sous-structure de l'arbre sur laquelle la règle s'applique (Figure E. 2)
- un résultat qui reproduit la structure de l'arbre résultant de l'application de la règle

Contexte:




```
<RAMA:CONTEXT >
        <RAMA:MOM name="+"/>
</RAMA:CONTEXT>
```

Résultat du traitement:




```
<RAMA:RESULT>
        <RAMA:MOM name="+" type="operator">
                <RAMA:for-each select="self/child::*">
                        <RAMA:apply-ruleset name="differentiate" select="self"/>
                </RAMA:for-each>
        </RAMA:MOM>
</RAMA:RESULT>
```

Figure E. 2: Exemple de description d'une règle : la dérivation d'une somme

B - 2 Description des MOM

Les expressions mathématiques sont transcrites sous la forme d'un arbre binaire [DAV]. Tous les opérateurs sont des nœuds binaires, c'est-à-dire qu'ils ont deux fils, à l'exception des nœuds représentant des invocations de fonction, qui sont des nœuds planaires dont les fils sont ordonnés en fonction de leur position dans l'invocation de la fonction.

Les MOM sont caractérisés par un type :

- opérateur : pour les opérations classiques de somme, produit, différence et quotient
- fonction : pour les fonctions mathématiques telle que le cosinus, la racine nième, …
- variables pour identifier les grandeurs qui sont des variables pour le traitement formel (par exemple, lors de la dérivation d'une expression par rapport à « ***a*** » seul a est une variable, et tous les autres éléments de l'expression sont des constantes
- constantes, pour identifier les opérandes constantes de l'expression (réel, entier ou paramètres d'une expression)

La différence entre les différents nœud de l'arbre se fait par un identifiant, leur nom, ainsi ;

- les opérateurs peuvent prendre 5 noms : « + », « / », « * », « - », ou « = »
- les fonctions sont repérées par leur nom (« cos » pour le cosinus, « pow » pour la fonction puissance, ….)
- les variables et les constantes sont identifiées par la chaîne de caractère utilisée pour les représenter

Les MOM sont également utilisés pour réaliser des opérations de fractionnement des expressions mathématiques trop longues. Cette capacité est particulièrement utile pour la génération de code afin de gérer les tailles des instructions informatiques.

B - 3 Fonctionnement du moteur d'application des règles

Dès lors que l'expression mathématique a été transcrite de sa forme textuelle sous la forme de MOM, le moteur d'application des règles est utilisé.

Le fonctionnement du moteur est très simple :

- il parcourt l'arbre binaire de l'expression mathématique, en se déplaçant de nœud en nœud
- en fonction du contexte qu'il rencontre, c'est-à-dire en fonction de la nature du sous-arbre dont le nœud courant est la racine, il détermine la règle à utiliser
- enfin, il applique la règle en construisant un arbre résultat par interprétation des instructions qui décrivent l'arbre résultant de l'application des règles