Définition et réalisation d'une nouvelle génération de logiciel pour la conception des moteurs du futur

Guillaume Lacombe

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de

DOCTEUR DE L'INP Grenoble

Spécialité : Génie Electrique

préparée au Laboratoire de Génie Electrique de Grenoble

dans le cadre de

l'Ecole Doctorale Electronique Electrotechnique Automatique Télécommunication Signal

présentée et soutenue publiquement

par

Guillaume Lacombe

Le 29 Novembre 2007

Définition et réalisation d'une nouvelle génération de logiciel pour la conception des moteurs du futur

DIRECTEUR DE THESE Albert Foggia
CO-DIRECTEUR DE THESE Yves Marechal

JURY

M. Georges Barakat	, Président
M. Guy Friedrich	, Rapporteur
M. Pascal Brochet	, Rapporteur
M. Albert Foggia	, Directeur de thèse
M. Yves Marechal	, Co-directeur de thèse
M. Jean-Claude Mipo	, Examinateur
M. Xavier Brunotte	, Examinateur

Table des matières

Table des figures

Liste des tableaux

Introduction

Les logiciels Flux, 2D et 3D, sont des logiciels de modélisation des phénomènes électromagnétiques en deux et trois dimensions par la méthode des éléments finis. Ils sont destinés au domaine du génie électrique pour la conception et la modélisation de dispositifs très variés comme les moteurs électriques, les actionneurs, les transformateurs, le chauffage par induction ou le contrôle non destructif pour n'en citer que quelques uns. Leur large éventail d'utilisation permet d'étudier aussi bien la tête de lecture d'un enregistreur magnétique que la signature magnétique d'un porte-avions. Ils sont utilisés de par le monde (Europe, Etats-Unis et Asie du sud-est) par un millier de sociétés et d'universités. Ces logiciels sont co-développés par la société Cedrat et le laboratoire de génie électrique de Grenoble (G2ELab) et commercialisés par le groupe Cedrat.

En 2000, Cedrat a entrepris la refonte complète de ces logiciels pour porter son interface au niveau des standards en la matière. De plus, on a constaté que les logicels Flux sont essentiellement utilisés par des experts dans les centres de recherche et développement et les laboratoires de recherche. Or une tendance lourde est à l'utilisation des outils de simulation dans les bureaux d'étude, à condition qu'ils soient simples d'utilisation et dédiés au métier des utilisateurs. Ainsi en 2003, Cedrat et le G2ELab entament un programme de recherche visant à développer une plateforme permettant la refonte des logiciels Flux, l'amélioration des processus de développement de tous les logiciels de la société Cedrat et la capacité à décliner les logiciels généralistes Flux, de manière à développer rapidement des logiciels dédiés à un métier. Ce projet a notamment été porté par la thèse de Yves Souchard intitulée "Réalisation d'une plateforme informatique dédiée au métier du génie électrique autour des logiciels Flux. Application à la réalisation de logiciels métiers" soutenue en 2005 [31].

L'application machine électrique est très présente dans le domaine de la simulation en génie électrique (A titre d'exemple, elle représente près de 30% du chiffre d'affaires de Cedrat et 50% des ventes Flux). Elle se développe de plus en plus dans l'automobile (4% par an et promotion des voitures hybrides) et dans l'aéronautique (pour diminuer les coûts opérationnels des actionneurs hydrauliques actuels). De plus, avec les récentes politiques énergétiques menées par la commission européenne, l'augmentation du rendement des machines électriques, qui représentent 65 à 70% de la consommation électrique dans l'industrie, devient un enjeu majeur.

Pour la conception de machines à haut rendement, l'utilisation d'outils de simulation puissants, permettant un dimensionnement fin, est donc renforcée. Cependant, pour être utilisés par un plus grand nombre, ces outils doivent être plus simple d'accès et doivent mieux s'inscrire dans la démarche de conception. Ils doivent permettre d'obtenir rapidement les résultats attendus et être automatisables de manière à pouvoir définir plus facilement des processus de dimensionnement ou d'optimisation. C'est pourquoi, dans ce contexte et dans la poursuite des travaux menés sur l'architecture des logiciels Flux, nous proposons de développer une démarche de personnalisation des logiciels Flux au métier des machines électriques. Ce travail a pour objectif d'améliorer et de capitaliser notre compréhension du métier de la conception des machines électriques. Ceci afin de mieux répondre aux besoins des concepteurs et de spécifier et développer un nouveau logiciel dédié. D'un point de vue génie logiciel, ce travail doit également permettre de consolider la démarche de personnalisation des logiciels Flux à un métier afin qu'elle garantisse une bonne évolutivité du logiciel que nous allons développer et qu'elle soit reproductible pour d'autres métiers.

Puisque le projet demande de travailler deux aspects (l'assimilation du métier moteur et la réalisation logicielle), j'ai choisi de développer ce mémoire en deux parties. Dans la première partie, à partir d'une analyse des outils de simulation pour la conception des machines électriques, je définirai les fonctionnalités auxquelles le logiciel devra répondre. On peut donc assimiler cette partie à la définition ou la spécification du logiciel. Elle est plutôt dédiée aux utilisateurs ou aux concepteurs de moteurs. La seconde partie portera plus particulièrement sur l'implémentation logicielle. Je m'intéresserai notamment aux concepts informatiques liés à la personnalisation des logiciels généralistes Flux en logiciels métiers. La stratégie développée dans la plateforme existante et les nouvelles fonctionnalités qui ont été mises en place pour faciliter et démocratiser cette démarche de personnalisation seront présentés. Enfin je montrerai comment ces outils ont été utilisés pour développer le logiciel que nous appelerons FluxMotor.

Première partie

Définition des exigences du métier moteur

Chapitre 1

Analyse de la démarche du concepteur

1.1 Introduction

Afin de mieux appréhender les attentes et les besoins pour la conception des machines électriques, je propose de commencer ce mémoire par une analyse de la démarche du concepteur. Je présenterai dans ce premier chapitre les différents outils ou logiciels utilisés. J'identifierai les atouts et les limitations de chacun d'eux afin de dégager les fonctionnalités ou les évolutions que nous développerons dans les logiciels Flux afin qu'ils soient plus efficaces.

1.2 Les outils dans la démarche de conception

1.2.1 Les contours de l'analyse

Avant tout, nous commencerons par définir les contours du domaine de conception. En effet, le dimensionnement d'un moteur comporte de nombreuses contraintes et fait intervenir plusieurs domaines de la physique (électromagnétisme, thermique, mécanique, électronique). Au vu du domaine d'application de Flux, nous limiterons ici notre analyse à la conception de la partie magnétique du moteur. Nous prendrons cependant en compte :

- La température pour les résistances des phases
- La charge mécanique
- La commande, limitée à de simples régulations en courant ou en tension

La prise en compte des besoins de la conception élargie sera abordée au travers de la communication avec :

- Les logiciels basés sur des méthodes analytiques pour le prédimensionnement magnétique et thermique
- Les logiciels pour la conception de la commande ou la simulation système

1.2.2 Les outils analytiques et le prédimensionnement

Fruits d'années d'expérience des concepteurs, les outils analytiques capitalisent des topologies et des méthodologies propres au métier des machines électriques. Ils ont souvent une IHM (Interface Homme Machine) dédiée et la prise en main est très rapide. Ces outils sont souvent basés sur des réseaux de reluctances pour la partie magnétique. Les calculs sont très rapides, ce qui rend ces outils idéaux pour le prédimensionnement et le choix de structures. C'est le cas du logiciel SPEED [42] développé par le SPEED Laboratory à Glasgow et commercialisé par le groupe Cedrat (Figure 1.1).

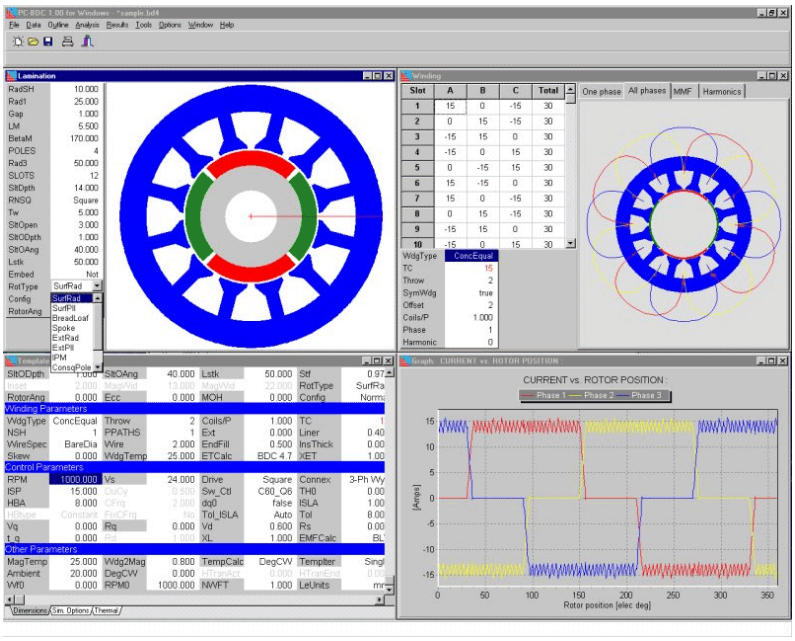


FIG. 1.1 – Logiciel Speed : Méthodes analytiques pour le prédimensionnement de la partie magnétique de la machine

L'aspect thermique est un élément essentiel dans le processus de conception des machines. L'augmentation de la température entraîne une augmentation des résistances des phases et une diminution de l'induction rémanente des aimants. Elle a donc un impact important sur les performances de la machine. Le logiciel MotorCAD [43] est un logiciel dédié à l'étude thermique des machines tournantes. Il est développé par la société MotorDesign à Shropshire (Angleterre) et est également commercialisé par le groupe Cedrat. Il est basé sur des réseaux de sources et de résistances thermiques qui permettent de déterminer rapidement l'évolution de la température dans les différentes parties de la machine suite à un fonctionnement en charge ou à des impulsions de courant. Il permet de décrire de nombreux systèmes de refroidissement (ailettes, ventilations, refroidissement par écoulements, ...). Il permet également de réaliser des études de sensibilité sur les différents paramètres.

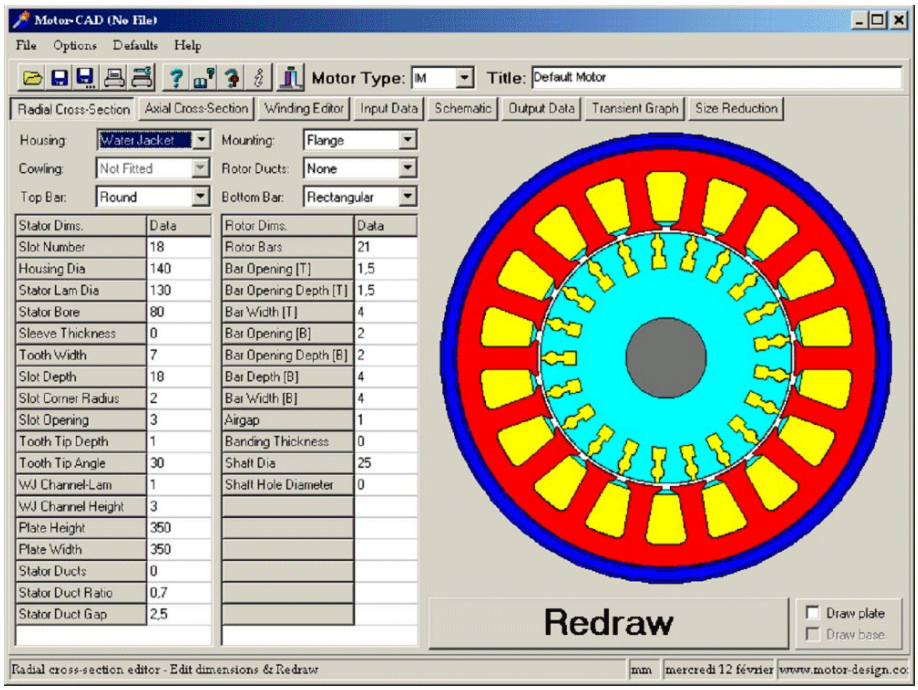


FIG. 1.2 – Définition métier d'un moteur sous MotorCAD

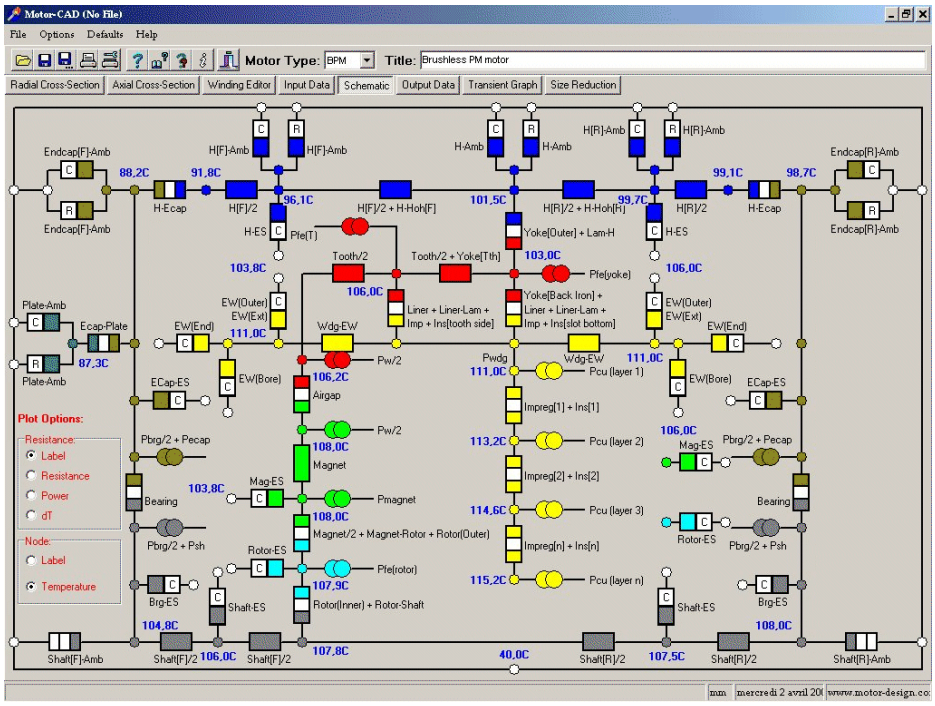


FIG. 1.3 – Schéma thermique équivalent au moteur proposé dans MotorCAD (fonction des paramètres métiers)

1.2.3 Les outils éléments finis et le dimensionnement

L'utilisation des éléments finis en électrotechnique s'est imposée grâce à son aptitude à résoudre les équations de Maxwell dans des domaines de forme complexe. Elle permet la résolution de problèmes statiques, transitoires (pas à pas dans le temps) et harmoniques ; les signaux sont supposés sinusoïdaux et on utilise les nombres complexes. En transitoire et en harmonique, on peut coupler les éléments finis à de la mécanique et associer un circuit électrique.

Les résultats sont précis mais le temps de calcul est long (même s'il a beaucoup diminué particulièrement en 2D). De plus l'IHM est souvent généraliste, ce qui rend beaucoup plus longue la définition du calcul. Pour toutes ces raisons, les éléments finis sont plutôt utilisés pour la conception de nouvelles structures, le dimensionnement fin, la caractérisation des schémas de reluctances ou la caractérisation des schémas équivalents.

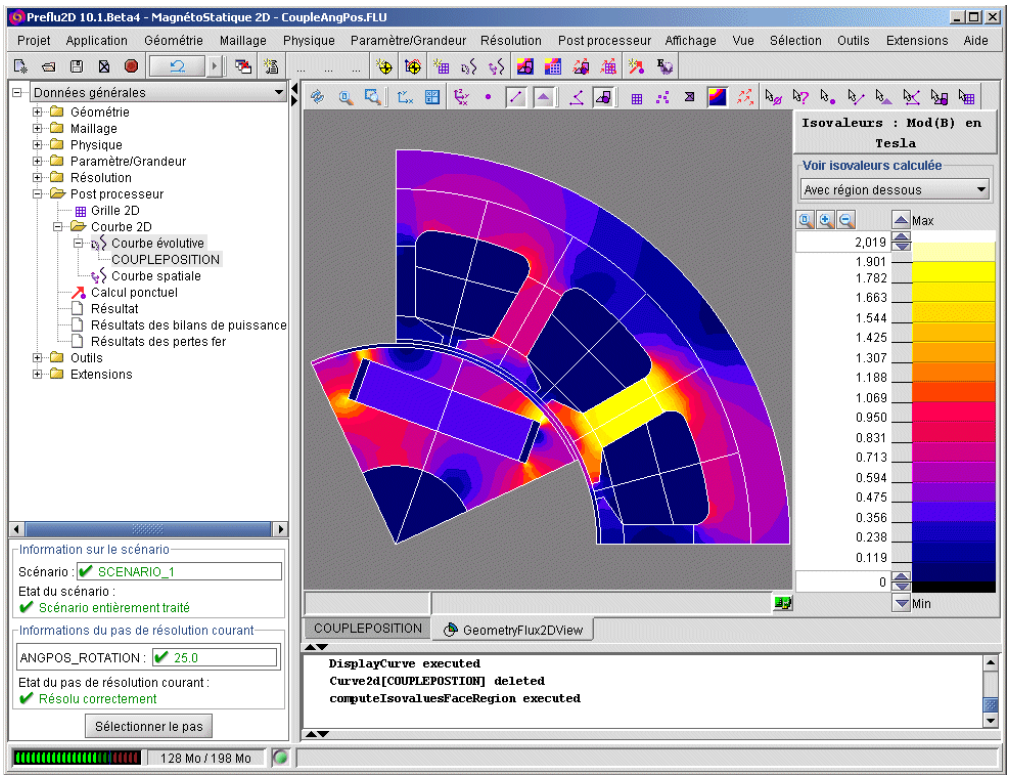


FIG. 1.4 – Logiciel Flux

1.2.4 L'optimisation

De plus en plus, les concepteurs se tournent vers l'optimisation. De nouvelles techniques d'optimisation (plan d'expérience, surface de réponse, ...) permettent main-

tenant de réaliser des optimisations utilisant les outils numériques [26] [27] [28] [29]. Mais ces techniques sont encore réservées à des utilisateurs expérimentés.

L'optimisation intervient dans des phases de prédimensionnement (les outils analytiques très rapides permettent d'explorer de nombreuses structures) et dans des phases de dimensionnement fin avec les outils éléments finis. Le logiciel GOT développé au G2ELab et qui fait partie de la suite logicielle Flux capitalise l'ensemble de ces techniques d'optimisation.

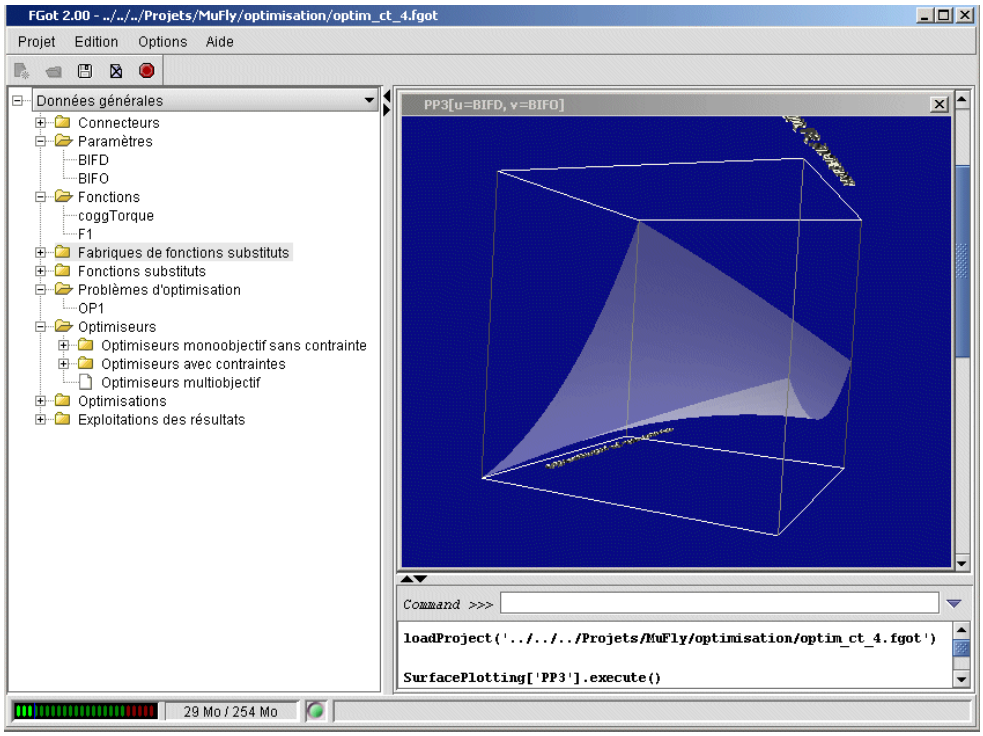


FIG. 1.5 – GOT : General Optimisation Tools

1.3 Les besoins

1.3.1 Démocratiser les méthodes numériques

Définir la géométrie, le maillage et la physique d'un moteur dans un logiciel généraliste peut paraître long pour le concepteur. Il doit comprendre les concepts de modélisation du logiciel généraliste et les utiliser pour traduire son métier. Cette complexité demande souvent de la formation et limite l'utilisation des méthodes numériques. En effet les concepteurs cherchent souvent à vérifier le résultat de leur prédimensionnement avec des calculs par la méthode des éléments finis parce qu'ils sont plus précis. Cependant leur utilisation paraît parfois trop onéreuse pour répondre à un appel d'offres par

exemple. Les concepteurs préfèrent donc, généralement, utiliser des outils dédiés plus rapides.

Pourtant lorsqu'on utilise les éléments finis, les études réalisées dans un domaine d'activité sont sensiblement les mêmes. La saisie répétitive des données relative à une étude peut paraître une perte de temps. Des topologies de moteur et des études classiques pourraient être automatisées à partir de concepts métier comme dans les logiciels basés sur les méthodes analytiques.

1.3.2 Faciliter la communication entre les différents outils

Les couplages entre les différents outils utilisés lors de la conception facilitent la vie du concepteur et évitent les erreurs. Pour les mettre en place, il est important de partager les mêmes paramètres.

Communication avec les outils analytiques

L'analyse des outils du concepteur de machines tournantes montre que les méthodes analytiques et numériques sont complémentaires. Les outils analytiques sont très utilisés en phase de prédimensionnement. Les calculs éléments finis sont souvent nécessaires pour vérifier la véracité des résultats. Il est donc souhaitable de faciliter la communication entre ces outils. Par exemple, l'import d'un moteur SPEED dans Flux permettrait d'automatiser le passage du prédimensionnement au dimensionnement et faciliterait l'accès aux éléments finis.

Couplage et cosimulation avec les logiciels de l'électronique

Une fois le dimensionnement magnétique de la machine effectué, les concepteurs réalisent ensuite le dimensionnement de la commande. Pour cela ils utilisent souvent un schéma équivalent de la machine. Les paramètres de ces modèles peuvent être calculés avec Flux pour être ensuite utilisés par le concepteur de la commande. Cependant les concepteurs de la commande ne sont pas toujours familiés avec les outils éléments finis et les méthodes d'analyse des machines électriques permettant d'identifier ces schémas. Nous pourrions donc proposer une fonctionnalité permettant d'automatiser, à partir de calculs Flux, la création de composants informatiques, représentants le schéma équivalent et qui soient directement utilisables par le logiciel de la commande.

Le logiciel Portunus [41] par exemple permet de dimensionner la commande et propose une bibliothèque de modèles équivalents d'actionneurs. Ce logiciel est développé en Allemagne par la société Adapted Solutions. Ce logiciel sera bientôt en mesure d'utiliser des composants au format VHDL-AMS (Very High Speed Integrated Hardware Description Lanaguage - Analog and Mixed System) [39]. Nous pourrons donc proposer dans Flux la génération de composants au format VHDL-AMS representant un schéma électrique équivalent au moteur.

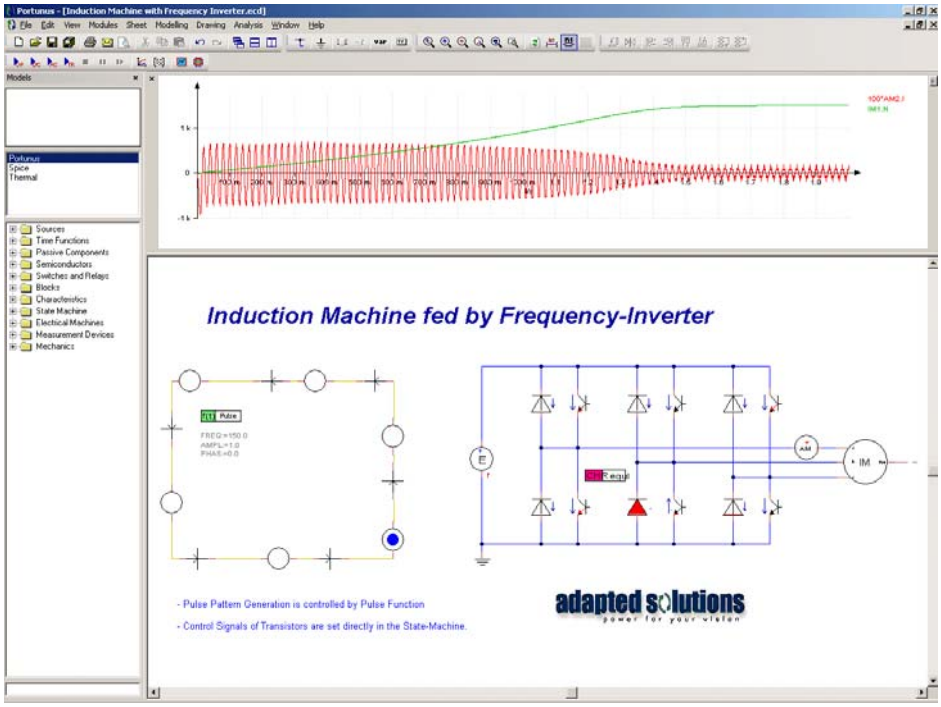


FIG. 1.6 – Simulation de la commande d'une machine asynchrone dans le logiciel Portunus

La commande a un impact important sur les performances réelles de la machine. Nous souhaitons donc également créer des composants informatiques directement utilisables dans un logiciel comme Portunus ou Simulink permettant de la simulation simultanée avec Flux (co-simulation). Ceci permettrait de voir comment la commande se comporte non plus face à un comportement de machine approximatif (schéma équivalent) mais de manière plus fine. De plus, alimenter le système éléments finis avec les signaux de la commande permet d'avoir une vue plus réaliste du taux d'harmoniques des courants et des tensions et ainsi avoir une meilleure estimation des performances de la machine, notamment au niveau des pertes fer par exemple (qui dépendent du spectre fréquentiel). Un module de co-simulation entre Flux2D et Simulink existe. Notre logiciel pourrait donc créer un composant Simulink.

Couplage avec la simulation système

La simulation système cherche à prendre en compte le moteur dans son environnement plus complet : dissipation de la chaleur, modélisation du réducteur et de la transmission mécanique (Figure 1.8). Les schémas équivalents de machines sont également utilisés et les concepteurs systèmes, qui ne sont pas spécialisés dans l'analyse éléments finis des moteurs, rencontrent la même difficulté à déterminer les paramètres des schémas équivalents des moteurs qu'ils choisissent. Des sujets de ce type existent

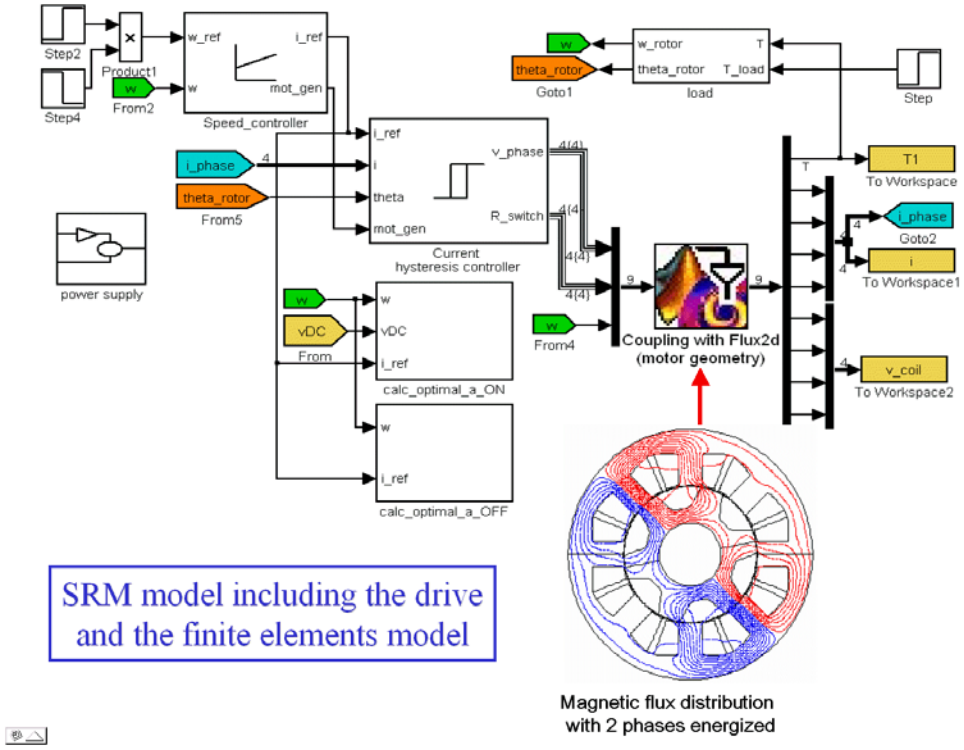


FIG. 1.7 – Commande et modèle éléments finis d'un moteur à reluctance variable

comme par exemple le projet RNTL appelé C6E2 dans lequel Cedrat participe avec la société LMS-Imagine qui développe le logiciel Amesim (Simulation Système). Ce projet vise pour Cedrat à coupler le logiciel Flux avec le logiciel Amesim de manière à proposer un assistant pour le calcul des paramètres des schémas équivalents.

Communication avec les outils hors conception

Enfin on peut citer également la communication avec les autres outils de la société du concepteur. C'est la notion de PLM (Product Life Management) qui permet notamment de garder les résultats des calculs réalisés lors du processus de conception. Ceci demande une bonne structuration des données du métier.

1.3.3 Vers la capitalisation de la démarche de conception

Capitaliser les études réalisées lors d'un projet de conception n'est pas simple aujourd'hui, et les utilisateurs sont parfois amenés à reproduire des simulations qu'ils ont déjà réalisées lors d'une conception ou d'une étude antérieure. L'utilisateur doit gérer un fichier par étude ou par type de calcul dans les logiciels. Une approche métier

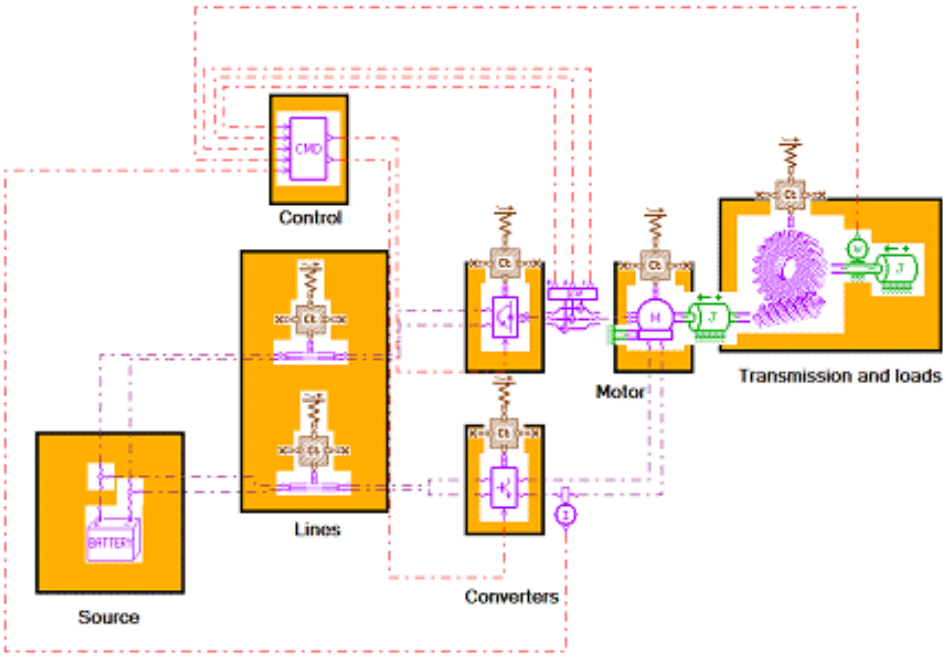


FIG. 1.8 – Amesim : Logiciel de simulation système

permet de reproduire rapidement les calculs ; cependant il convient de se poser la question de la capitalisation.

Il pourrait être interessant pour le concepteur de pouvoir :

- Visualiser et conserver les voies explorées et ainsi formaliser son expérience
- Rendre compte plus facilement du travail effectué (rapports automatiques, arbre de conception, ...)

Ceci permettrait à la société de :

- Formaliser la démarche d'un concepteur, son expérience et sa connaissance métier
- Capitaliser l'historique d'une conception, d'une affaire donnée

1.4 Conclusion

Notre analyse des outils du concepteur montre que le logiciel Flux est particulièrement bien adapté pour la conception de nouvelles structures de machines. Il est également très utilisé pour vérifier les résultats du prédimensionnemnt et pour réaliser un dimensionnement fin, notamment grâce à sa capacité à réaliser des études paramétriques (études de sensibilité sur les paramètres du moteur). L'arrivée de nouvelles techniques d'optimisation renforce d'ailleurs ce dernier point.

Cependant son utilisation est lourde et donc coûte cher aux concepteurs. Il en découle que l'approche métier proposée par les outils analytiques fait défaut dans les logiciels Flux. Enfin, en partageant les mêmes paramètres que les autres outils métier, Flux pourrait mieux s'insérer dans le processus de conception.

Chapitre 2

Objectif de l'application moteur

2.1 Introduction

Dans le chapitre précédent, des fonctionnalités qui permettraient de mieux adapter le logiciel Flux à la conception des machines électriques ont été identifiées. Dans ce chapitre, je présenterai les développements que nous avons décidé d'entreprendre pour répondre à ces besoins et la façon dont notre travail s'est organisé.

2.2 Amener la connaissance du métier moteur dans Flux

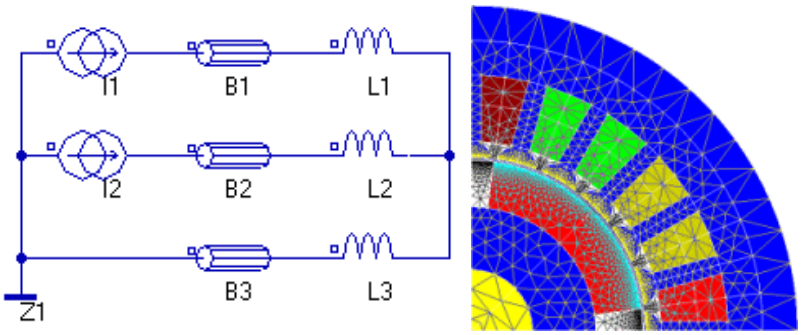
Le premier besoin identifié est d'introduire dans Flux une approche métier. C'est pourquoi nous voulons développer une interface dédiée à l'étude des moteurs. L'objectif de cette nouvelle interface est de simplifier la définition d'un moteur dans Flux. De plus, les études d'analyse des machines électriques souvent réalisées par les concepteurs seront proposées et automatisées.

2.2.1 La géométrie et le maillage

Dans une utilisation classique du logiciel Flux, la définition de la géométrie d'un moteur passe par la définition des coordonnées des points, la définition des lignes et des densités de maillage. Dans l'approche métier, nous proposons des topologies de moteur classiques permettant d'automatiser la construction de la géométrie et du maillage à partir de quelques paramètres métiers. Ceci a l'avantage d'accélérer considérablement le pré-processing et de masquer les notions liées aux éléments finis puisque le maillage est automatisé. Cette démarche a également l'avantage de capitaliser l'état de l'art des topologies des moteurs les plus utilisées.

2.2.2 Les algorithmes métiers

Simuler un moteur dans Flux 2D en régime transitoire demande à l'utilisateur de définir un circuit dans lequel il peut décrire l'alimentation des phases. Dans ce circuit sont représentées les sources de courant ou de tension mais aussi les résistances des phases et les inductances permettant de prendre en compte l'influence des têtes de bobines (Tab 2.1). Flux, en tant que logiciel généraliste, ne peut aider l'utilisateur dans le calcul de ces composants. Dans l'approche métier au contraire, nous pouvons proposer à l'utilisateur des méthodes de calculs analytiques permettant le calcul de ces composants à partir d'une définition métier du moteur et de son bobinage. D'autres algorithmes métiers, comme des algorithmes de répartition classique des conducteurs dans les encoches seront proposées pour simplifier la définition du bobinage. Ces algorithmes peuvent faire gagner du temps puisque le logiciel permet maintenant de faire des calculs que l'utilisateur devait faire lui même auparavant.



Tab. 2.1 – Couplage circuit pour l'alimentation des phases

2.2.3 Les essais normalisés

En plus de proposer des machines prédéfinies et des algorithmes métiers, nous souhaitons capitaliser les méthodes d'analyse par éléments finis de ces machines. On propose donc de développer des études classiques précâblées. Elles permettent d'obtenir rapidement les performances et les caractéristiques principales du moteur. La construction de la physique et le paramétrage des calculs sont automatisés et les résultats sont directement présentés à l'utilisateur. Ces résultats peuvent contenir des valeurs, des courbes, des dégradés et animations ou encore des schémas équivalents et des rapports. Ainsi, on masque la complexité des éléments finis.

2.2.4 La communication

Cette nouvelle interface possède un modèle de donnée dédié à la conception et à l'étude des machines électriques. Il lui est donc maintenant beaucoup plus facile

de communiquer avec des outils dédiés comme les outils analytiques. En présentant des études précâblées, ces bibliothèques capitalisent les modes d'analyse des machines électriques et permettent de développer des fonctions d'export de schémas équivalents vers les logiciels de la simulation système.

2.3 Développer une nouvelle plateforme dédiée moteur

2.3.1 Une plateforme multi-projets

Le logiciel Flux est mono-projet, c'est à dire qu'il ne peut contenir au sein du même projet qu'une seule géométrie et une seule application physique. Dans notre recherche de capitalisation de la démarche du concepteur, nous souhaitons cependant :

- Visualiser tous les résultats des études réalisées sur un moteur.
- Comparer les performances de plusieurs moteurs sur un même essai.

Nous devons donc développer un logiciel multi-projet, proposant différents types de machines (machines synchrones, asynchrones, à courant continu, à reluctance variable, etc..). C'est cette plateforme que l'on appelle FluxMotor.

2.3.2 Une plateforme multi-outils

Même si le logiciel moteur doit contenir des fonctions d'import à partir des outils analytiques et d'export vers les logiciels de la simulation système, l'élaboration d'une nouvelle plateforme multi projet qui pilote Flux nous permet également d'envisager le pilotage d'autres logiciels. Piloter les différents outils sous la même interface comporte de nombreux avantages :

- Calculer de manière analytique ou par éléments finis les études classiques proposées / Importer le résultat d'un prédimensionnement
- Comparer leurs résultats respectifs et ainsi aider l'utilisateur à définir des processus de conception optimaux
- Définir plus simplement une optimisation à partir de concepts métiers et pouvant faire appel à tous les outils (analytiques et numériques : méthode appelée Space Mapping [28] [29])
- Proposer des architectures de commandes classiques de manière à automatiser la cosimulation entre les logiciels Flux et Portunus
- Proposer des études 2D, 3D ou Skewed (prise en compte de l'inclinaison des encoches) sous la même interface.

2.3.3 Notre vision

La modélisation métier des paramètres, des études et des résultats classiques des moteurs permet à l'utilisateur de réaliser plus facilement son processus de conception.

Notre objectif à terme est qu'il puisse modéliser ce processus dans notre interface et ainsi le reproduire à l'aide du language de commande par exemple. Pour cela il convient que l'interface suive au mieux la démarche du concepteur. Je vais donc présenter notre vision à moyen terme de l'organisation des contextes dans notre plateforme.

Le cahier des charges

Un premier contexte devrait proposer la definition du cahier des charges, basé sur la modélisation des paramètres de la machine et de ses résultats. Ainsi on pourra par exemple contraindre l'encombrement de sa machine, afficher un niveau requis de couple moyen à une vitesse donnée et exprimer l'intention de minimiser les ondulations de couple. Le concepteur pourra se restreindre à une ou plusieures technologies si cela fait partie de son cahier des charges. Suite à cette description du cahier des charges, un rapport pourra être créé.

Le prédimensionnement

Dans ce contexte, l'utilisateur devrait pouvoir étudier rapidement la faisabilité de son cahier des charges. Pour cela il aura accès à des méthodes de calcul analytique rapides appliquées aux solutions qu'il envisage (déjà proposée par le logiciel SPEED). Pour l'aider à faire un premier choix de structures, des algorithmes de prédimensionnement et des outils d'optimisation lui seront proposés. La définition du problème d'optimisation, basée sur l'expression des contraintes et des objectifs en partie définis dans le contexte cahier des charges, sera alors plus facile. A la suite de cette optimisation, une ou plusieurs solutions lui seront proposées.

Le dimensionnement

Une fois le prédimensionnement réalisé, il pourra vérifier les résultats à l'aide de calculs numériques et les comparer au cahier des charges. Enfin des méthodes d'études de sensibilité ou d'optimisation numérique lui seront proposées afin d'affiner le dimensionnement.

La validation

Dans la dernière partie, on propose de synthétiser le travail de conception. Pour cela nous proposerons des outils permettant de rédiger des rapports décrivant les solutions envisagées et les résultats obtenus. Enfin des fonctions d'export seront proposées :

- Calcul de schémas équivalents et export vers des logiciels de l'électronique.
- Fonctionnalités d'export vers la CAO et les logiciels de mécanique.

Enfin pour capitaliser l'expérience et l'historique du projet, on permettra d'importer d'éventuels résultats de mesures sur prototype et de les comparer aux résultats de simulation.

2.4 Organisation du travail de thèse

L'ambition affichée précédemment nous pousse à définir les premières étapes du travail et les objectifs pour ces travaux. L'évolution de ces objectifs est détaillée ci-après dans l'ordre chronologique.

2.4.1 Des bibliothèques géométriques de moteurs dans Flux

La première étape de ce travail consiste à développer dans Flux des bibliothèques de machines électriques. Cette étape permet d'abord de simplifier et d'accélérer l'utilisation du logiciel Flux pour les concepteurs de moteurs. Grâce aux logiciels des partenaires, nous avons déjà une bonne spécification des topologies existantes pour les moteurs sans balais à aimants permanents, les machines asynchrones, les moteurs à reluctance variable et les machines à courant continu à aimants ou à stator bobiné. Ce travail ne demandera donc pas beaucoup de spécification même si nous devons réfléchir à l'organisation de l'IHM et à la structuration des données. En revanche une telle approche, pour pouvoir suivre l'innovation, demande des outils évolutifs car de nouvelles topologies de moteur devront être ajoutées. De plus si nous nous interessons au métier du moteur dans le cadre de ces travaux, la démarche métier est destinée à s'étendre à d'autres métiers auxquels peut répondre le logiciel généraliste Flux. Nous devons donc travailler sur une démarche de personnalisation du logiciel Flux réutilisable pour d'autres métiers. Enfin pour encourager et accélérer le développement de nombreuses interfaces métier, nous ferons en sorte de simplifier au maximum la démarche pour qu'elle devienne accessible à un utilisateur avancé de Flux.

2.4.2 Etendre les bibliothèques à la notion de laboratoire virtuel

Une fois ces bibliothèques géométriques en place, il s'agit de les étendre à la notion de laboratoire virtuel. En fait il s'agit d'automatiser les méthodes d'analyse par éléments finis classiquement réalisées sur les moteurs. Cette étape demande un travail de spécification des méthodes d'analyse et de leurs résultats. Puisque toute l'étendue des topologies de moteurs et leur mode d'analyse ne pourra pas être implémenté pendant la thèse, nous avons choisi de nous concentrer sur un type de moteur, ce qui permettra de valider la méthode. Le marché a guidé notre choix sur les moteurs sans balais à aimants permanents (Brushless Permanent Magnet Motor - BPM).

2.4.3 Développement d'une nouvelle plateforme multi-projet pour la conception des moteurs

La dernière étape consiste à proposer des solutions techniques et à développer un prototype de la plateforme logicielle multi-projet FluxMotor. Dans un premier temps ce prototype doit être capable de piloter Flux et ses bibliothèques moteur. Cela demandera donc de mettre en place une architecture logicielle permettant le pilotage de Flux. FluxMotor présente la même interface métier que les bibliothèques sauf qu'il peut contenir plusieurs types de moteurs. De plus il permet de construire plusieurs moteurs et de réaliser plusieurs études dans un seul projet. La gestion des fichiers Flux est masquée à l'utilisateur. Le prototype propose également des solutions pour créer des rapports d'étude.

2.5 Conclusion

Dans ce chapitre, j'ai présenté nos axes de travail, qui sont divisés en trois étapes :

- Développement des bibliothèques géométriques de machines électriques et un couplage avec les outils analytiques. Ce travail sera présenté d'un point de vue fonctionnelle dans le chapitre 3. Il permettra de mettre en oeuvre une méthodologie de personnalisation de Flux qui sera présentée dans la seconde partie.
- Développement d'une bibliothèque complète pour l'étude des moteurs sans-balais à aimants (Brushless Permanent Magnet motor - BPM). Les méthodes d'analyse des moteurs à aimants et les résultats classiques proposés par la bibliothèque seront présentés au chapitre 4.
- Le développement d'un prototype du logiciel FluxMotor. Cette partie fait intervenir essentiellement des considérations informatiques et sera présentée dans la seconde partie du mémoire.

Chapitre 3

La définition des moteurs

3.1 Introduction

Dans ce chapitre je présenterai plus en détails la définition métier des moteurs BPM (Brushless Permanent Magnet) que nous proposons d'intégrer dans Flux2D et FluxSkewed, logiciel Flux qui permet de prendre en compte l'inclinaison des encoches (Figure 3.1). Nous envisageons également une version 3D. Cette bibliothèque permet la définition de la géométrie, du maillage et du bobinage en deux dimensions de nombreux moteurs BPM à partir de paramètres de haut niveau. Elle contient également la définition des matériaux et le calcul des paramètres circuits.

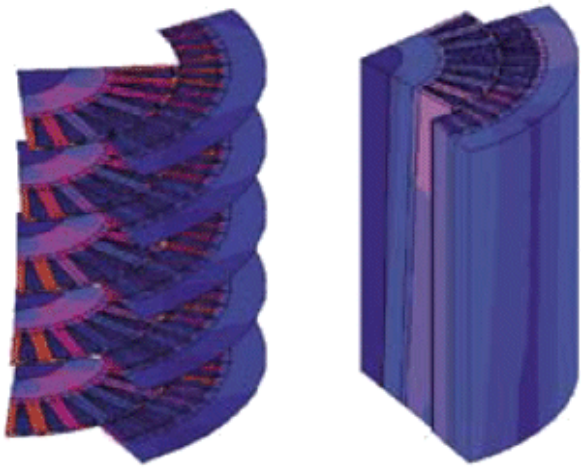


FIG. 3.1 – FluxSkewed propose deux méthodes pour l'inclinaison du rotor ou du stator

3.2 Les topologies

La paramétrisation des moteurs est la même que celle utilisée dans le logiciel SPEED car notre objectif est également de faciliter le lien entre les outils analytiques et numériques. Elle contient une fonction d'import des moteurs SPEED dans Flux. De plus nous travaillons pour retourner les résultats éléments finis dans le logiciel SPEED.

3.2.1 Les rotors

Dans la bibliothèque BPM nous proposons des moteurs à rotors intérieurs. Il existe plusieurs familles de rotors (Figure 3.2) comme par exemple :

- Les rotors à aimantation radiale
- Les rotors à aimantation parallèle
- Les rotors à aimants enterrés
- Les rotors à concentration de flux
- Les rotors à aimants enterrés avec cage d'écureuil

Une fois que l'utilisateur a fait le choix d'un type d'aimantation, on lui propose de choisir parmis plusieurs types d'enfoncement des aimants. En tout la bibliothèque contient 36 topologies de rotors. Chacune des topologies est définie par un certain nombre de paramètres métiers comme l'épaisseur des aimants ou sa longueur angulaire par exemple.

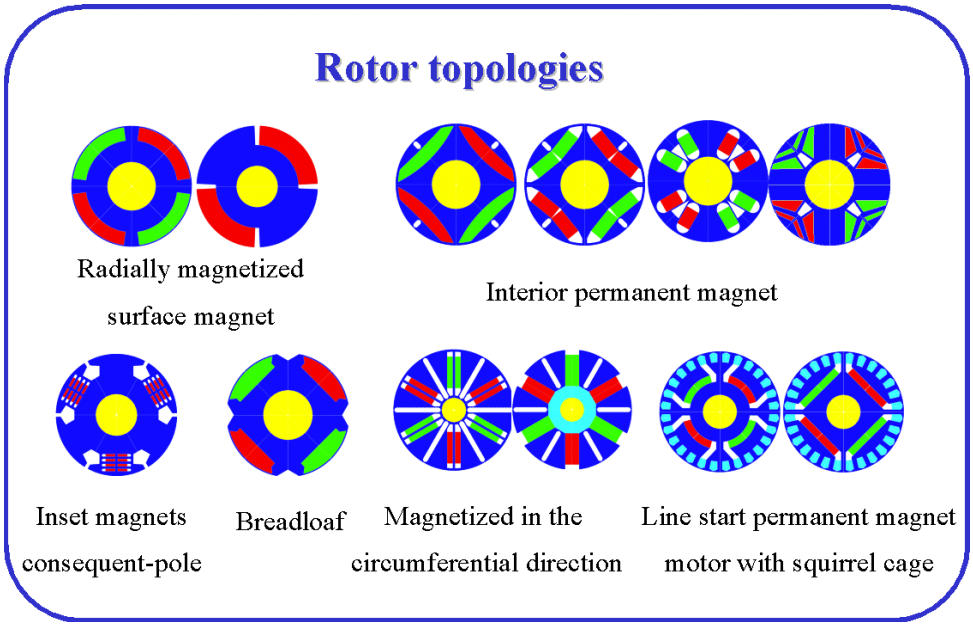


FIG. 3.2 – Les topologies de rotor dans la bibliothèque de moteur sans balais à aimants permanents

3.2.2 Les stators

La définition du stator comprend le choix de la forme d'encoche et le choix de la forme de la carcasse (Figure 3.3). On propose 12 formes d'encoches et 4 formes de carcasses. Comme pour les rotors, chaque forme d'encoche est décrite par une série de paramètres comme la largeur de dent ou la profondeur d'encoche. La technologie employée doit nous permettre de rajouter facilement des formes d'encoches ou des topologies de rotor. On peut déjà penser qu'un mode dans lequel l'utilisateur pourra developper ses propres nouvelles géométries sera développé.

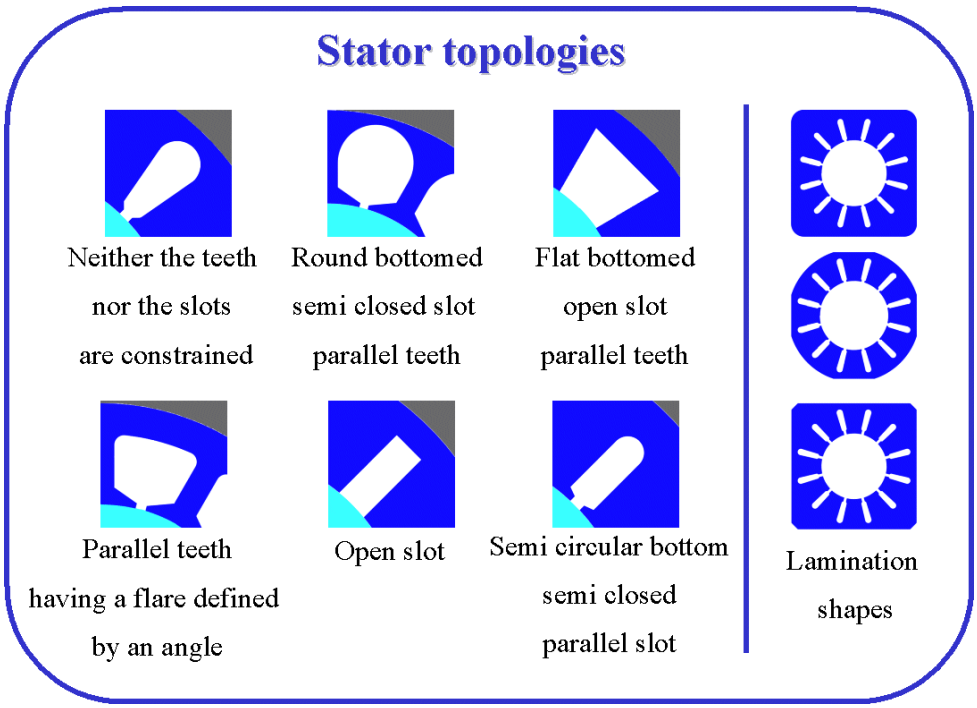


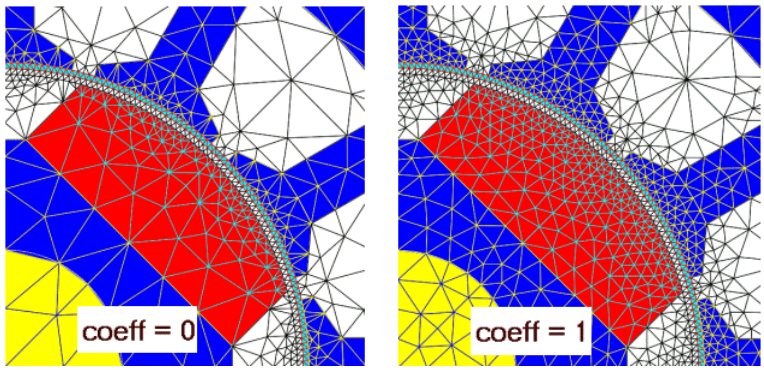
FIG. 3.3 – les topologies de stator dans les bibliothèques moteur

3.2.3 Des paramètres spécifiques

Cependant, nous devons ajouter à cette description purement métier du moteur un certain nombre de paramètres liés à la modélisation éléments finis. Ces paramètres ont été réduits à leur strict minimum de manière à en simplifier l'utilisation.

Coefficient de maillage Dans l'approche métier, la répartition des densités de maillage est automatisée. Cependant certaines études demande un maillage plus dense

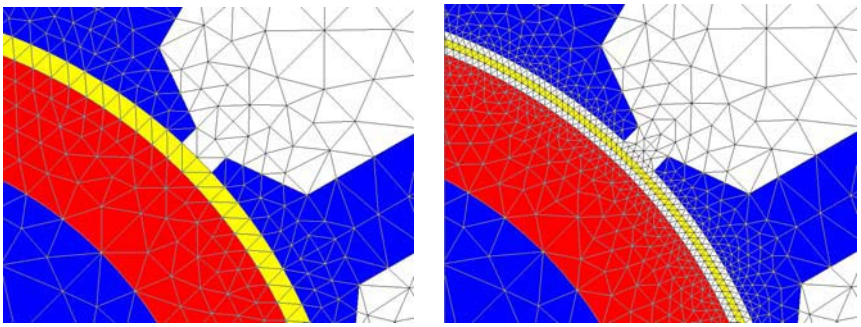
que d'autres. En effet, la densité de maillage est directement liée à la qualité des résultats et au temps de calcul que souhaite obtenir l'utilisateur. C'est pourquoi nous avons ajouté un paramètre compris entre 0 et 1 pour contrôler la densité générale du maillage. Ce paramètre permet de raffiner le maillage lorsque c'est nécessaire ou au contraire de l'alléger. On a donc réduit à son plus bas niveau la définition du maillage pour la rendre extrémement simple et rapide.



Excentricités Le calcul par éléments finis permet un calcul précis de certains défauts et notamment d'étudier l'impact d'éventuelles excentricités. C'est pourquoi nous avons introduit la possibilité de définir des excentricités : on peut décaler le stator et/ou le rotor par rapport à l'axe de rotation du rotor. Ceci permet d'analyser l'impact de la variation d'entrefer sur les performances de la machine.

Les périodicités Afin de diminuer le temps de calcul et d'économiser de la mémoire, il est possible dans Flux de définir des symétries ou des périodicités. Nous avons donc introduit la possibilité d'utiliser les périodicités dans le cas où il n'y a pas d'excentricités. Le logiciel ne trace alors qu'une partie de la géométrie ($\frac{1}{p \gcd(Npôles, Nencoches)}$).

Le nombre de couches dans l'entrefer Certaines études, notamment le calcul du couple de détente, demandent un maillage très fin dans l'entrefer pour obtenir de bons résultats. On propose donc dans les bibliothèques la possibilité de choisir le nombre de couches de mailles dans l'entrefer.



La boîte infinie La résolution d'un problème éléments finis demande la définition d'un certain nombre de conditions aux limites. Dans Flux il est possible d'imposer une condition de champ tangent à l'extérieur du rotor ou de définir une boîte infinie. On peut retrouver cette fonctionnalité dans les bibliothèques à partir d'un simple choix entre les deux solutions.

3.3 Les méthodes de bobinage

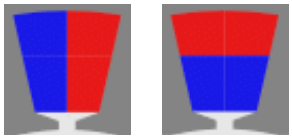
La définition du bobinage dans le logiciel Flux généraliste consiste à rassembler les régions géométriques correpondant à une phase et à leur affecter un composant du circuit externe. On indique alors le sens des conducteurs et le nombre de spires traversant la région. Cette définition est longue. Nous proposons une vision plus métier de la définition du bobinage dans le cadre des bibliothèques.

Même si d'un point de vue éléments finis, en 2 dimensions, l'induction dans l'entrefer ne dépend que de la répartition des conducteurs dans les encoches, la définition des parties frontales est nécessaire pour le calcul des résistances des phases et des inductances des têtes de bobines. On définit donc chaque phase comme une somme de bobines décrites par leur encoche aller, leur encoche retour et le nombre de spires. Ensuite on définit le nombre de chemins parallèles par phase. Cette définition du bobinage est appellée personnalisée car elle permet la définition de n'importe quel bobinage. On associe également à cette définition de la répartition du bobinage, le matériau cuivre utilisé.

Cependant, pour simplifier encore la definition du bobinage, nous proposons à l'utilisateur des répartitions de bobinages classiques [1] [17] que sont les bobinages concentriques par pôles, imbriqués par pôle et à pas fractionnaire. Ceci permet de définir la distribution des conducteurs à partir de deux ou trois paramètres généraux comme le pas de bobinage ou le nombre de bobines par pôle et par phases. Encore une fois notre outil est évolutif et d'autres algorithmes pourraient être implantés dans le futur (bobinage ondulé, ondulé répartis, concentrique à pôles conséquents et imbriqué à pôles conséquents). Enfin grâce à cette définition métier, le changement du pas de bobinage devient plus simple et plus rapide dans un processus de dimensionnement.

3.3.1 Le bobinage imbriqué par pôle

Dans le bobinage imbriqué (Figures 3.4, 3.5, 3.8, 3.9), les bobines d'une même phase et correspondant à un même pôle ont toutes le même pas et sont décalées entre elles d'une encoche. Le pas du bobinage peut être diamétral (c'est à dire égal aux nombre d'encoches sur le nombre de pôles) ou raccourci. Le pas raccourci permet de réduire le taux d'harmoniques dans la fmm (Force magnéto-motrice) et du même coup diminuer le cuivre et donc la résistance et le coût. Une fois une phase définie, les autres sont obtenues par rotation de $\frac{2\pi}{3} radélec$ dans le cas d'un bobinage triphasé. Le nombre de bobines par pôle peut être différent suivant que l'on utilise un bobinage à une couche ou à deux couches. Dans le cas d'un bobinage à deux couches on peut positionner les conducteurs de manière superposée ou adjacente.



TAB. 3.1 – Bobinage à deux couches : Positionnnement dans l'encoche adjacente ou superposée

Les paramètres du bobinage imbriqué sont :
- Le nombre de bobines par pôle et par phase
- Le pas du bobinage
- Le nombre de spires par bobine
- Le nombre de chemins en parallèle par phase (Nombre de voies d'enroulement)

3.3.2 Le bobinage concentrique par pôle

Ce bobinage consiste à placer les bobines sur un pôle en spirale (Figures 3.6, 3.7). Il est souvent à une couche. Il permet notamment de réduire le coût du bobinage (coût du cuivre) et il est moins cher à bobiner. Cependant même s'il favorise le fondamental, il introduit un fort taux d'harmoniques. Les paramètres de ce bobinage sont les mêmes que le bobinage imbriqué .

3.3.3 Le bobinage à pas fractionnaire

Le nombre d'encoches par pôle et par phase n'est pas toujours un entier, ce qui interdit l'utilisation des algorithmes classiques qui viennent d'être présentés. Nous proposons donc un algorithme permettant de définir ce que l'on appelle le bobinage à pas fractionnaire (Figures 3.10, 3.11).

Le bobinage à pas fractionnaire est défini par :

– Le pas du bobinage
– Le nombre d'encoches de décalage entre deux phases (comme le nombre d'encoches par pôles et par phases n'est pas un entier, on ne peut pas toujours décaler les phases d'exactement $\frac{2\pi}{3} radélec$).

3.3.4 La définition des spires

On définit la résistivité du cuivre et la section des conducteurs et de leur isolant. Nous proposons deux sections différentes : rectangulaires ou ciculaires.

Suite à cette définition, nous calculons le coefficient de foisonnement du cuivre et le coefficient de foisonnement total. Nous proposons également une définition du cuivre à partir de nomenclatures comme l'American Wire Gage (AWG). D'autres classifications pourront être introduites par la suite. Pour évaluer lors des essais l'impact de la température sur la résistance de phase, la résistivité est donnée pour une température et on définit un coefficient multiplicateur. La résistivité est alors évaluée comme suit :

$$\rho = \rho_{20} \left[1 + \alpha \left(T - 20^{\circ}\right)\right] \tag{3.1}$$

$\alpha = 3,9.10^{-3}$ pour le cuivre et l'aluminium
$\alpha = 3,8.10^{-3}$ pour l'argent

3.3.5 La définition des têtes de bobines [19]

Dans un calcul éléments finis en deux dimensions, les têtes de bobines sont prises en compte dans le couplage circuit par l'introduction d'une résistance et d'une inductance de têtes de bobines par phase. L'utilisateur peut donc définir la longueur d'une tête de bobine et l'inductance par phase. Cependant nous lui proposons un certains nombre de formules analytiques permettant de les calculer automatiquement.

Le modèle de M. Liwschitz

Dans ce modèle l'utilisateur ne définit que la longueur de la tête de bobine. Cependant ce modèle ne s'applique qu'à des bobinages imbriqués à deux couches.

$$L_{TB} = 2\frac{N^2}{p}\Lambda_{TB} \tag{3.2}$$

avec

$$\Lambda_{TB} = 0.43\mu_0 l_{TB} k_r^2 \tag{3.3}$$

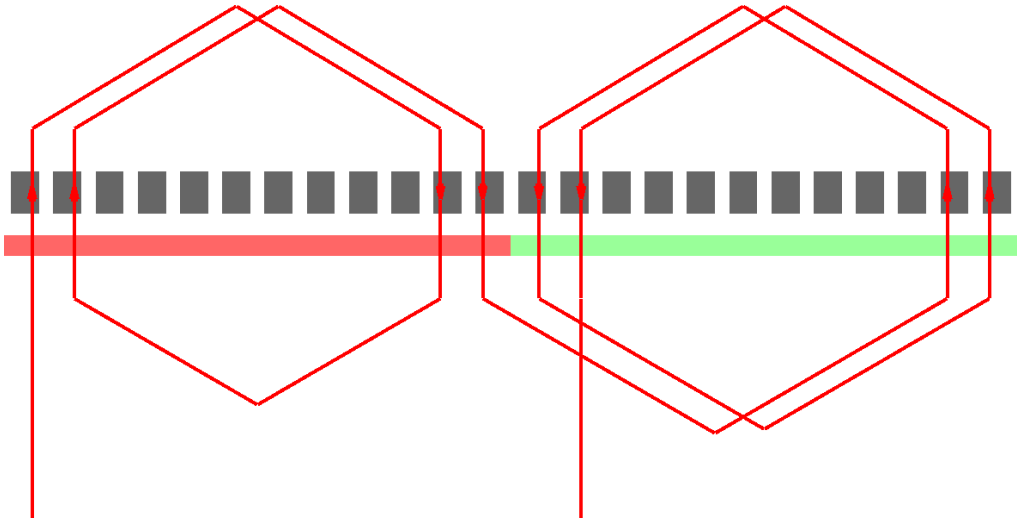


FIG. 3.4 – Bobinage imbriqué par pôle à 1 couche - Phase 1 - Moteur à 24 encoches et 2 pôles - Pas du bobinage : 10 - Nombre de bobines par pôles : 2

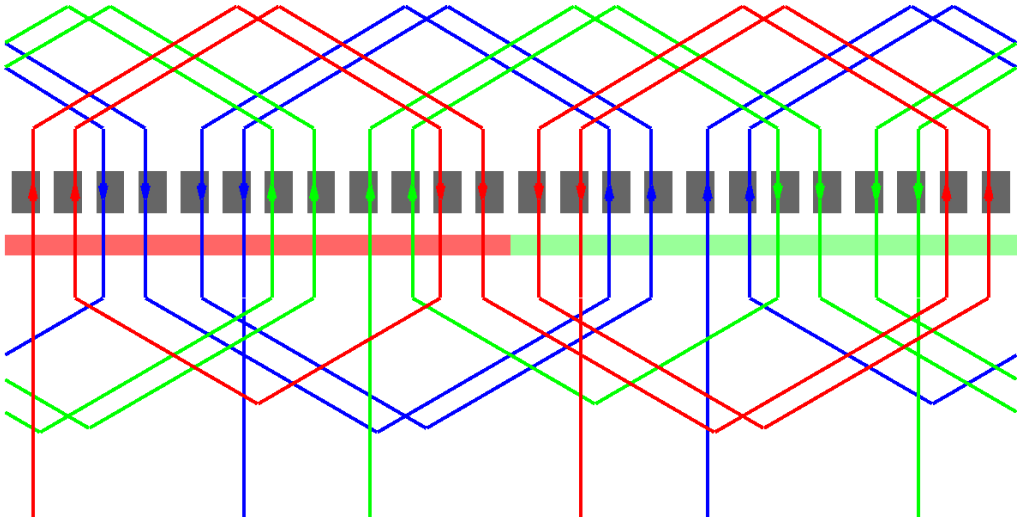


FIG. 3.5 – Bobinage imbriqué par pôle à 1 couche - Les 3 phases - Moteur à 24 encoches et 2 pôles - Pas du bobinage : 10 - Nombre de bobines par pôles : 2

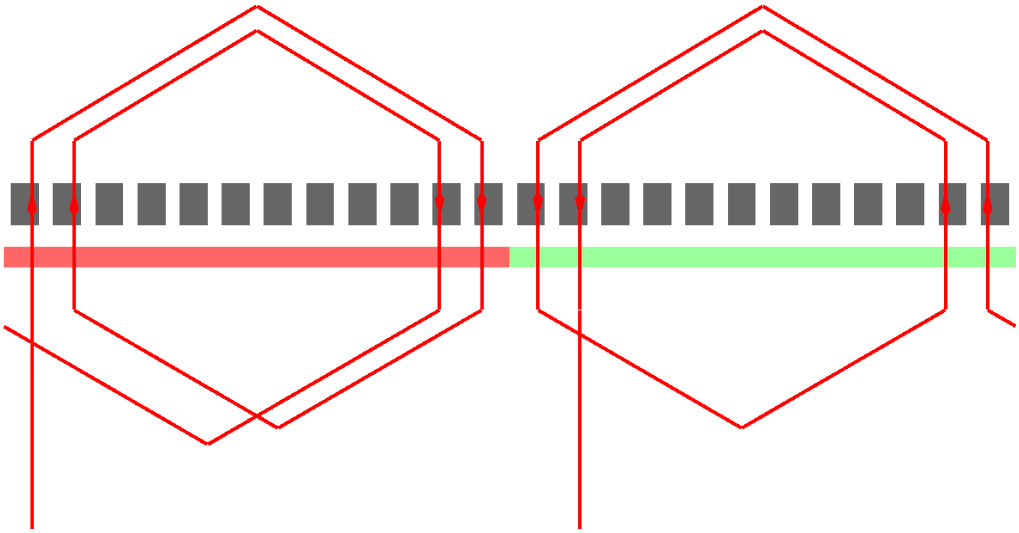


FIG. 3.6 – Bobinage concentrique par pôle à 1 couche - Phase 1 - Moteur à 24 encoches et 2 pôles - Pas du bobinage : 11 - Nombre de bobines par pôles : 2

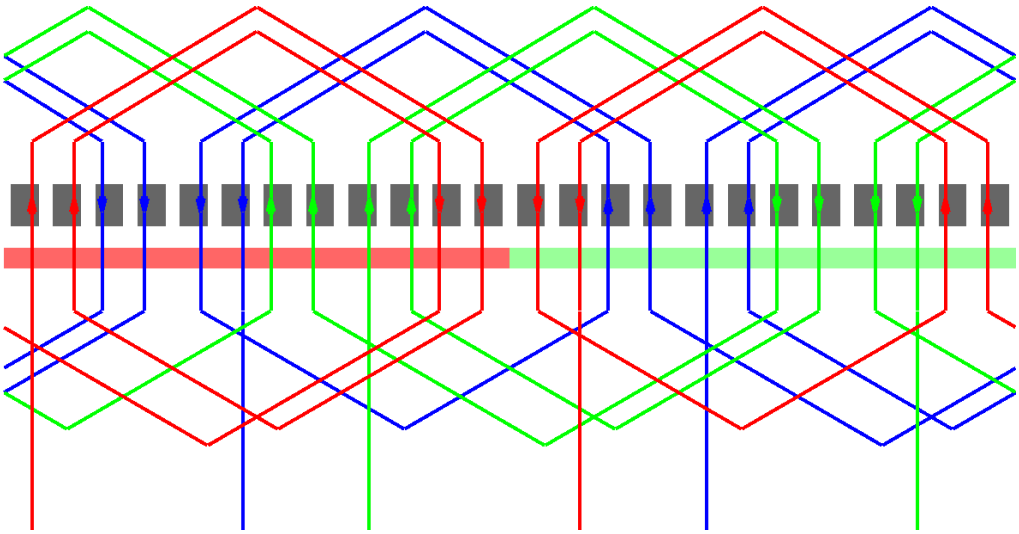


FIG. 3.7 – Bobinage concentrique par pôle à 1 couche - Les 3 phases - Moteur à 24 encoches et 2 pôles - Pas du bobinage : 11 - Nombre de bobines par pôles : 2

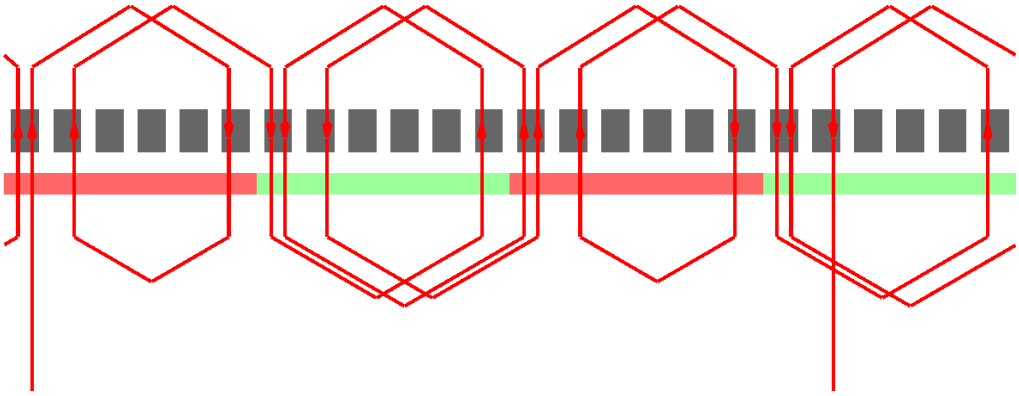


FIG. 3.8 – Bobinage imbriqué par pôle à 2 couches - Phase 1 - Moteur à 24 encoches et 4 pôles - Pas du bobinage : 5 - Nombre de bobines par pôles : 2

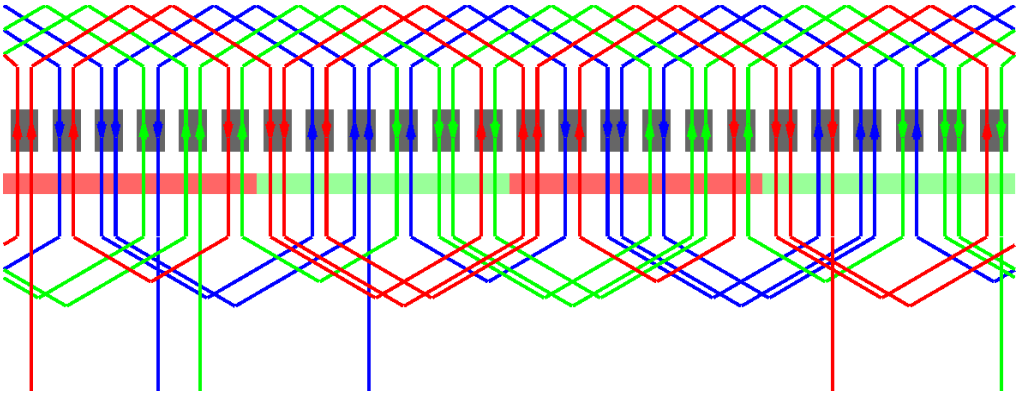


FIG. 3.9 – Bobinage imbriqué par pôle à 2 couches - Les 3 phases - Moteur à 24 encoches et 4 pôles - Pas du bobinage : 5 - Nombre de bobines par pôles : 2

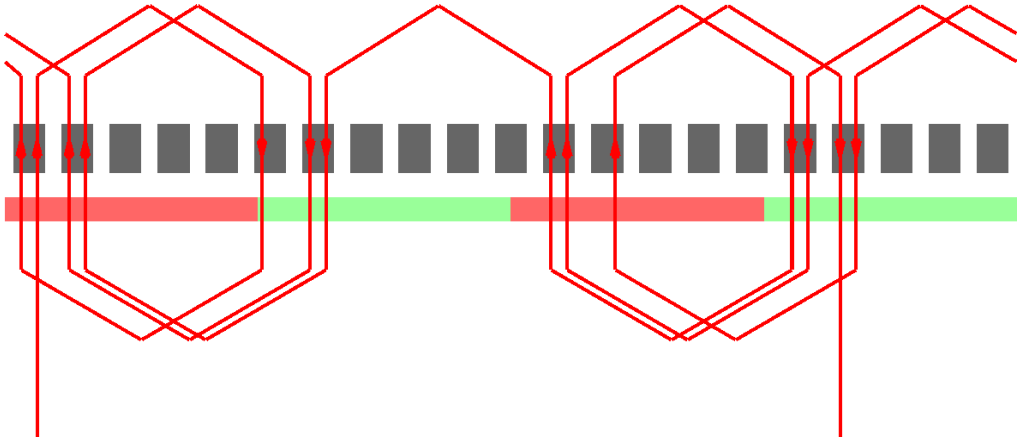


FIG. 3.10 – Bobinage à pas fractionnaire - Phase 1 - Moteur à 21 encoches et 4 pôles - Pas du bobinage : 5

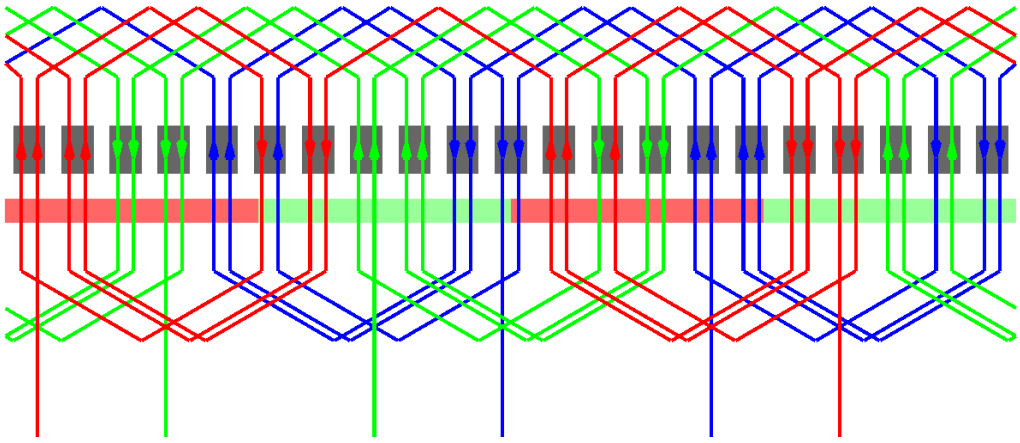


FIG. 3.11 – Bobinage à pas fractionnaire - Les 3 phases - Moteur à 21 encoches et 4 pôles - Pas du bobinage : 5 - Offset : 7

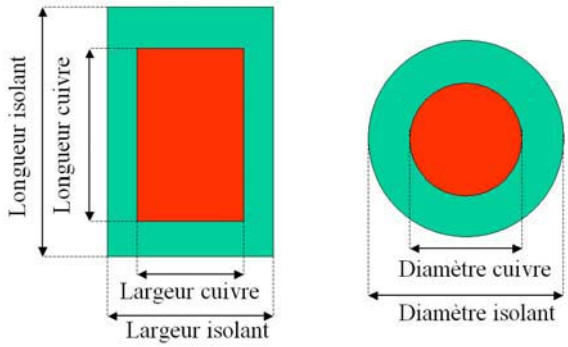


FIG. 3.12 – Définition de la section des spires

Λ_{TB}	:	Perméance des têtes de bobines (H)
N	:	Nombre de spires en série par phase
p	:	Nombre de paires de pôles
k_r	:	Coefficient de raccourcissement du bobinage
μ_0	:	Perméabilité du vide
l_{TB}	:	Longueur d'une tête de bobine (m)

Le modèle de Kostenko

Dans ce modèle l'utilisateur ne définit que le diamètre moyen des enroulements statoriques. Cependant ce modèle s'applique à des têtes de bobines en développantes de cône et ne sera proposé que pour les bobinages imbriqués.

$$L_{TB} = 2\mu_0 \frac{N^2}{p} \Lambda_{TB} \tag{3.4}$$

avec

$$\Lambda_{TB} = 0.57 \frac{3\gamma - 1}{2} \tau \tag{3.5}$$

Λ_{TB}	:	Perméance des têtes de bobines (H)
N	:	Nombre de spires en série par phase
p	:	Nombre de paires de pôles
γ	:	Raccourcissement du pas par rapport au pas diamétral
τ	:	Ouverture du bobinage (m), $\tau = \frac{\pi D}{2p}$ (D : diamètre moyen des enroulements)

Le modèle de P.L. Alger

Encore une fois ce modèle se limite aux bobinages imbriqués, cependant il est très utilisé.

$$L_{TB} = \mu_0 \frac{m N_c^2 D}{P^2} f(\gamma) \tag{3.6}$$

avec

$$f(\chi, \gamma) = \gamma \frac{\tan \chi}{4} \left(1 - \frac{\sin \gamma\pi}{\gamma\pi}\right) + \frac{k_d^2 k_r^2}{6} \left(1 + 0.12\gamma^2\right) \tag{3.7}$$

m	:	Nombre de phases
N_c	:	Nombre de conducteurs en série par phase
D	:	Diamètre intérieur de l'induit (m)
P	:	Nombre de pôles
γ	:	Raccourcissement du pas
χ	:	Angle d'ouverture des développantes, en radians
k_d	:	Coefficient de distrubution
k_r	:	Coefficient de raccourcissement

Le modèle de T. Miller

Ce modèle a l'avantage d'être applicable à tous les types de bobinage. Il est proposé par Tim Miller et JR Hendershot dans "Design Brushless Permanent-Magnet Motors".

$$L_{TB} = \frac{n}{a^2} \frac{\mu_0 N^2 D}{2} \ln\left(\frac{4D}{GMD} - 2\right) \tag{3.8}$$

μ_0	:	Perméabiblité du vide
N	:	Nombre de spires en séries par bobine
n	:	Nombre de bobines par phases
a	:	Nombre de chemins en parallèle
D	:	Diamètre moyen des spires (m)
GMD	:	Distance moyenne entre deux conducteurs dans l'encoche (m) ($GMD = 0.447\sqrt{A}$)
A	:	Surface d'une encoche (ou d'une demi encoche dans le cas d'un bobinage à deux couches)

Méthode par calcul éléments finis 3D

Des méthodes ont été développées sur la base d'une représentation 3D d'une extrémité de la machine pour le calcul des inductances des têtes de bobines [18]. L'une d'elle consiste à alimenter les phases et à calculer l'énergie emmagasinée dans l'air entourant les têtes de bobines. Ces méthodes sont cependant assez lourdes à mettre en place. Les automatiser dans une interface métier a donc un intérêt fort. Cependant cela demande le développement d'une bibliothèque dédiée à la construction des têtes de bobines en 3D, ce qui n'a pas été envisagé pendant la thèse.

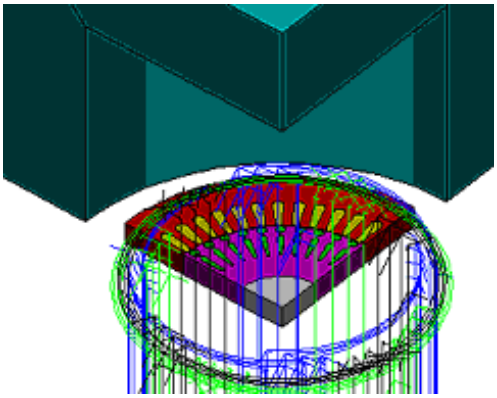


FIG. 3.13 – Représentation 3D des têtes de bobines

3.3.6 La définition de l'anneau de court-circuit

Pour les rotors à aimants contenant une cage d'écureuil, l'influence de l'anneau est pris en compte dans le couplage circuit avec l'introduction de résistances et d'inductances inter-barres. La bibliothèque propose le choix parmis différents algorithmes commes les formules de Trickey ou celles d'Alger [20] [21] [22]. Elles sont basées sur une description géométrique de l'anneau et la résistivité du matériau utilisé.

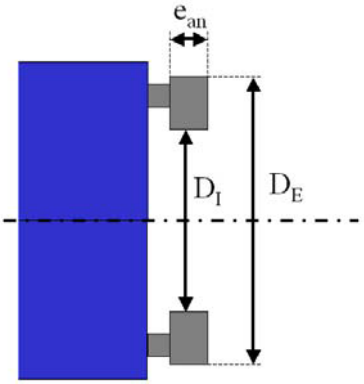


FIG. 3.14 – Paramétrage de l'anneau

$$
\begin{aligned}
P_{fann} &= 0.365\mu_0 \left(\frac{l_{ann}}{2N_R} \log \left(\frac{1.5\frac{l_{ann}}{2}}{e_{an} + \frac{D_E+D_I}{2}} \right) \right) \\
L_{fann} &= \frac{P_{fann}}{2\sin\left(\frac{p\pi}{N_R}\right)^2} \qquad (3.9) \\
R_{ann} &= \varrho_{Al} \frac{l_{an}}{S_{an} N_R} \qquad (3.10)
\end{aligned}
$$

l_{ann} : Périmètre de l'anneau (m)
N_R : Nombre de barres rotoriques
ρ_{Al} : Résistivité des barres (Ωm)
S_{an} : Section de l'anneau $S_{an} = e_{an}\frac{D_E - D_L}{2}$ (m^2)

3.4 Les matériaux

La définition métier des matériaux est proche de celle utilisée dans Flux. En effet de nombreux modèles de courbes B(H) existent déjà dans Flux. Cependant, on pourra différencier, dans le cadre d'une définition métier, les matériaux magnétiques durs des matériaux magnétiques doux. De plus, alors que le calcul des pertes est réalisé en post-processing dans Flux, on poura dans l'approche métier caractériser les pertes dans les matériaux dès le préprocessing.

3.4.1 Stator et rotor

Définition de la courbe B(H).

Plusieurs modèles sont proposés pour définir les courbes B(H) et correspondent à ceux déjà proposés dans Flux (Figures 3.15 et 3.16).

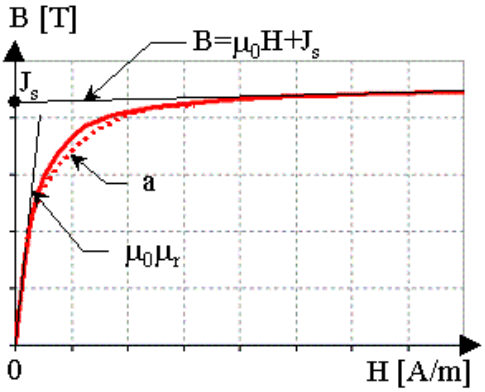


FIG. 3.15 – Modèle isotrope analytique avec contrôle du coude

Des outils permettant de régler les paramètres des modèles arctangentes à partir d'une courbe points par points pourraient être développés par la suite. Certains modèles dépendant de la température, pourraient être introduits. Ceci demanderait cependant pour les essais de définir une température pour chaque région du moteur évaluée à partir de MotorCAD par exemple.

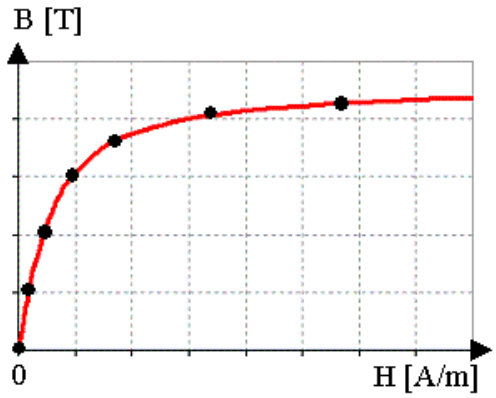


FIG. 3.16 – Modèle points par points

Les pertes fer

Les pertes fer sont évaluées à partir du modèle de Bertotti lors d'un calcul transistoire ou harmonique. Ce calcul est un calcul à posteriori. Il nécessite la définition des coefficients de Bertotti qui sont :

- Coefficients des pertes par hystérésis k_h $(WsT^{-2}m^{-3})$
- Coefficient des pertes classiques qui est la conductivité σ (Sm^{-1})
- Coeffcients des pertes par excès k_e $(W(Ts^{-1})^{-3/2}m^{-3})$
- Epaisseur de tôles d
- Coefficient de foisonnement k_f
- La fréquence f (celle-ci sera déduite à partir du paramétrage de l'essai).

La densité volumique instantanée des pertes fer est évaluée à partir de la formule (3.11) en transitoire :

$$dP(t) = \left[k_h B_m^2 f + \sigma \frac{d^2}{12} \left(\frac{dB}{dt}(t) \right)^2 + k_e \left(\frac{dB}{dt}(t) \right)^{3/2} \right] k_f \qquad (3.11)$$

Ces pertes pourront être affichées sous la forme de dégradés. On donnera aussi les pertes fer sur une période et par régions en intégrant sur la période et sur les volumes. Les constructeurs donnent souvent des courbes de pertes en fonction du niveau d'induction et de la fréquence. Nous travaillerons donc sur un outil permettant de déterminer k_h et k_e à partir des courbes fournies par les constructeurs.

On remarque qu'en transitoire, on donne la fréquence et donc seul le fondamental est pris en compte. Il existe cependant un modèle dynamique dans Flux appelé LSModel. Cependant ce modèle, qui nécessite la connaissance de la courbe de H(B, dB/dt), demande des mesures expérimentales et encore peu de matériaux sont caractérisés dans Flux. Nous devrons donc travailler par la suite à clarifier la démarche de caractérisation de ces matériaux, à l'élaboration d'un éditeur de matériaux et à l'élaboration d'une banque de matériaux plus complète. De plus ces calculs de pertes sont réalisés à postériori et on peut éventuellement les faire apparaître dans le bilan

de puissance. Mais pour être complet, il faudrait être en mesure de prendre en compte ces pertes pendant le calcul. Des travaux de recherche sur ce sujet ont déjà été engagés au G2Elab.

Résistivité

En plus des propriétés magnétiques et des pertes des matériaux, on peut définir la résistivité, notamment dans le cas d'un rotor massif, ce qui induira en transitoire un calcul des courants de Foucault.

Masse volumique

On calcule les volumes des matériaux. La définition de la masse volumique nous permet de calculer la masse des matériaux ainsi que l'inertie du rotor qui pourra être utilisée lors d'un couplage avec la charge mécanique.

3.4.2 Aimants rotor

Nous avons choisi de séparer la définition des matériaux magnétiques doux de celle des aimants puisque les modèles de courbes B(H) proposés sont différents. Cependant les différentes propriétés à définir sont les mêmes (courbe B(H), coefficients de Bertotti, résistivité, masse volumique).

3.4.3 Cage d'écureuil

Les dimensions géométriques de la cage sont définies (section et anneau). Il reste donc à définir la résistivité du matériau.

3.5 Conclusion

Dans ce chapitre, j'ai présenté la façon dont nous souhaitons pouvoir définir un moteur dans les bibliothèques dédiées dans Flux en prenant l'exemple de la bibliothèque de moteur BPM. En plus d'automatiser la construction de la géométrie, ces bibliothèques sont capables de calculer les volumes, les masses, les résistances et les inductances équivalentes aux parties frontales. Les technologies informatiques mises en place pour développer ces biblothèques seront développées dans la seconde partie.

Chapitre 4

La caractérisation des moteurs

4.1 Introduction

Maintenant qu'un moteur BPM (Brushless Permanent Magnet) peut être défini rapidement dans Flux, je vais présenter les fonctionnalités que nous proposons pour obtenir une évaluation rapide de ses performances. L'approche est basée sur des études prédéfinies qui permettent d'obtenir directement les résultats carctéristiques du moteur. Certains résultats permettront à l'utilisateur d'adapter le dimensionnement, d'autres lui permettront de déterminer les caractéristiques électriques équivalentes du moteur pour une utilisation dans la simulation système. Après avoir rappelé les modes de fonctionnement des moteurs BPM et les schémas équivalents utilisés pour ce type de machine, je présenterai les études que nous avons spécifiées.

4.2 Les moteurs brushless à aimants permanents

Les moteurs BPM sont constitués d'un stator, dans lequel sont placés les conducteurs des phases, et d'un rotor à aimants. L'absence de conducteurs au rotor, contrairement à la machine à courant continu ou à la machine synchrone à rotor bobiné, permet de se passer des balais. Il existe deux types de moteurs BPM, les moteurs BDC (Brushless Direct Current) et les moteurs PMSM (Permanent Magnet Synchronous Machine). Leur fonctionnement et leur mode d'analyse sont largement traités dans le livre de JR Hendershot et TJE Miller [1].

4.2.1 Les moteurs BDC

Dans un moteur BDC, les phases sont alimentées par des courants en créneaux fonctions de la position du rotor. Ce régime de fonctionnement est très proche de celui de la machine à courant continu sauf que les aimants tournent et la commutation dans

les conducteurs du stator est assurée par l'électronique. Pour maximiser le couple et minimiser ses ondulations, on cherche à avoir des tensions à vides trapézoïdales (Figure 4.1). En négligeant les pertes, on obtient à vitesse constante un couple constant donné par l'équation 4.1.

$$E_{LL} I = \Gamma_{em} \Omega \tag{4.1}$$

E_{LL}	:	La fem à vide entre 2 phases en série (En BDC deux phases sont alimentées à chaque instant)
I	:	Le courant continu de ligne
Γ_{em}	:	Le couple électro-magnétique
Ω	:	La vitesse mécanique

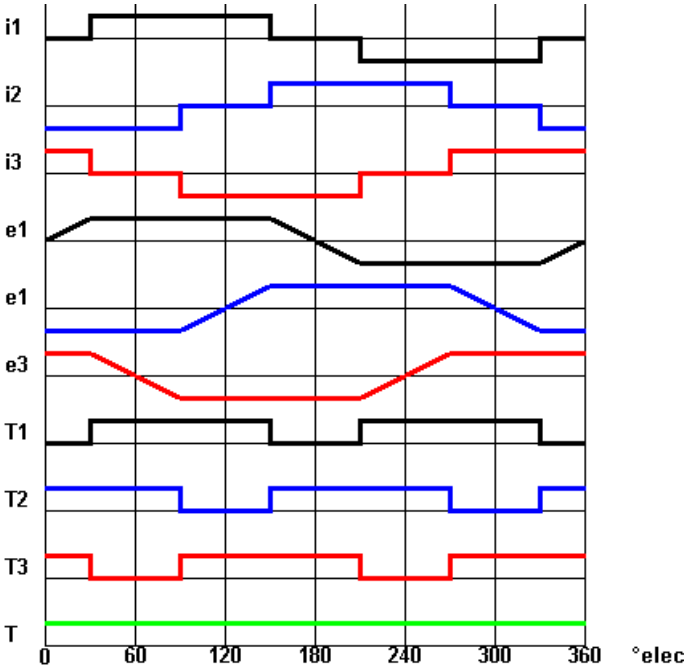


FIG. 4.1 – Formes des courants et des tensions à vide dans un moteur BDC

FIG. 4.2 – Exemple d'un rotor de moteur BDC

4.2.2 Les PMSM

Les PMSM sont alimentés par des courants sinusoïdaux. Ils ont un fonctionnement plus proche de celui de la machine synchrone classique : le rotor bobiné est remplacé par un rotor à aimants permanents. Dans ce cas, on cherche à avoir une fem sinusoïdale. La plupart des signaux sont considérés sinusoïdaux et on privilègie une représentation vectorielle dans le repère du rotor (Figure 4.3).

TAB. 4.1 – Exemples de rotor de PMSM

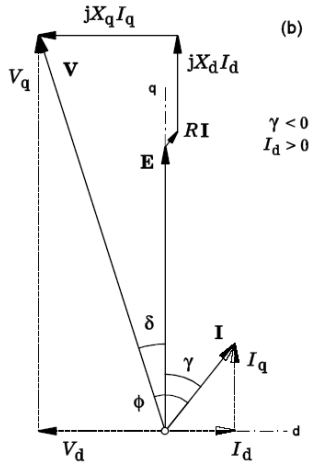


FIG. 4.3 – Diagramme vectoriel du fonctionnement d'un PMSM

4.3 La géneration de composants système

Il existe plusieurs modèles équivalents pour les moteurs BPM. Les trois les plus couramment utilisés seront présentés ici, c'est à dire le modèle par phase et les modèles dans la transformation de Park (premier et second ordre). L'utilisation de ces modèles dépend des hypothèses que l'on peut faire sur la saturation, la forme des pôles et le type d'alimentation.

Dans un moteur brushless à aimants permanents, les tensions aux bornes des phases s'écrivent :

$$\{V\} = [R]\{I\} + \frac{d}{dt}\{\phi\} \tag{4.2}$$

$[R]$: La matrice diagonale des résistances des phases
$\{\phi\}$: Le vecteur des flux traversant chaque phase

Ces flux sont composés de deux termes : le premier correspond aux flux produits par les phases du stator et le deuxième aux flux produits par les aimants.

$$\{\phi\} = [L(\theta, \{I\})]\{I\} + \left\{\phi_{pm}(\theta, \{I\})\right\} \tag{4.3}$$

Chacun des termes dépend de la position du rotor et des courants qui modifient l'état de saturation de la machine. En pratique, il est difficile d'évaluer les inductances en fonction de tous ces paramètres. On fera donc un certain nombre d'hypothèses.

4.3.1 Le modèle par phase

En négligeant les pertes fer par hystérésis et par courant de Foucault, et en supposant que la machine est non saturée, la matrice inductance et les flux produits par les aimants ne dépendent plus de la valeur des courants. De plus si l'on considère un rotor à pôles lisses, les inductances ne dépendent plus de la position du rotor. Les tensions aux bornes de chaque phases s'écrivent alors :

$$\{V\} = [R]\{I\} + [L]\frac{d}{dt}\{I\} + \{E\} \tag{4.4}$$

Le terme $\{E\} = \frac{d}{dt}\left\{\phi_{pm}(\theta)\right\}$ est appelé force électro motrice (fem à vide). Il correspond aux tensions à vide.

En supposant un bobinage correctement réparti, on peut admettre que toutes les inductances propres (notées L) sont égales et que toutes les mutuelles (notées M) sont aussi égales. On peut alors écrire en triphasé :

$$\begin{bmatrix} \phi_a \\ \phi_b \\ \phi_c \end{bmatrix} = \begin{bmatrix} L & M & M \\ M & L & M \\ M & M & L \end{bmatrix} \begin{bmatrix} I_a \\ I_b \\ I_c \end{bmatrix} + \begin{bmatrix} E_a(\theta) \\ E_b(\theta) \\ E_c(\theta) \end{bmatrix} \tag{4.5}$$

ϕ_a, ϕ_b, ϕ_c : Les flux traversant les phases a, b et c
I_a, I_b, I_c : Les courants dans chaque phase
E_a, E_b, E_c : Les fem à vide phase-neutre

La somme des courants étant nulle (en fonctionnement BDC et PMSM), on a :

$$\{V\} = R\{I\} + (L - M)\frac{d}{dt}\{I\} + \{E\} \tag{4.6}$$

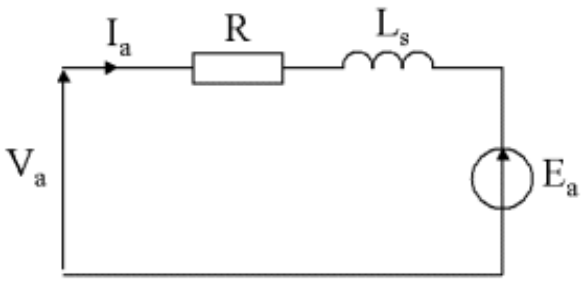


FIG. 4.4 – Modèle équivalent par phase

On obtient le schéma équivalent par phase de la figure 4.4 en posant $L_s = L - M$. L_s est alors l'inductance propre cyclique. Dans le cas des moteurs BDC, où deux phases conduisent à chaque instant, on donnera cependant les valeurs entre deux phases des inductances, des résistances et des tensions à vide (4.7).

$$\begin{aligned} L_{LL} &= 2(L - M) \\ R_{LL} &= 2R \\ E_{LL} &= 2E = k_E\Omega \end{aligned} \tag{4.7}$$

k_E est appelée constante de fem.

Modèle par phase	
Hypothèses	**Paramètres**
• Machine non saturée (Linéarité des propriétés du milieu)	R
• Pas de courants de Foucault	L_s
• **Pôles lisses**	E
• Les inductances propres sont égales	
• Les inductances mutuelles sont égales	
• La somme des courants est nulle	

TAB. 4.2 – Hypothèses et paramètres du modèle par phase

4.3.2 Le modèle dans la transformation de Park

En régime synchrone, on utilise plus souvent des rotors à aimants enterrés ou à aimantation orthoradiale (concentration de flux) car ils permettent d'obtenir une

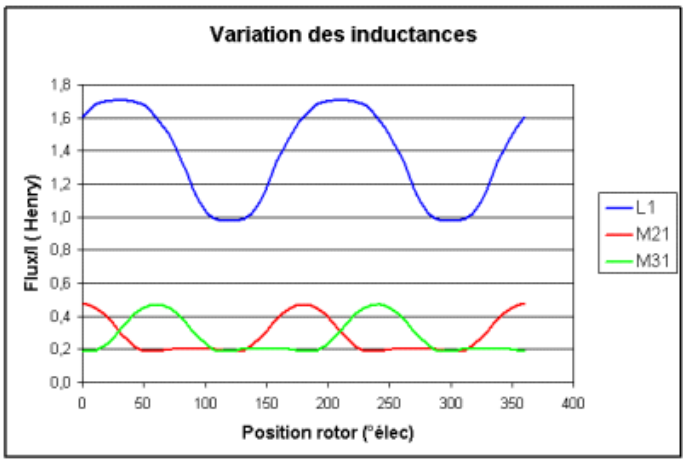


FIG. 4.5 – Inductances propres et mutuelles dans un moteur à pôles saillants

force électro-motrice sinusoïdale. Pour toutes ces topologies, l'hypothèse de rotor à pôles lisses ne tient plus et les inductances varient en fonction de la position du rotor (Figure 4.5). Pour prendre en compte l'effet de saillance des pôles, on introduit un terme supplémentaire dans la matrice inductance fonction de la position du rotor :

$$[L_s] = [L_{s0}] + [L_{s2}(\theta)] \tag{4.8}$$

Ce terme est basé sur une décomposition en série de fourier des inductances propres et mutuelles qui sont fonctions périodiques de θ (position en degrés électriques du rotor), de période π. En considérant uniquement le terme constant et l'harmonique de rang 2, en triphasé, on a :

$$[L_{s0}] = \begin{bmatrix} L & M & M \\ M & L & M \\ M & M & L \end{bmatrix}, [L_{s2}] = L_\nu \begin{bmatrix} \cos(2\theta) & \cos(2\theta - \frac{2\pi}{3}) & \cos(2\theta - \frac{4\pi}{3}) \\ \cos(2\theta - \frac{2\pi}{3}) & \cos(2\theta - \frac{4\pi}{3}) & \cos(2\theta) \\ \cos(2\theta - \frac{4\pi}{3}) & \cos(2\theta) & \cos(2\theta - \frac{2\pi}{3}) \end{bmatrix} \tag{4.9}$$

Dans la pratique, on utilise la tranformation de Park. Cette transformation consiste à exprimer les grandeurs dans un repère d'axes d et q lié à la position du rotor (Figure 4.6). Pour une grandeur λ quelconque, la transformation d'un système triphasé dans le modèle de park est donnée par :

$$[\lambda_{dq0}] = \frac{2}{3} \begin{bmatrix} \cos(\theta) & \cos(\theta - \frac{2\pi}{3}) & \cos(\theta - \frac{4\pi}{3}) \\ \sin(\theta) & \sin(\theta - \frac{2\pi}{3}) & \sin(\theta - \frac{4\pi}{3}) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} [\lambda_{abc}] \tag{4.10}$$

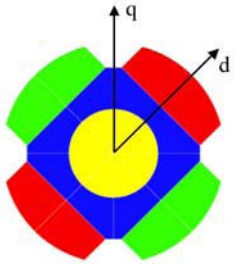


FIG. 4.6 – Axes d et q du rotor

En considérant des grandeurs sinusoïdales, et en exprimant les tensions à vide sous la forme :

$$\begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} = -\omega\phi_{pm} \begin{bmatrix} \sin(\theta) \\ \sin\left(\theta - \frac{2\pi}{3}\right) \\ \sin\left(\theta - \frac{4\pi}{3}\right) \end{bmatrix} \tag{4.11}$$

$\omega = p\Omega$	:	La pulsation électrique
p	:	Le nombre de paires de pôles
Ω	:	La vitesse du rotor

On obtient après transformation :

$$\begin{bmatrix} \phi_d \\ \phi_q \end{bmatrix} = \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \begin{bmatrix} I_d \\ I_q \end{bmatrix} + \phi_{pm} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{4.12}$$

avec

$$\begin{aligned} L_d &= (L - M) + \frac{3}{2}L_v \\ L_q &= (L - M) - \frac{3}{2}L_v \end{aligned} \tag{4.13}$$

L_d	:	Inductance synchrone d'axe directe
L_q	:	Inductance synchrone d'axe en quadrature

Les équations électriques de la machine dans le modèle de Park deviennent :

$$\begin{aligned} V_d &= RI_d + \frac{d\phi_d}{dt} - \omega\phi_q \\ V_q &= RI_q + \frac{d\phi_q}{dt} + \omega\phi_d \end{aligned} \tag{4.14}$$

Avec l'hypothèse de non saturation, les inductances sont constantes et on obtient le modèle équivalent de la figure 4.7.

$$\begin{aligned} V_d &= RI_d + L_d\frac{dI_d}{dt} - \omega L_q I_q \\ V_q &= RI_q + L_q\frac{dI_q}{dt} + \omega L_d I_d + \omega\phi_{pm} \end{aligned} \tag{4.15}$$

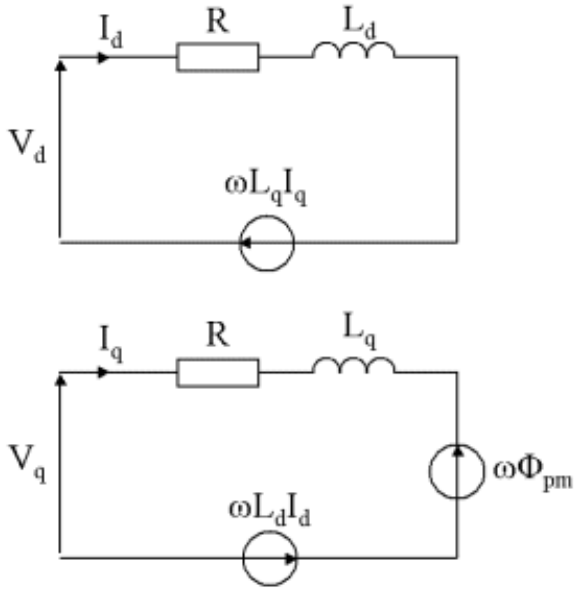


FIG. 4.7 – Modèle équivalent dans la transformation de Park

Dans ce modèle, le couple est donné par :

$$\Gamma_{em} = 3p\left[\phi_d I_q - \phi_q I_d\right] = 3p\left[\phi_{pm} I_q + \left(L_d - L_q\right) I_d I_q\right] \tag{4.16}$$

Le premier terme est appelé **couple synchrone dû aux aimants** ("magnet alignment torque"). Le second, appelé **couple reluctant** ("reluctance torque"), correspond à la variation de reluctance due à la saillance des pôles. Ces deux termes dépendent de la position relative du rotor par rapport à l'induction statorique, mais n'ont pas la même période (Figure 4.8).

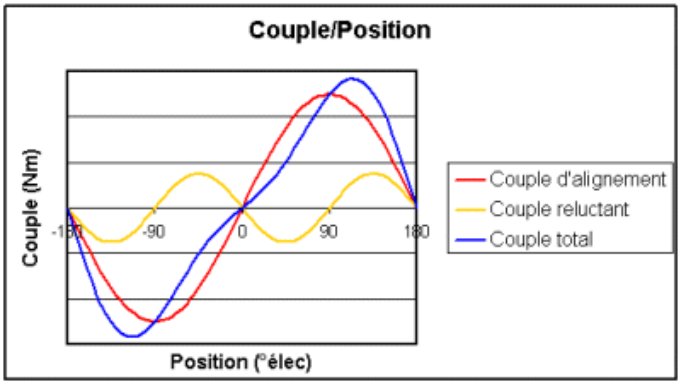


FIG. 4.8 – Couple dans le modèle de Park

Modèle dans la transformation de Park	
Hypothèses	**Paramètres**
• Machine non saturée (linéarité des propriétés du milieu)	L_d
• Pas de courants de Foucault	L_q
• **Grandeurs sinusoïdales (PMSM)**	ϕ_{pm}
• Les inductances propres sont égales	R
• Les inductances mutuelles sont égales	
• La somme des courants est nulle	

TAB. 4.3 – Hypothèses et paramètres du modèle dans la transformation de Park

4.3.3 La prise en compte d'amortisseurs

Lorsque le rotor contient des amortisseurs ("Damper windings") comme une cage d'écureuil (Figure 4.9) ou simplement lorsque l'on veut prendre en compte la résistivité des aimants et celle du circuit magnétique rotor, il convient d'affiner le modèle précédent [8] [15].

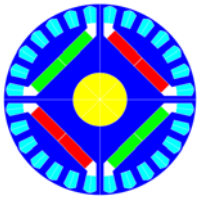


FIG. 4.9 – Exemple de rotor avec cage d'écureuil

En introduisant dans les équations les courants qui circulent dans les amortisseurs et les flux qui traversent les amortisseurs, on obtient :

$$\begin{bmatrix} \phi_d \\ \phi_q \\ \phi_{kd} \\ \phi_{kq} \end{bmatrix} = \begin{bmatrix} L_l + L_{md} & 0 & L_{md} & 0 \\ 0 & L_l + L_{mq} & 0 & L_{mq} \\ L_{md} & 0 & L_{md} + L_{kd} & 0 \\ 0 & L_{mq} & 0 & L_{mq} + L_{kq} \end{bmatrix} \begin{bmatrix} I_d \\ I_q \\ I_{kd} \\ I_{kq} \end{bmatrix} + \phi_{pm} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{4.17}$$

L_l : Inductance de fuite.

On ajoute également aux équations des tensions de phases (4.14), les tensions des amortisseurs en court circuit.

$$\begin{aligned} V_{kd} &= 0 = R_{kd} I_d + \frac{d\phi_{kd}}{dt} \\ V_{kq} &= 0 = R_{kq} I_q + \frac{d\phi_{kq}}{dt} \end{aligned} \tag{4.18}$$

On obtient le schéma équivalent de la figure 4.10.

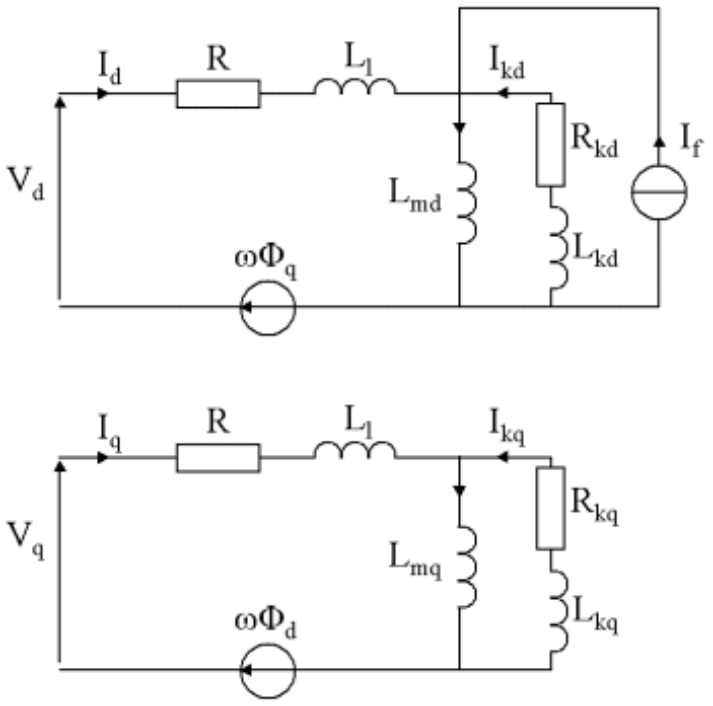


FIG. 4.10 – Modèle équivalent avec prise en compte des amortisseurs

Dans le schéma 4.10, les aimants permanents sont représentés par une source de courant dans l'axe d tel que $\phi_{pm} = L_{md} i_f$ au lieu d'une source de tension dans l'axe q. Cependant on retrouve ce flux dans la source de tension $\omega\phi_d$ de l'axe q. Les deux représentations sont donc équivalentes. Les conditions d'utilisation sont les mêmes que pour le modèle de Park du premier ordre sauf que cette fois-ci on prend en compte les courants de Foucault dans les amortisseurs, les aimants et le rotor massif.

Modèle avec amortisseurs	
Hypothèses	**Paramètres**
• Machine non saturée (linéarité des propriétés du milieu)	L_d, L_q
• **Grandeurs sinusoïdales (PMSM)**	L_{kd}, L_{kq}
• Les inductances propres sont égales	R, R_{kd}, R_{kq}
• Les inductances mutuelles sont égales	ϕ_{pm}
• La somme des courants est nulle	

Tab. 4.4 – Hypothèses et paramètres du modèle avec prise en compte des amortisseurs

4.4 Les essais normalisés

Nous désignerons par essais normalisés les simulations éléments finis réalisées pour l'étude de ces moteurs. On peut voir ces calculs comme les essais réalisés dans un laboratoire virtuel permettant à l'utilisateur d'extraire rapidement les caractéristiques et les performances de son moteur. Elles permettront notamment d'identifier les paramètres des schémas équivalents. Ces études correspondent aux essais décrits par les normes [3] [4], cependant elles comportent quelques différences. En effet ces normes ont été éditées principalement pour les machines synchrones à rotor bobiné. De plus les éléments finis permettent de faire abstraction de certains problèmes de mesures que l'on rencontre en laboratoire. Dans notre outil métier, elles seront automatisées à la demande de l'utilisateur à partir de quelques paramètres généraux et les résultats seront directement affichés. Un des atouts supplémentaires des éléments finis dans le dimensionnement est qu'ils permettent de visualiser les niveaux d'induction et les lignes de champ sur la géométrie, ce qui permet d'avoir une compréhension plus fine du comportement de la machine pendant l'essai.

4.4.1 L'essai à vide

L'essai à vide permet de déterminer les fem à vides d'une machine. Cet essai est essentiel dans le dimensionnement de la machine. D'abord selon la commande, on cherchera à avoir une fem trapézoïdale ou sinusoïdale. De plus en BDC, les tensions à vides entre phases sont proportionelles à la vitesse et on en déduit la constante de fem k_E.

$$E = k_E \Omega \tag{4.19}$$

En reprenant l'équation

$$EI = \Gamma_{em} \Omega \tag{4.20}$$

On obtient

$$\Gamma_{em} = k_E I = k_t I \tag{4.21}$$

La constante de fem donne donc une idée de la constante de couple. Enfin ces courbes sont souvent utilisées pour l'export système. En effet le couple étant obtenu par le produit de la fem par le courant, on cherche souvent à avoir une fem point par point pour rendre compte des ondulations de couple d'harmonique 6.

Ce calcul est réalisé dans Flux en magnétique transitoire sur une période électrique à vitesse imposée et avec couplage circuit. On place des résistances très grandes aux bornes des phases pour simuler un circuit ouvert et ainsi forcer les courants à zéro. On obtient la force électro-motrice en relevant les tensions aux bornes de chaque phase.

$$\{V\} = \frac{d}{dt} \{\phi_m(\theta)\}$$

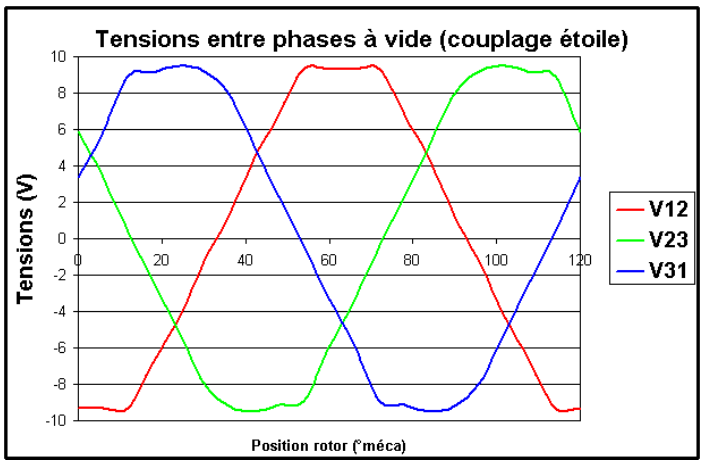


FIG. 4.11 – Courbes des forces électro-motrices

Dans le cas d'un couplage en triangle, on peut aussi mesurer le courant qui circule dans le triangle dû à un éventuel déséquilibre et aux harmoniques homopolaires. Dans le cas du couplage étoile, le potentiel du neutre renseigne le concepteur sur la présence de composantes homopolaires. Cet essai fait également l'objet d'un calcul des pertes fer à vide. Enfin, de manière optionnelle, on peut réaliser une étude paramétrique faisant varier l'induction rémanente de l'aimant. Ce calcul supplémentaire permet de tracer les courbes $\phi_{pm}(B_r)$, $k_E(B_r)$ qui peuvent être interessantes pour le dimensionnement des aimants.

Essai à vide	
Paramètres	**Résultats**
• Nombre de pas de calcul • Vitesse • Connection entre les phases • Option calcul paramétrique sur B_r	• Tensions et flux à vide • k_E et ϕ_{pm} • Pertes fer à vide • Dégradé induction et pertes fer • Lignes de champ

TAB. 4.5 – Paramètres et résultats de l'essai à vide

4.4.2 Le couple de détente

Cet essai permet de calculer le couple à vide. Lorsque les aimants tournent avec le rotor et passent devant les dents du stator, ils rencontrent une reluctance variable. Cette variation de reluctance entraîne un couple de valeur moyenne nulle que l'on appelle **couple de détente** ("cogging torque"). On rajoute un essai pour ce calcul qui aurait pu être fait pendant l'essai à vide, car la période du couple de détente ($\frac{360°méca}{ppcm(N_{pôles},N_{encoches})}$) est différente de la période électrique ce qui demande un pas angulaire différent. Il peut avoir un impact sur les ondulations de couple en fonctionnement nominal et peut entrainer un bruit gênant pour certaines applications. On cherchera donc à le minimiser.

Pour obtenir ce résultat, on réalise une étude multi-positions en magnéto-statique avec du vide dans les encoches. A chaque position, on relève le couple. Cet essai permet également de calculer l'induction dans l'entrefer et son carré donne une image des efforts à la surface des aimants et donne aussi une idée des vibrations que transmet la machine à son support.

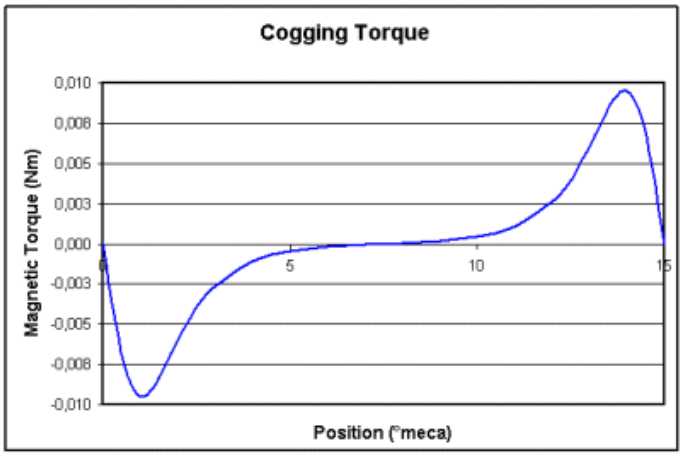


FIG. 4.12 – Courbe du couple de détente

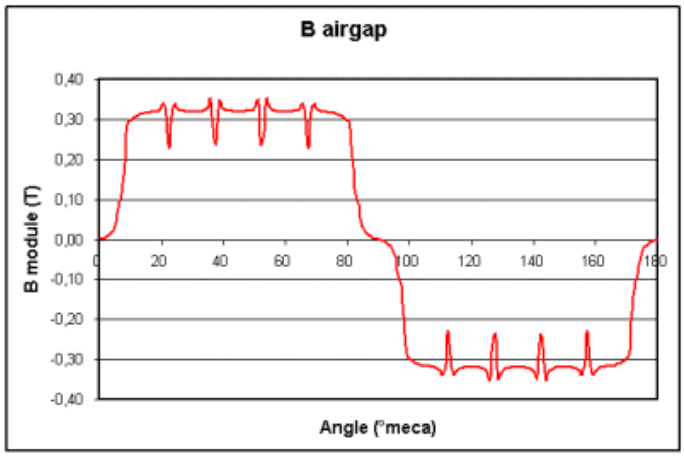


FIG. 4.13 – Courbe de l'induction dans l'entrefer

Couple de détente	
Paramètres	**Résultats**
• Discrétisation de la courbe	• Courbe du couple de détente. • Induction dans l'entrefer • Niveaux d'induction • Lignes de champ

TAB. 4.6 – Paramètres et résultats de l'étude du couple de détente

4.4.3 L'analyse statique

L'analyse statique consiste à réaliser un calcul en magnéto-statique à un point de charge donné. Ce calcul permet d'étudier le couple statique, mais aussi de calculer les inductances de la machine. Le point de charge peut être défini pour un fonctionnement en BDC ou en PMSM.

En BDC, deux phases étant alimentées à chaque instant. On définit alors :
– le courant nominal I
– la position du rotor β

En PMSM, où les phases sont parcourues par des courants sinusoïdaux, on définit :
– la position du rotor ($°méca$)
– l'amplitude des courants I
– l'angle de décalage β ($°élec$) entre l'axe d du rotor et l'induction statorique

A partir du calcul des flux dans les phases, on obtient l'inductance L_s du modèle par phase (4.22).

$$L_s = \frac{\phi_1 - \phi_{pm1}}{I_1} \tag{4.22}$$

ϕ_{pm1} : Flux traversant la phase 1 à vide (dû aux aimants). Il se déduit de l'essai à vide ou peut être réévalué à partir d'un calcul préliminaire à $I = 0$.
ϕ_1 : Flux total traversant la phase 1 (contribution des aimants et des courants)
I_1 : Courant dans la phase 1

Les Flux sont obtenus en intégrant le potentiel vecteur sur les surfaces traversées par les conducteurs.

$$\iint_{\Omega^+} A_1 d\Omega/\Omega^+ - \iint_{\Omega^-} A_2 d\Omega/\Omega^-$$

A_1 et A_2 : Les potentiels vecteurs
Ω^+ et Ω^- : Les surfaces orientées traversées par la bobine

En PMSM, en projettant les flux et les courants sur les axes d et q du rotor, on obtient les inductances dans la transformation de Park (4.23).

$$\begin{aligned} L_d &= \frac{\phi_d - \phi_{pm}}{I_d} \\ L_q &= \frac{\phi_q}{I_q} \end{aligned} \tag{4.23}$$

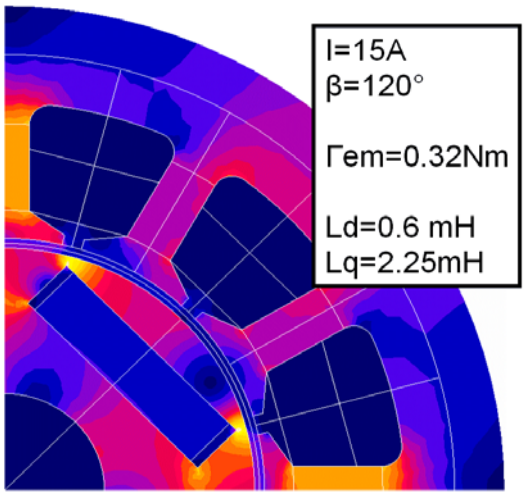


FIG. 4.14 – Dégradés de B

Ce calcul permet donc de déterminer le modèle équivalent au moteur dans la transformation de Park. Cependant pour étudier ces différents paramètres, on propose à l'utilisateur de faire varier l'amplitude des courants et/ou l'angle de décalage. On peut ainsi obtenir des surfaces de réponses comme $\Gamma_{em}(I, \beta)$, $L_d(I, \beta)$ et $L_q(I, \beta)$.

Etude avec variation de l'angle de pilotage

En PMSM, on fait varier l'angle β pour un courant donné. On obtient alors la courbe de couple en fonction de β. Cette courbe met en évidence la présence ou non d'un couple reluctant et permet de déterminer l'angle correspondant au couple maximal (Figure 4.15).

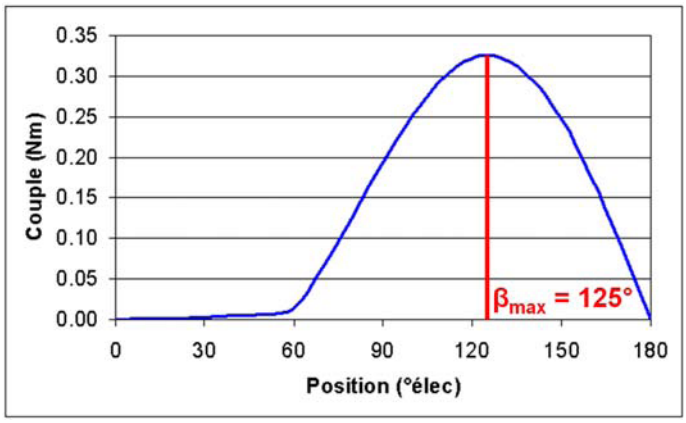


FIG. 4.15 – Courbe du couple en fonction de l'avance angulaire (°élec) de l'induction statorique sur le rotor

Etude avec variation du paramètre courant

Varier l'amplitude des courants permet de tracer la courbe de couple en fonction du courant et de visualiser l'effet de la saturation sur le couple (Figure 4.16). On en déduit la constante de couple $k_t = \frac{\Gamma_{em}}{I}$. De plus on peut évaluer l'influence de la saturation sur la valeur des inductances (Figure 4.17). De manière à évaluer correctement le comportement dynamique de la machine, les concepteurs de la commande cherchent souvent à avoir une bonne connaissance des inductances dynamiques, qui traduisent la capacité d'une variation de courant à fournir du flux . Elles ont la même valeur en régime linéaire que les inductances statiques, mais sont différentes en régime saturé (Figure 4.18).

$$L'_s = \frac{\Delta\phi_1}{\Delta I_1} \tag{4.24}$$

$$L'_d = \frac{\Delta\phi_d}{\Delta I_d} \tag{4.25}$$

$$L'_q = \frac{\Delta\phi_q}{\Delta I_q}$$

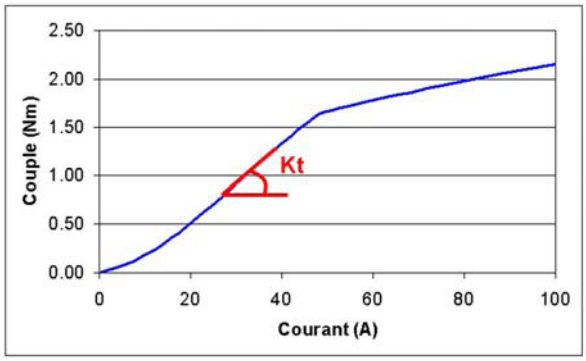


FIG. 4.16 – Courbe couple(courant) pour β=120°$élec$

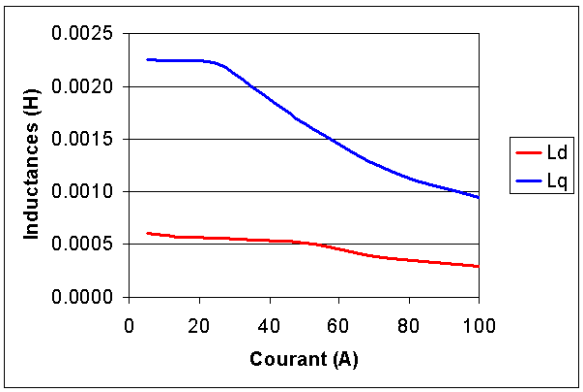


FIG. 4.17 – Courbes des inductances synchrones en fonction du courant pour β=120°$élec$

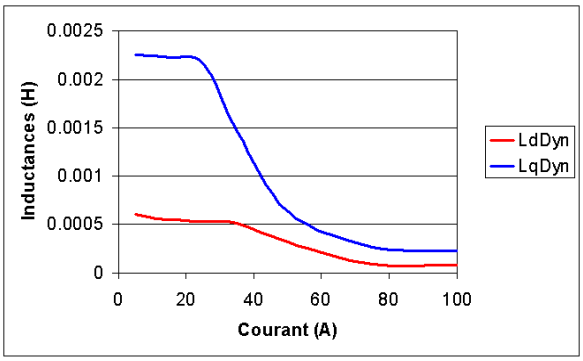


FIG. 4.18 – Courbes des inductances dynamiques en fonction du courant pour β=120°$élec$

Courbe du couple en fonction de la vitesse

A partir des courbes du couple et des inductances statiques en fonction du courant et de l'avance angulaire, on peut évaluer les courbes de couple(vitesse). En effet, l'onduleur délivre une tension qui est limitée par son alimentation en tension continue. Lorsque la vitesse augmente, les tension à vides augmentent et la commande, régulée en courant, ne peut plus délivrer la tension permettant l'établissement du courant requis pour obtenir le couple voulu. Cette tension maximale dépend donc de la tension continue et de la stratégie de commande. Une fois la tension maximale atteinte, lorsque la vitesse augmente, le courant et donc le couple diminuent.

On peut tracer la courbe couple(vitesse) à partir d'une valeur de tension maximale et de la courbe couple(courant). En effet considérant une tension maximale ($V_{\max}$), pour une vitesse donnée, en reprenant les équation du modèle de Park, on peut considérer, en régime établi, que :

$$\begin{aligned} V_d &= RI_d - \omega\phi_q \\ V_q &= RI_q + \omega\phi_d \end{aligned} \tag{4.26}$$

La tension est donc limitée par

$$\sqrt{V_d^2 + V_q^2} \leqslant V_{\max} \tag{4.27}$$

En partant d'un courant nominal, on peut écrire un algorithme qui augmente progressivement la vitesse. Pour une vitesse donnée, on regarde si le courant nominal peut être atteint. Si la condition (4.27) n'est pas respectée, on diminue le courant progressivement. Une fois la condition respectée, on obtient la valeur du courant maximum et le couple à partir de la courbe de couple/courant. A la fin de l'algorithme on obtient la courbe de couple/vitesse. Cette courbe peut être obtenue pour différents angles de pilotage (Figures 4.19 et 4.20).

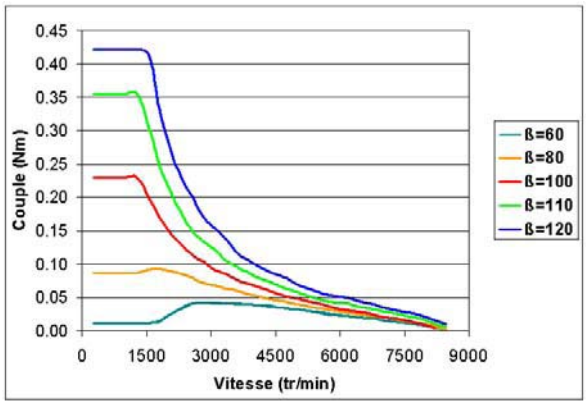


FIG. 4.19 – Courbe couple(vitesse) pour des angles de pilotage inférieurs à l'angle de couple maximal

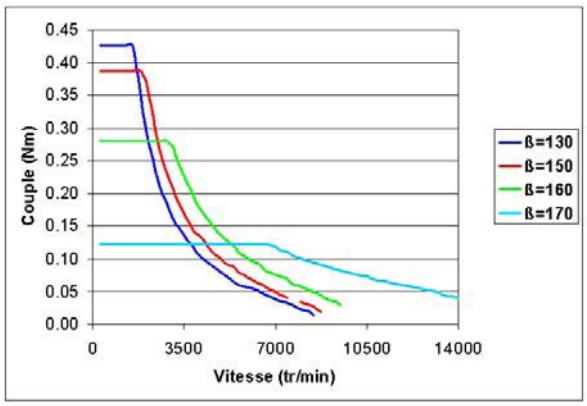


FIG. 4.20 – Courbe couple(vitesse) pour des angles de pilotages supérieurs à l'angle de couple maximal

On constate sur ces deux courbes (Figures 4.19 et 4.20) que l'on retrouve l'angle de pilotage maximal. En effet pour des angles de pilotage faibles, la courbe couple(vitesse) augmente avec l'angle. Une fois l'angle de pilotage maximal atteint, on constate que sur la plage de vitesse à courant constant, le couple diminue. Cependant, pour ces angles, la plage de vitesse est augmentée. Ce phénomène est lié au fait que l'on rajoute une composante au courant dans l'axe direct qui s'oppose à l'aimantation des aimants et donc qui diminue la fem, ce qui permet d'atteindre des vitesses plus grandes. Cependant cette zone de fonctionnement comporte des risques de désaimantation des aimants que l'on peut contrôler en visualisant le dégradé du champ H. Une stratégie de commande appelée défluxage consiste justement à rester à angle de pilotage maximal sur la plage à courant constant. Une fois cette vitesse obtenue, on garde cette fois ci le même module du courant mais on augmente l'angle de pilotage. Par algorithme, on peut obtenir, à partir de la courbe couple(position), la courbe couple(vitesse) avec défluxage (Figures 4.21, 4.22).

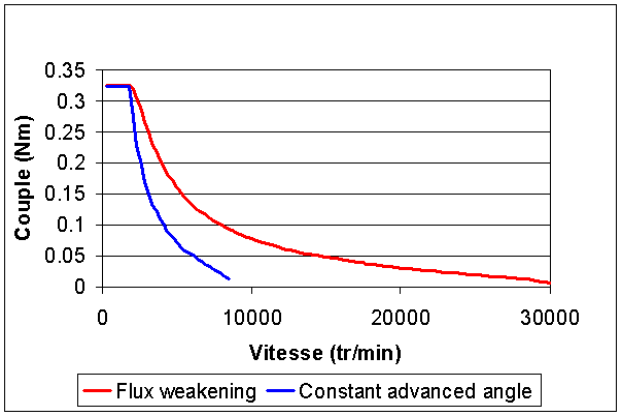


FIG. 4.21 – Courbe couple(vitesse) avec défluxage

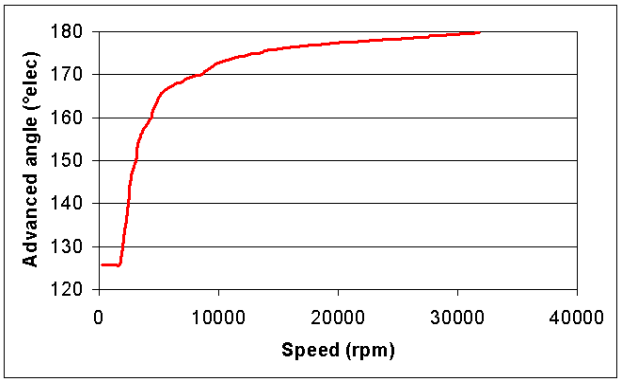


FIG. 4.22 – Evolution de l'angle de pilotage en fonction de la vitesse pour défluxage

Ces courbes couple(vitesse) permettent donc de dimensionner les inductances pour obtenir une plus grande plage de vitesse et pour discuter de l'intérêt ou non d'utiliser une stratégie de défluxage. Cependant elles sont basées sur la connaissance du couple statique et du schéma équivalent (inductances). L'augmentation de la vitesse et donc de la fréquence peut entraîner une augmentation des pertes qui ne sont pas prises en compte dans ce calcul. Pour une meilleure évaluation de ces courbes, il est nécessaire de réaliser une succession d'essais en charge à différentes vitesses.

Static analysis	
Paramètres	**Résultats**
• Alimentation BDC ou PMSM	• $\Gamma_{em}(I,\beta)$
• En BDC : I et β	• $\phi_{1,2,3,d,q}(I,\beta)$
• En PMSM : I, β et θ	• $L_{s,d,q}(I,\beta)$
• Variation du paramètre I	• $L'_{s,d,q}(I,\beta)$
• Variation du paramètre β	• Constante de couple
• Limitation en tension de l'onduleur	• $\Gamma_{em}(\Omega)$ avec et sans defluxage
• Limitation en courant	• Modèles équivalents par phase
• Essai à vide ou axes d et q	et dans la transformation de Park

TAB. 4.7 – Paramètres et résultats de l'essai de couple statique

4.4.4 L'essai en charge

C'est l'essai en régime nominal. Il consiste à réaliser un calcul en transitoire sur une période électrique à vitesse imposée avec alimentation des phases. Il permet de visualiser les ondulations de couple et de faire un bilan de puissance pour le calcul du rendement. Plusieurs types de commandes sont proposées à l'utilisateur. Ce sont les mêmes que celles proposées dans le logiciel analytique SPEED de notre partenaire. Il y a des commandes pour les moteurs PMSM et pour les moteurs BDC. On propose

d'alimenter les phases avec signaux idéaux ou avec régulation du courant en fonction de la position. Lorsque les signaux ne sont pas idéaux, on utilise un onduleur dans le couplage circuit et la limite en tension est prise en compte. La régulation amène de nouvelles harmoniques dans les signaux et permet de mieux évaluer le rendement. Cependant ces commandes demandent un grand nombre de pas de calcul.

Commandes BDC

Courants idéaux On alimente les phases avec des courants en créneaux idéaux (Figure 4.1). L'utilisateur définit l'amplitude et le déphasage. Par défaut, le courant commence à circuler dans la phase 1 à θ =30°*élec* où θ est la position du rotor par rapport à l'axe de la phase 1. Cette commande peut être utilisée quelque soit le nombre de phases. La courbe de la Figure (4.23) met en évidence les ondulations de couple d'harmonique 6, liées à la forme de la fem.

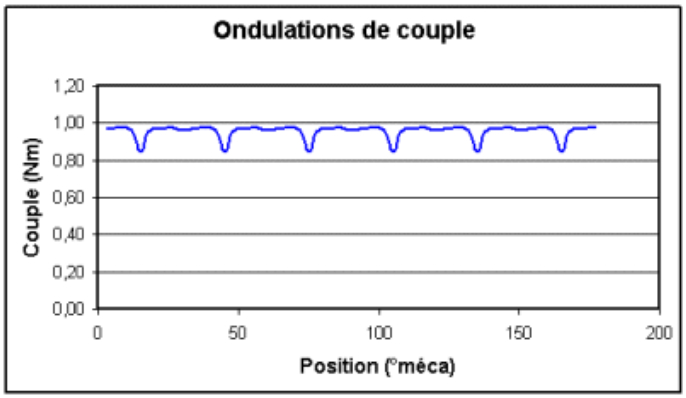


FIG. 4.23 – Ondulations de couple obtenues lors d'un essai en charge à 1000 tr/min avec courants idéaux

Régulation du courant par bande d'hystérésis En utilisant cette commande, on cherche à représenter l'onduleur. Ceci permet de prendre en compte la limitation en tension et une forme plus réelle des courants. On modélise donc l'onduleur dans le couplage circuit (Figure 4.24). Les intervalles de conduction des transistors en BDC sont représentés à la figure 4.25. De manière à réguler le courant, pendant chaque intervalle de 60°*élec* où deux transistors conduisent, l'un au moins des transistors doit commuter. La commutation est réalisée de manière à ce que les courants de ligne suivent la référence en courant dans une certaine bande d'hystérésis (Figure 4.26).

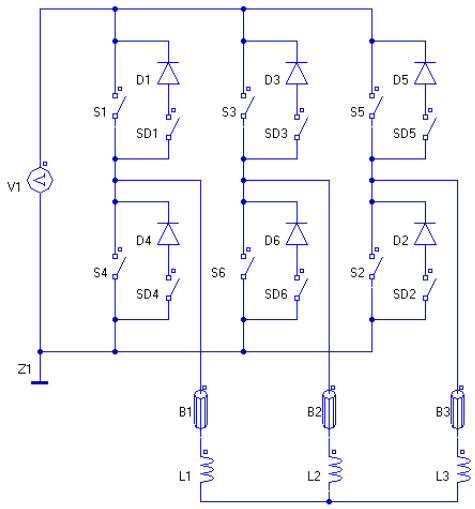


FIG. 4.24 – Couplage circuit

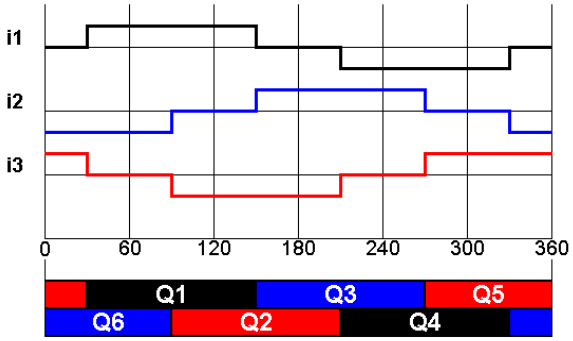


FIG. 4.25 – Intervalles de conductions des transistors en BDC

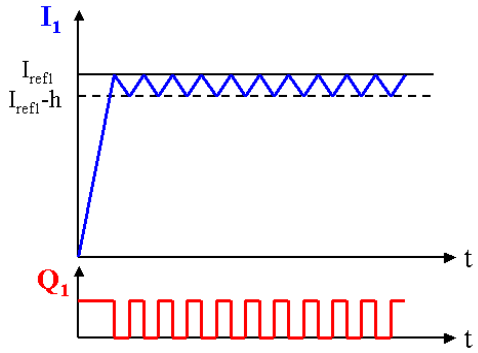


FIG. 4.26 – Régulation par bande d'hystérésis

Comme dans chaque période de 60°*élec* deux transistors peuvent commuter, on propose plusieures stratégies :

- Lorsque trois transistors seulement peuvent commuter (les trois du haut Q1, Q3, Q5 ou les trois du bas Q2, Q4, Q6), alors chaque transistor doit commuter pendant une période de 120°*élec*. Si ce sont les transitors du haut on appelera la stratégie de commande C120-Q1, si ce sont les transistors du bas on l'appelera C120-Q6.
- Lorsque tous les transistors peuvent commuter, chaque transistor commute pendant 60°*élec*. La commutation peut se dérouler pendant les 60 premiers degrés de l'intervalle de conduction du transistor, ou pendant les 60 derniers. Si ce sont les 60 permiers, on appele la stratégie C60-Q1, si ce sont les derniers, on appelle la stratégie C60-Q6.

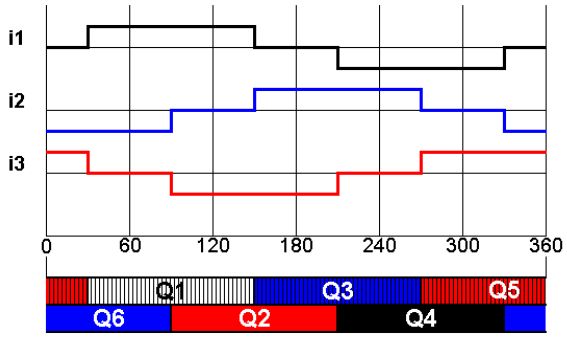


FIG. 4.27 – C120-Q1 : Les trois transistors du haut commutent pendant 120°*élec* chacun

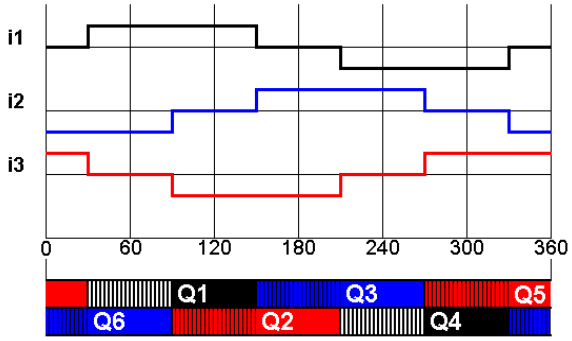


FIG. 4.28 – Tous les transistors commutent pendant les 60 premiers degrés de leur intervalle

Chaque commutation des transistors correspond à une commutation de la diode qui lui est associée. Par exemple lorsque Q1 s'ouvre pendant son interval de conduction,

la diode D4 conduit le courant. La commutation des transistors est calculée à chaque nouveau pas de calcul. Pour éviter que le courant dépasse la bande d'hystérésis, il est donc important d'avoir un nombre de pas de calcul suffisants.

Tension MLI S'il n'y a pas de capteur de courant, on peut utiliser une consigne en tension réalisée à l'aide d'une modulation de largeur d'impulsions. Dans ce cas l'utilisateur définit la fréquence des créneaux et la durée de conduction dans chaque créneau.

Commandes PMSM

Courants idéaux On alimente les phases avec des courants sinusoïdaux idéaux, l'utilisateur définit le nombre de pas de calculs, l'amplitude des courants et l'avance angulaire de l'induction statorique sur l'axe d du rotor. Dans ce cas la limitation en tension de l'onduleur n'est pas prise en compte.

Régulation du courant par bande d'hystérésis Les courants sont régulés indépendamment dans chaque phase pour suivre la référence sinusoïdale. L'utilisateur définit le nombre de pas de calculs, l'amplitude des courants, l'avance angulaire, la bande d'hystérésis et la limite en tension de l'onduleur. Comme les courants sont régulés indépendamment, on n'assure pas que la somme des courants est nulle à chaque instant.

Régulation du courant par le contrôle du vecteur tension Dans ce cas le courant est régulé en commutant le vecteur tension. En PMSM, à chaque instant, trois transistors conduisent. Les vecteurs tensions résultats sont représentés Figure 4.29.

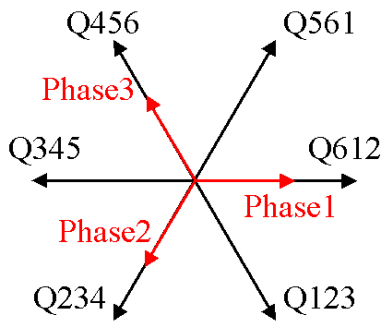


FIG. 4.29 – Les 6 vecteurs commandés

A partir de la référence en courant Iref1, Iref2, Iref3, la régulation du vecteur tension se déroule comme suit :

```
Si I1 > Iref1 :
    Si I2 > Iref2 :
        Vecteur Q456
    Sinon :
        Si I3 > Iref3 :
            Vecteur Q234
        Sinon :
            Vecteur Q345
Sinon :
    Si I2 > Iref2 :
        Si I3 > Iref3 :
            Vecteur Q612
        Sinon :
            Vecteur Q561
    Sinon :
        Vecteur Q123
```

Pour cette commande, l'utilisateur définit l'amplitude des courants, l'avance angulaire de l'induction statorique sur le rotor, la fréquence de commutation et la limite en tension de l'onduleur.

Six étapes Dans cette stratégie les transistors sont conducteurs pendant 180° consécutifs (Figure 4.30). Trois transistors conduisent à chaque instant.

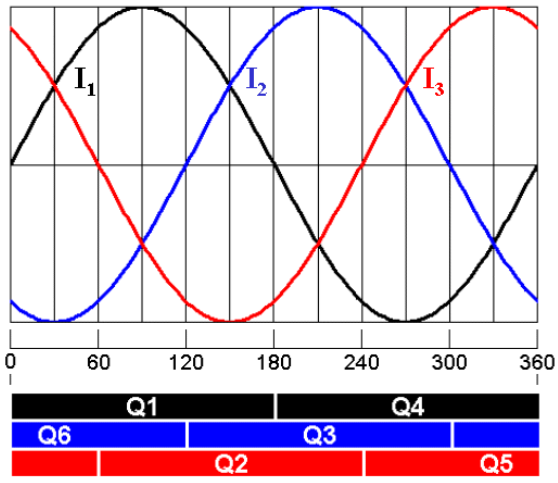


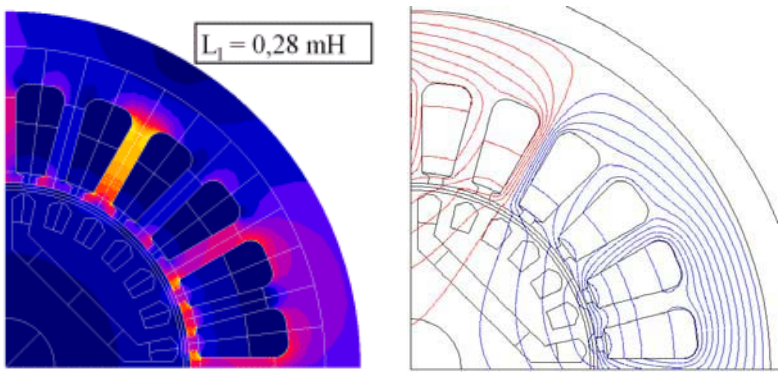
FIG. 4.30 – Intervals de conduction des transitors en PMSM

Essai en charge	
Paramètres	**Résultats**
• Commande • Vitesse	• Ondulations de couple • Courants et tensions • Bilan de puissance • Pertes fer • Rendement • Lignes de champs • Dégradé Induction et pertes fer

TAB. 4.8 – Paramètres et résultats de l'essai en charge

4.4.5 L'essai sans rotor

Cet essai permet de déterminer l'inductance de fuite du stator, mises à part les fuites de têtes de bobines qui ont déjà été évaluées de manière analytique. Cette inductance est utilisée dans le modèle de Park lorsque les aimants sont modélisés par une source de courant. L'étude consiste à réaliser un calcul en magnéto-statique à rotor enlevé (les régions du rotor sont assimilées à de l'air). On alimente les phases en triphasé et on calcule l'énergie dans tout le domaine. On obtient l'inductance de fuite à partir de $W = \frac{3}{2}LI^2$ [16].



TAB. 4.9 – Essai sans rotor - Dégradé de B et lignes de champ

Essai sans rotor	
Paramètres	**Résultats**
• Amplitude des courants	• Inductance de fuite
	• Dégradé induction
	• Lignes de champ

TAB. 4.10 – Paramètres et résultats de l'essai sans rotor

4.4.6 L'essai fréquentiel

Cet essai, que l'on peut retrouver dans les normes sous le nom de "*StandStill Frequency Response*", a pour objectif de réaliser une analyse fréquentielle des inductances synchrones afin d'identifier les paramètres du modèle dans la transformation de Park du second ordre. Ce modèle permet de prendre en compte la présence d'amortisseurs ou les courants de Foucault qui se développent dans les aimants ou un rotor massif. Pour le calcul dans Flux, on réalise une simulation en magnéto-harmonique paramétrique en faisant varier la fréquence entre 1mHz et 1kHz. Deux phases sont alimentées avec une tension simusoïdale d'amplitude $V = 5\%V_n$ et on aligne successivement l'axe d et q du rotor avec la phase A pour déterminer respectivement les inductances opérationnelles d'axe q et d.

Comme la simulation est réalisée en magnéto-harmonique, l'induction rémanente de l'aimant n'est pas prise en compte. On peut cependant utiliser l'initialisation en magnéto-statique pour prendre en compte l'effet de saturation des aimants. Lorsque l'axe q du rotor est aligné avec la phase A, on déduit l'inductance opérationnelle $L_d(p)$ des équations suivantes écrites dans la transformée de Laplace .

$$Z(p) = \frac{V(p)}{I(p)} = 2(R + pL_d(p)) \tag{4.28}$$

On trace ensuite les diagramme de bode des inductances opérationnelles $L_d(p)$ et $L_q(p)$.

Pour identifier les paramètres du modèle de Park au second ordre à partir de l'inductance opérationnelle, il faut reprendre les équations de ce modèle (4.14), (4.17), (4.18). On pose :

$$\begin{aligned} L_D &= L_l + L_{md} \\ L_Q &= L_l + L_{mq} \\ L_{KD} &= L_{md} + L_{kd} \\ L_{KQ} &= L_{mq} + L_{kq} \end{aligned} \tag{4.29}$$

A partir des équations (4.18), on peut exprimer I_{kd} en fonction de I_d (4.30).

$$\begin{aligned} I_{kd} &= \left(\frac{-L_{md}}{R_{kd} + pL_{KD}}\right) I_d \\ I_{kq} &= \left(\frac{-L_{mq}}{R_{kq} + pL_{KQ}}\right) I_q \end{aligned} \tag{4.30}$$

En remplaçant ces expressions dans l'équation des phases (4.14), on obtient :

$$\begin{aligned} V_d &= RI_d + pL_d(p)I_d - \omega L_q(p)I_q \\ V_q &= RI_q + pL_q(p)I_q + \omega L_d(p)I_d + \omega\phi_m \end{aligned} \tag{4.31}$$

Les expressions des inductances opérationnelles sont données par les équations (4.32).

$$\begin{aligned} L_d(p) &= L_D\frac{(1+\tau_d^{"}p)}{(1+\tau_{d0}^{"}p)} \\ L_q(p) &= L_Q\frac{(1+\tau_q^{"}p)}{(1+\tau_{q0}^{"}p)} \end{aligned} \tag{4.32}$$

avec

$$\begin{aligned} \tau_d^{"} &= \frac{L_{KD}}{R_{kd}}\left(1-\frac{L_{md}^2}{L_D L_{KD}}\right) \\ \tau_{d0}^{"} &= \frac{L_{KD}}{R_{kd}} \\ \tau_q^{"} &= \frac{L_{KQ}}{R_{kq}}\left(1-\frac{L_{mq}^2}{L_D L_{KQ}}\right) \\ \tau_{q0}^{"} &= \frac{L_{KQ}}{R_{kq}} \end{aligned} \tag{4.33}$$

$\tau_d^{"}$: Constante de temps subtransitoire longitudinale en court-circuit
$\tau_{d0}^{"}$: Constante de temps subtransitoire longitudinale à circuit ouvert
$\tau_q^{"}$: Constante de temps subtransitoire transversale en court-circuit
$\tau_{q0}^{"}$: Constante de temps subtransitoire transversale à circuit ouvert

De plus on définit :

$$\begin{aligned} X_d &= \omega L_D = \omega\left(L_l + L_{md}\right) \\ X_d^{"} &= \omega L_D\left(1-\frac{L_{md}^2}{L_D L_{KD}}\right) \\ X_q &= \omega L_Q = \omega\left(L_l + L_{mq}\right) \\ X_q^{"} &= \omega L_Q\left(1-\frac{L_{mq}^2}{L_Q L_{KQ}}\right) \end{aligned} \tag{4.34}$$

X_d : Réactance synchrone longitudinale
$X_d^{"}$: Réactance subtransitoire longitudinale
X_q : Réactance synchrone transversale
$X_q^{"}$: Réactance subtransitoire transversale

Cette expression théorique des inductances opérationnelles donne le diagramme de bode suivant :

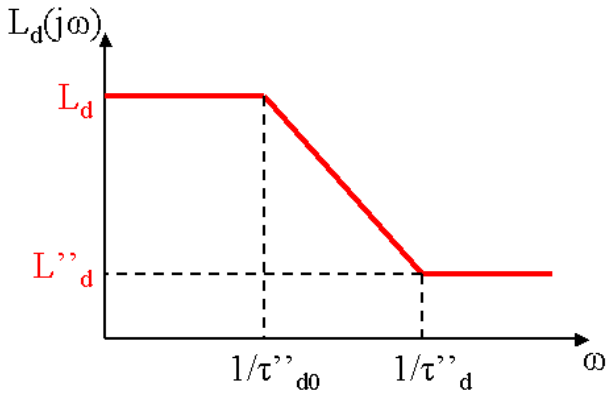


FIG. 4.31 – Module de l'inductance opérationnelle d'axe directe

Sur le diagramme de bode résultat de la simulation, on identifie les réactances et les constantes de temps. Les paramètres du modèle se déduisent des équations (4.35).

$$\begin{aligned} L_{md} &= \frac{X_d}{\omega} - L_l \\ R_{kd} &= \frac{L_{md}^2}{\tau''_{d0} - \tau''_d} \\ L_{kd} &= \tau''_{d0} R_{kd} - L_{md} \end{aligned} \tag{4.35}$$

On notera que l'inductance de fuite L_l est calculée à partir de l'essai sans rotor et le courant d'excitation i_f à partir de l'essai à vide.

StandStill Frequency Response	
Paramètres	**Résultats**
• Discrétisation logarithmique • Initialisation en magnéto-statique	• Diagrammes de bodes $L_d(p)$ et $L_q(p)$ • Réactances synchrones et subtransitoires • Constantes de temps • Schéma équivalent dq_dampers • Niveaux d'induction et Lignes de champ

TAB. 4.11 – Paramètres et résultats de l'essai fréquentiel

4.4.7 L'essai de court-circuit

Cet essai permet de simuler un court circuit entre les trois phases à la suite d'un essai à vide. Il permet d'observer l'intensité des courants de court-circuit et d'évaluer les risques de désaimantation des aimants. Concrètement on passe les valeurs des grandes résistances aux bornes de chaques phases de l'essai à vide à des valeurs très petites. L'expression des courants de court-circuit nous est donnée à partir de la théorie des régimes transitoires des machines synchrones (4.36) [2]. Pour les machines synchrones à rotor bobiné, cet essai est souvent réalisé en diminuant le courant d'exitation. Avec les moteurs à aimants permanents, cet ajustement n'est pas possible. Il y a donc ici un intérêt à simuler les défauts éventuels et étudier les risques de désaimantation des aimants.

$$\begin{aligned} i_A &= -E\left[\frac{1}{X_d}+\left(\frac{1}{X_d^{"}}-\frac{1}{X_d}\right)\exp\left(-\frac{t}{\tau_d^{"}}\right)\right]\cos(\omega t+\theta_0) \\ &\quad +\frac{E}{2}\exp\left(-\frac{t}{\tau_a}\right)\left[\left(\frac{1}{X_d^{"}}+\frac{1}{X_q^{"}}\right)\cos(\theta_0)+\left(\frac{1}{X_d^{"}}-\frac{1}{X_q^{"}}\right)\cos(2\omega t+\theta_0)\right] \\ \tau_a &= \frac{2X_d^{"}X_q^{"}}{\omega R(X_d^{"}+X_q^{"})} \end{aligned} \tag{4.36}$$

θ_0 : Angle entre l'axe d du rotor et la phase A au moment du court-circuit.

i_B et i_C sont obtenus en remplaçant θ_0 par $\theta_0 - \frac{2\pi}{3}$ et $\theta_0 - \frac{4\pi}{3}$.

4.5 Conclusion

Dans ce chapitre nous avons montré qu'en proposant à l'utilisateur des études classiques des moteurs BPM on peut simplifier l'utilisation des logiciels Flux et capitaliser du savoir-faire. Les études proposées permettront à l'utilisateur de dimensionner un moteur et de le caractériser rapidement. L'essai à vide permet de dimensionner la machine pour obtenir la forme et la constante de fem (proportionnelle à la constante de couple) voulues. Avec l'étude du couple statique, on détermine l'angle de charge maximal et on peut obtenir une estimation des courbes couple(vitesse). L'essai en charge permet de vérifier le couple et ses ondulations pour quelques points de fonctionnement. Le calcul des inductances permet de déterminer les circuits équivalents pour une utilisation dans la simulation système. Pour des moteurs avec armortisseurs, on peut utiliser l'essai fréquentiel. D'autres modèles équivalents pourront être rajoutés dans l'avenir, je pense notamment à des modèles qui prennent en compte les pertes fer par l'introduction dans le circuit équivalent d'une résistance [14] [7].

Dans cette partie, la personnalisation du logiciel Flux que nous mettons en place pour l'analyse des moteurs BPM a été présentée. Des outils équivalents seront développés pour d'autres types de machines comme les machines asynchrones, les machines à courant continu à aimants et les moteurs à reluctance variable. La stratégie est basée sur la capacité à définir dans Flux un moteur, ses méthodes d'analyses et ses résultats de manière métier. Dans la seconde partie je montrerai comment ont pu être réalisés ces développements dans Flux pour qu'ils soient évolutifs et réutilisables.

Deuxième partie

Concepts informatiques et implémentation logicielle

Chapitre 5

Modéliser un métier

5.1 Introduction

Dans la première partie, à partir d'une analyse des outils du concepteur de moteurs, les fonctionnalités d'un nouveau logiciel dédié à la conception des machines tournantes ont été définies. Ce logiciel a pour but de faciliter et accélérer l'accès aux méthodes éléments finis pour la simulation des machines électriques. Pour cela, il présente une IHM dédiée qui intégre les paramètres et les concepts manipulés par le concepteur (comme les topologies de moteurs, les caractéristiques et les performances classiques d'un moteur). Il doit également faciliter le lien avec les autres outils de la conception : il est couplé avec les outils analytiques du prédimensionnement et permet de créer des composants pour la simulation système. Enfin il permet de capitaliser les dimensionnements et les études réalisées pendant le processus de conception, et propose de les synthétiser sous forme de rapports. Dans cette partie, nous nous interesserons à son implémentation logicielle et aux concepts liés à la réalisation de logiciels métier.

Pour comprendre comment ce développement s'est mis en place, je présenterai dans ce chapitre la plateforme logicielle existante à Cedrat et au G2ELab pour la génération de logiciels métier. Mon objectif ici n'est pas de faire une description exhaustive de son fonctionnement. Je vais cependant essayer, en me plaçant d'un point de vue utilisateur, de dégager ses principales fonctionnalités et ses atouts. Pour des informations plus détaillées, je me réfère à la thèse de Yves Souchard [31]. Enfin les outils que nous avons développés dans cette plateforme pour personnaliser le logiciel généraliste Flux à un métier seront présentés.

5.2 La FluxFactory

5.2.1 Principe

Cedrat et le G2ELab ont commencé en 2002 un projet appelé FluxLab visant à développer une plateforme pour la création de logiciels métiers. Cette plateforme, ap-

pellée FluxFactory, a pour but de diminuer drastiquement les coûts de développement et de maintenance des logiciels de simulation et de garantir leur évolutivité. La plateforme développée est basée sur l'idée que l'on peut *modéliser un logiciel plutôt que le coder*. On peut retrouver cette notion dans la littérature sous le nom de Model Driven Architecture (Architecture pilotée par modèle) [30] [33] [34].

La plateforme est constituée de deux composants :
- *FluxBuilder* qui est un logiciel permettant de décrire le modèle d'un logiciel et de le sauvegarder sous forme de fichier.
- La *FluxCore* qui est une machine virtuelle. Elle interprète un modèle décrit dans FluxBuidler et exécute le logiciel correspondant. Le logiciel exécuté contient déjà, sur la base du modèle, une IHM complète, un langage de commande et la gestion de la base de données (création/édition/destruction de données, sauvegarde,...).

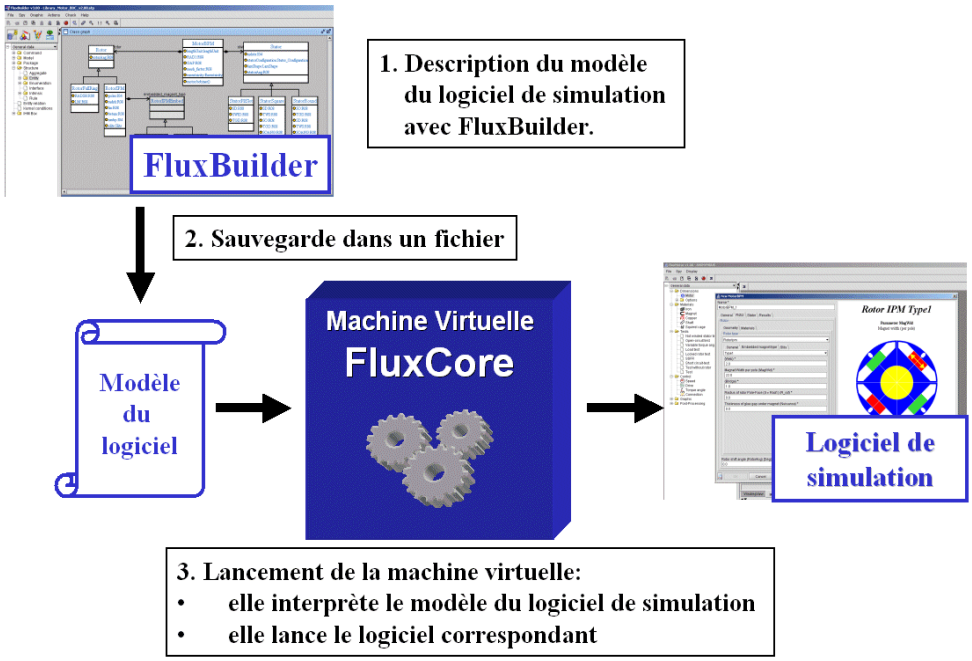


FIG. 5.1 – Le modèle d'un logiciel : description et exécution avec les composants de la FluxFactory

Cette architecture est particulièrement bien adaptée à la réalisation de logiciels métier. La réalisation d'un logiciel métier demande de repenser complètement l'interface et les concepts manipulés, or cette plateforme permet de créer une interface sans avoir à la coder. Le seul code réside dans l'implémentation des algorithmes métier. Pour mieux comprendre ce fonctionnement général, je présenterai de manière plus précise la façon de définir un modèle dans FluxBuilder.

5.2.2 La description du modèle de données

Le diagramme UML :

Pour décrire un logiciel, on commence d'abord par la définition de son modèle de données. Ce modèle de données correspond à la description des objets qui pourront être manipulés par l'utilisateur. Dans le monde du génie logiciel, le formalisme UML (Unified Modelling Language) [32] est reconnu et très utilisé. Notamment grâce au diagramme de classes, il permet de définir un modèle de données structuré, maintenable et évolutif. C'est donc sur la base de ce formalisme que l'on peut décrire et visualiser un modèle de donnée dans FluxBuilder.

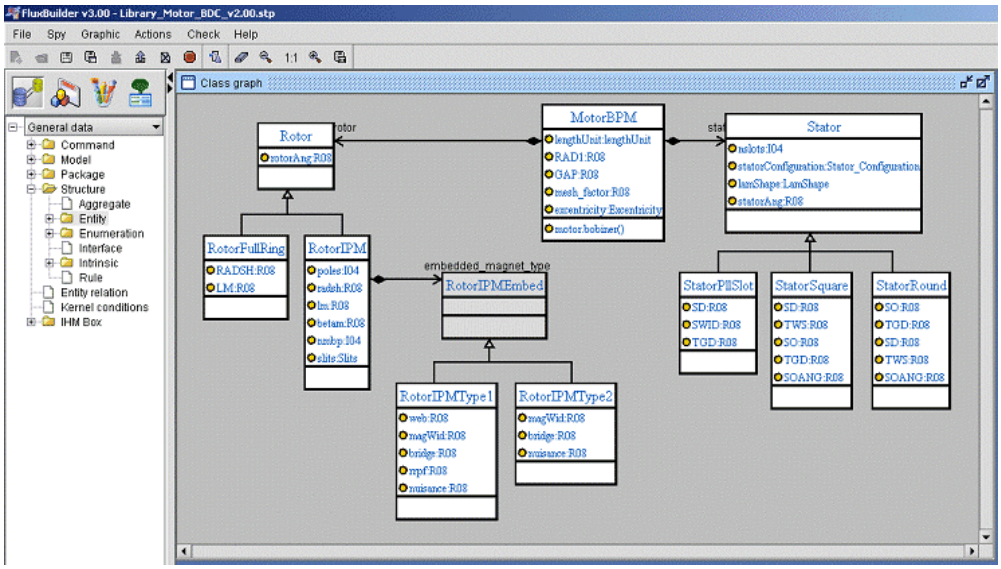


FIG. 5.2 – Visualisation du diagramme UML dans FluxBuilder

Les entités et les attributs :

Dans FluxBuilder, une entité décrit un objet ou une partie d'un objet que l'utilisateur pourra manipuler dans le logiciel de simulation. Elle contient un nom et des attributs. Ces attributs décrivent la structure des paramètres de l'objet. Prenons l'exemple du modèle d'un logiciel métier moteur. Il nous faut décrire comment l'utilisateur pourra définir un moteur dans le logiciel. Le moteur est constitué d'un rotor et d'un stator. Il nous faut donc définir une entité pour chacun de ces objets. L'entité "Rotor" par exemple permet de décrire les paramètres que devra renseigner l'utilisateur pour définir le rotor d'un moteur. Les attributs de l'entité "Rotor" sont : le nombre de pôles, le rayon extérieur ou le rayon de l'arbre (Figure 5.3). Ils portent un nom et

peuvent être de différents types : certains sont des entiers (Integer), d'autres des réels (Double) ou des chaînes de caractères (String). Certains sont de cardinalité multiple : un attribut peut être de type tableau de réels par exemple.

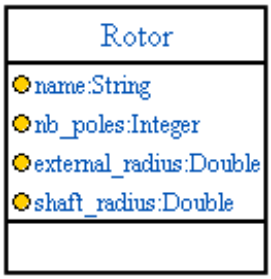
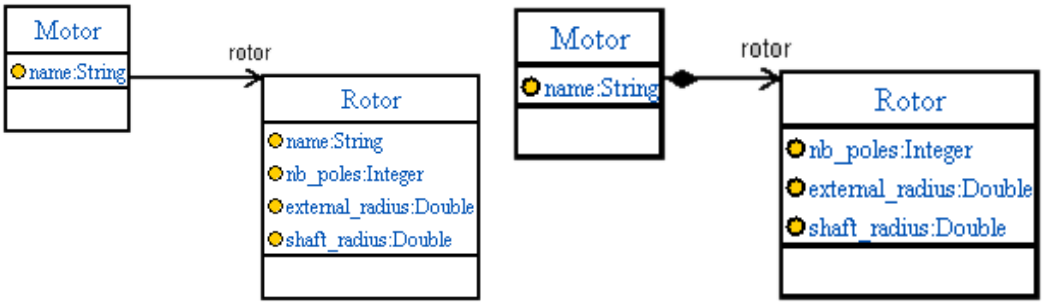


FIG. 5.3 – L'entité rotor et ses attributs

Certains attributs peuvent être des entités que le développeur a décrit précédemment. L'entité "Moteur" par exemple contient un attribut de type "Rotor". Un attribut de type entité peut être lié de deux façons différentes : par association ou par agrégation. Prenons l'entité "Moteur" qui a un attribut "Rotor". Dans le cas de l'agrégation, la définition du rotor est interne à celle du moteur. Dans le cas de l'association, la définition du rotor est externe à celle du moteur, c'est à dire que l'utilisateur peut définir de manière indépendante un rotor et ensuite l'associer à un moteur (Tableau 5.1).



TAB. 5.1 – L'attribut rotor de type entité : Association et aggregation

L'héritage :

Les entités peuvent également avoir des attributs de type entité abstraite. Une entité abstraite est une entité qui n'est pas en soit complètement définie. En effet, elle se dérive en plusieurs sous-types qui eux sont des entités concrètes. Par exemple, l'entité "Rotor" est abstraite et se décline en plusieurs topologies de rotor qui sont

concrètes, c'est à dire que l'on peut tracer. L'entité abstraite "Rotor" contient des attributs génériques qui sont communs à toutes les topologies comme le nombre de pôles. L'entité concrète dérivée, par exemple "Rotor à aimants enterrés", contient des paramètres spécifiques à cette topologie comme l'enfoncement des aimants. Cette déclinaison est appelée héritage en UML. On dit que l'entité "Rotor à aimants enterrés" hérite de l'entité "Rotor". Le diagramme de la Figure 5.4 illustre l'évolutivité du modèle de donnée et donc du logiciel à développer. En effet, pour ajouter une nouvelle topologie de rotor, il suffit d'ajouter une entité dérivant de l'entité rotor.

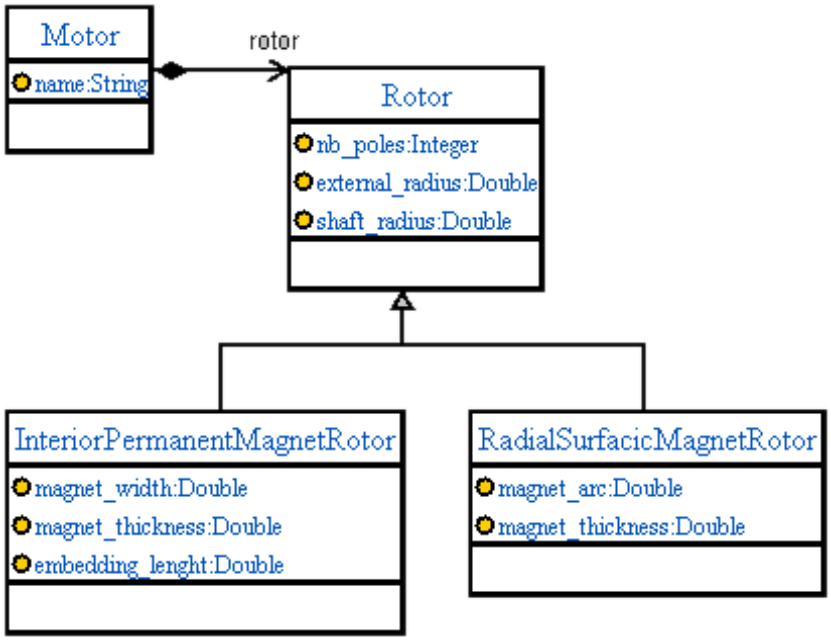


Fig. 5.4 – L'héritage UML pour modéliser les différentes topologies

Les méthodes et les commandes :

Sur chaque objet on peut définir des méthodes. Ces méthodes modélisent des comportements de l'objet et sont des commandes que l'on peut lui associer. Par exemple, sur l'objet moteur, on peut ajouter des méthodes comme "Mailler()" (Figure 5.5). Le modèle de donnée contient également la description des commandes que l'on retrouvera dans les menus et les barres d'icônes du logiciel que l'on cherche à décrire. Ces commandes sont décrites par un nom et des arguments. Par exemple, la commande "Import moteur SPEED" contient un argument qui est l'emplacement du fichier SPEED à importer.

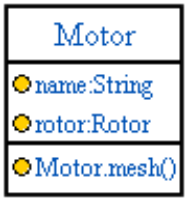


FIG. 5.5 – La méthode mesh() dans l'entité moteur

5.2.3 La description du modèle de présentation

Le modèle de données décrit donc les concepts métiers qui seront manipulés par l'utilisateur. Ce modèle de données conditionne énormément l'IHM du logiciel de simulation. La FluxCore se propose d'aller plus loin. Etant basée sur des composants informatiques réutilisables et configurables, elle permet de lancer automatiquement le logiciel correspondant au modèle décrit. Pour cela, la description UML a été enrichie. En plus du nom d'une entité ou d'un attribut, le développeur décrit son label anglais et français et son nom dans le langage de commande. Enfin, le développeur peut associer à chaque attribut ou entité une image et du texte pour définir une aide contextuelle. La machine virtuelle est alors capable d'interpréter ce modèle et de le retranscrire sous forme de boîtes de dialogue et de commandes textuelles. Par exemple en reprenant le modèle très simple de la figure 5.4, l'héritage UML qui correspond à un choix nécessaire de topologie de rotor est traduit automatiquement par la machine virtuelle dans l'IHM par un bouton de choix (Figure 5.6). On retrouve également les paramètres du modèle. Dans le modèle d'une entité, il est également possible de définir des onglets dans lesquels on pourra ranger les attributs.

Toute application construite avec la FluxFactory bénéficie immédiatement d'un langage de commande. Comme pour les boîtes de dialogues, ce langage est construit sur la base du modèle de données. Le langage utilisé est le python. Ce language est très utilisé pour le script et permet aussi la programmation objet. Toute action réalisée de manière graphique peut donc également être réalisée à l'aide d'une commande python (Figure 5.7). Aujourd'hui, pour être pilotable, un logiciel se doit d'avoir un langage de commande. L'avoir à moindre coût constitue un réel atout.

Une fois le modèle décrit, FluxBuilder propose également au développeur d'organiser l'IHM de son application. La description du modèle de données et des commandes et donc des boîtes de dialogue n'est pas suffisante, il faut maintenant décrire comment ces boîtes seront accessibles dans l'interface du logiciel. Puisque la plateforme est dédiée à la génération de logiciels de simulation, l'interface graphique se présente toujours de la même façon. Ainsi la FluxFactory propose au développeur un certain nombre de composants qui constitueront l'interface de son logiciel et dans lesquels il pourra placer son modèle de données :

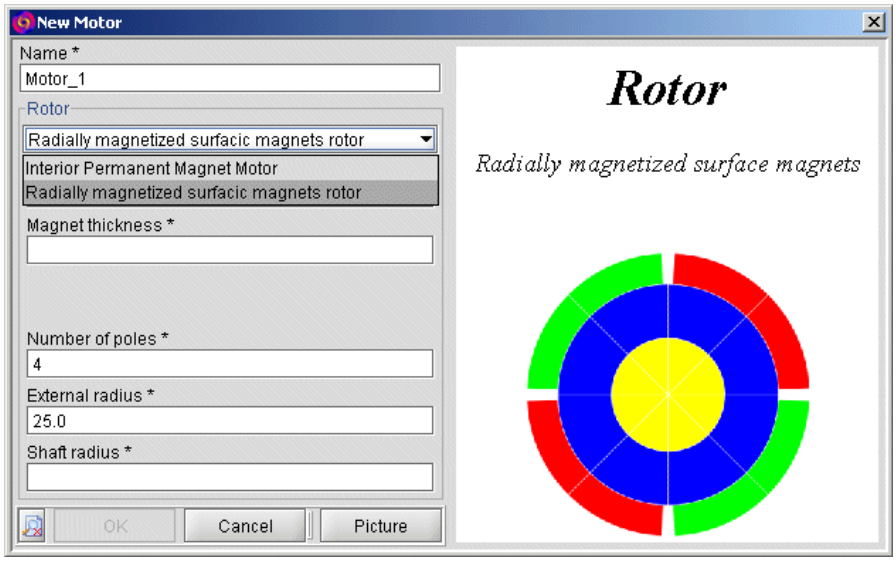


FIG. 5.6 – Boîte de création d'un moteur générée par la FluxCore à partir du modèle de la Figure 5.4

```
Motor(name='Motor_1',
      rotor=RadialSurfacicMagnetRotor(nb_poles=4,
                                      external_radius=25.0,
                                      shaft_radius=8.0,
                                      magnet_arc=120.0,
                                      magnet_thickness=5.0))
```

FIG. 5.7 – Commande python de création d'un moteur, basée sur le modèle de la Figure 5.4

- Un arbre qui permet de créer, modifier ou détruire des objets correspondant aux entités du modèle de données.
- Des menus permettant d'accéder aux méthodes des objets qui se trouvent dans l'arbre.
- Des menus et des barres d'icônes pour accéder aux commandes décrites dans le modèle de données.
- Des contextes d'utilisation qui ont chacun leur propre organisation de l'arbre et des menus.
- Une fenêtre pour copier et exécuter le langage de commande.

La dernière chose à décrire est le mode de représentation des données. Ceci peut être réalisé par ce que l'on a appelé des afficheurs. Il en existe plusieurs : la fenêtre graphique (représentant la géométrie, le maillage, les dégradés, etc...), les courbes, les rapports, les modèles UML. Ceux ci sont codés une fois pour toute dans la factory

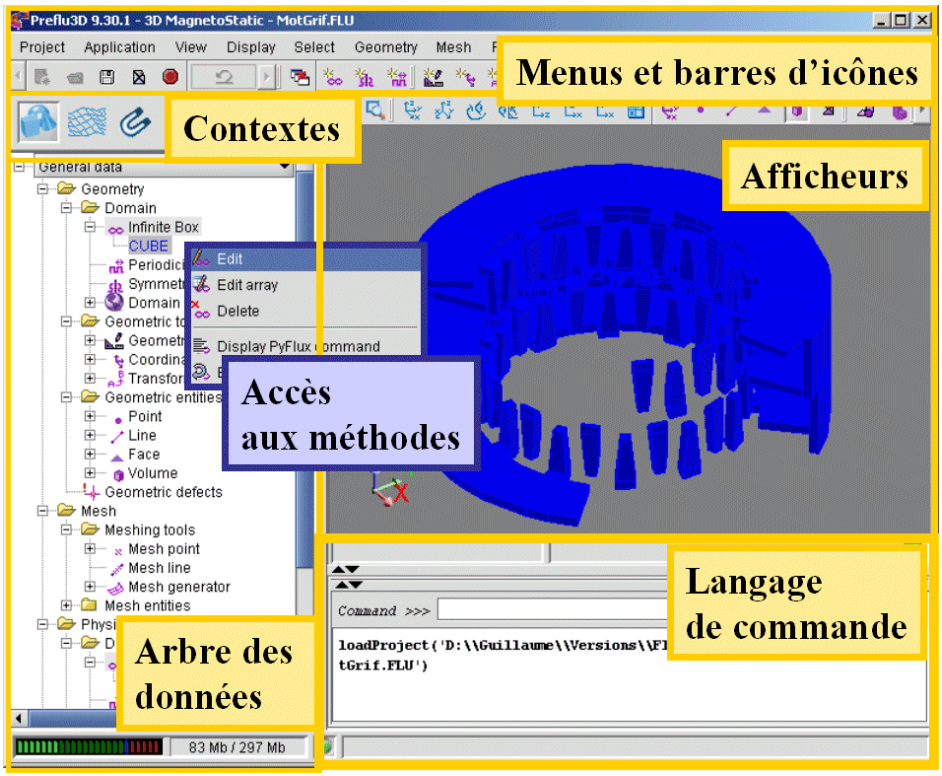


FIG. 5.8 – Composition de l'IHM dans les logiciels de la FluxFactory. (Exemple avec Flux)

et le développeur peut décider de représenter via ces afficheurs les données qui seront créées par l'utilisateur. Dans le cas de la fenêtre graphique, le développeur peut choisir des formes qu'il souhaite représenter (comme des points, des lignes, des faces, des dégradés). En plus de définir une forme, il indique où trouver les coordonnées de ces formes : directement dans les attributs d'une entité ou dans un algorithme dont il spécifie l'emplacement. C'est la même chose pour les courbes, le développeur ajoute un afficheur courbe dans un contexte. Ensuite il spécifie dans le modèle de données où il peut trouver les données relatives à ses courbes.

5.2.4 La machine virtuelle

Une fois le modèle décrit (aucune ligne de code n'a encore été écrite), il est interprété par la machine virtuelle qui le traduit dans le logiciel de simulation souhaité. Celui-ci contient déjà :

- Une IHM complète
- La possibilité de créer, modifier ou détruire des objets
- La persistance des données (sauvegarde et ouverture de projet)
- La gestion du langage de commande

– La représentation des données dans des afficheurs de graphiques 2D et 3D, de courbes, etc...

On comprend immédiatement que tout le code réside dans la machine virtuelle. Ce code est certes plus complexe qu'un code classique du fait de sa généricité mais il est codé une seule fois pour tous les logiciels. Cette méthode a donc plusieurs avantages :
– Le processus de développement est fortement diminué puisqu'un grand nombre de services et de composants sont déjà codés et sont configurables par modèle.
– L'évolutivité du logiciel est assurée grâce à une bonne structuration des données.
– Cette méthode de développement est un véritable atout dans l'approche métier. Sur la base des paramètres d'un métier, on peut immédiatement visualiser l'interface du logiciel. Dès la phase de spécification logicielle, face à un expert du métier qui n'est pas forcément familié avec les concepts UML, on peut lui montrer l'IHM et ainsi alimenter la discussion. En fait, FluxBuilder devient un outil de spécification avec le client ou l'expert métier.

Cependant, à ce stade du développement, la manipulation des objets dans le logiciel ne correpond qu'à de la création, modification et destruction de données. Les objets n'ont pas encore de comportement métier. De même les commandes que l'on peut retrouver dans les menus ou les barres d'icônes apparaissent mais ne produisent aucun effet. C'est pourquoi FluxBuilder permet de générer des templates (en Java ou en Fortran) qui correspondent au modèle de données et aux commandes. En remplissant ces templates, le développeur écrit les algorithmes métiers nécessaires au fonctionnement complet du logiciel. Ces templates sont générées en marge du code de la FluxCore car elles correspondent à un code spécifique qu'on appelle surcharge (Figure 5.9). Cette phase constitue la seule phase de codage à la main pour le développeur. Elle a donc été fortement réduite par rapport à une approche plus traditionnelle. C'est sur ce principe que Flux a été complètement refondu avec une implémentation des données en Fortran (langage historique de Flux). C'est également sur ce principe qu'a été développée l'interface du logiciel d'optimisation GOT. Cette fois l'implémentation a été réalisée en Java pour la richesse et la robustesse de ce language object. Il est amusant de noter que FluxBuilder lui aussi est construit sur ce principe, c'est à dire qu'il se décrit lui-même !

5.3 Les bibliothèques métier

5.3.1 Surcharger le modèle du logiciel Flux

La FluxFactory permet donc de générer rapidement de nouveaux logiciels de simulation. Cependant cette démarche demande encore un certain niveau de compétences informatiques. Dans l'approche métier, c'est d'abord le logiciel Flux que nous souhaitons personnaliser. Ceci afin de rendre son utilisation plus rapide et plus simple pour un métier donné. Devant le nombre de métiers auxquels le logiciel Flux peut

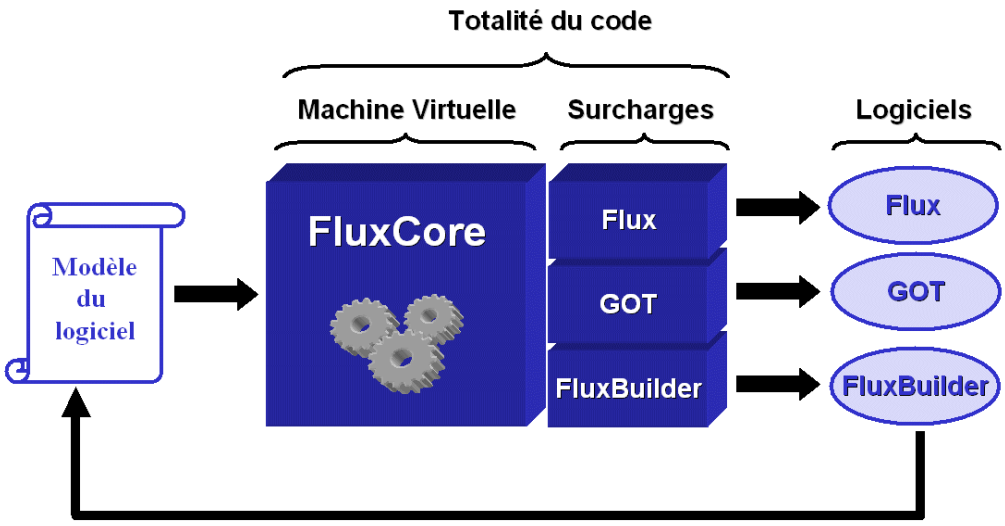


FIG. 5.9 – Réutilisabilité du FluxCore pour les logiciels Flux

répondre, on peut penser que de nombreuses personnalisations devront être développées dans l'avenir. C'est pourquoi nous souhaitons mettre en place une démarche de personnalisation du logiciel Flux qui soit la plus simple et la plus réutilisable possible. L'idéal serait de permettre à un utilisateur expérimenté de Flux de mettre en place lui-même la personnalisation de Flux liée à son métier.

Personnaliser le logiciel Flux signifie introduire dans le logiciel de nouveaux objets. Ces objets ont un comportement plus complexe que les objets standards de Flux et permettent d'automatiser un enchainement de tâches dans Flux souvent répétées par l'utilisateur. Dans le cas des moteurs électriques, des objets proposent des topologies de moteur et automatisent leur construction dans le logiciel. Le maillage est automatique et des méthodes d'analyse classiques sont proposées.

Flux étant construit sur le principe de la FluxFactory, il est décrit par un modèle. Pour personnaliser Flux et l'enrichir de nouveaux objets, nous devons donc enrichir le modèle de Flux, existant. En ajoutant dans le modèle de Flux le modèle des objets moteurs, des études et des résultats et en les plaçant dans l'arbre d'un nouveau contexte dédié, on pourra, suite à une commande, ouvrir ce contexte et présenter à l'utilisateur de Flux une toute nouvelle interface dédiée à l'étude des moteurs. Cependant, cette solution n'est pas encore tout à fait satisfaisante. En effet, avec la stratégie présentée, le modèle du logiciel Flux pourrait devenir très important et difficilement maintenable. De plus, souhaitant donner accès à cette démarche de personnalisation au plus grand nombre, le développement des interfaces métier ne doit pas se situer au niveau du modèle de Flux qui relève du domaine réservé des développeurs de Flux. Nous proposons donc que le modèle dédié à un métier soit décrit séparément et chargé dynamiquement dans Flux.

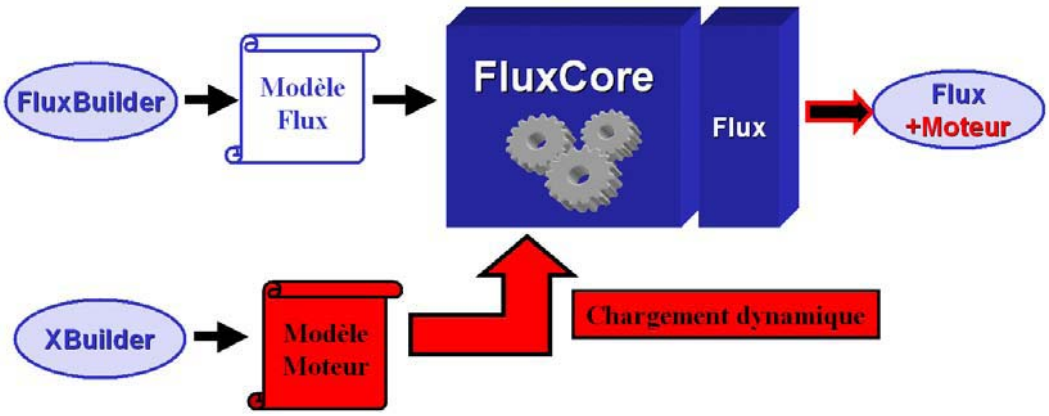


FIG. 5.10 – Schéma de principe du chargement d'une extension dans la FluxFactory

Nous avons créé dans Flux un contexte vide et une commande permettant de charger un modèle métier et de le placer dans ce contexte vide. Une fois le modèle métier choisi par l'utilisateur (Figure 5.11), le nouveau contexte est ouvert et une nouvelle interface apparaît avec de nouveaux objets dédiés à l'étude des moteurs (Figure 6.7). Ainsi la description d'un modèle peut se faire séparément et n'est plus liée au développement du logiciel Flux. Cette notion de modèle métier que l'on peut charger dans Flux est appelée *contexte métier*. A tout moment, on peut quitter le contexte métier et retourner dans un contexte de Flux pour des possibilités de géométries, d'études ou d'exploitations illimitées.

5.3.2 Implémenter ce modèle dans le language de commande de Flux

Pour définir le comportement dans Flux des nouveaux objets métier, nous devons également prévoir une implémentation de leur modèle. Cette implémentation sera chargée dans Flux en même temps que le modèle métier. Elle contiendra la description de l'enchainement des commandes Flux à réaliser à partir des valeurs des objets métiers créés pour automatiser la construction de la géométrie, effectuer une étude ou afficher des résultats. Toujours parce que nous souhaitons que le développement de ces extensions soient accessibles au plus grand nombre, nous avons introduit dans la FluxFactory une implémentation des objets métiers dans le langage de commande de Flux à savoir Python. En effet, Flux étant construit suivant le principe de la FluxFactory, toute action dans le logiciel est reproductible à l'aide d'une ligne de script Python, ce langage est donc idéal pour décrire le comportement des objets métier. De plus, ce langage est bien connu des utilisateurs avancés de Flux, qui n'auront ainsi pas de problèmes pour développer leur propres extensions.

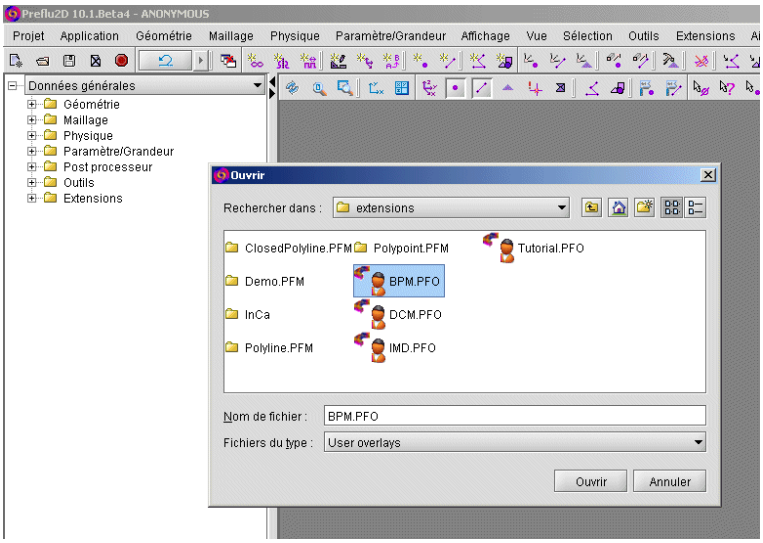


FIG. 5.11 – Commande permettant de charger un contexte métier dans Flux

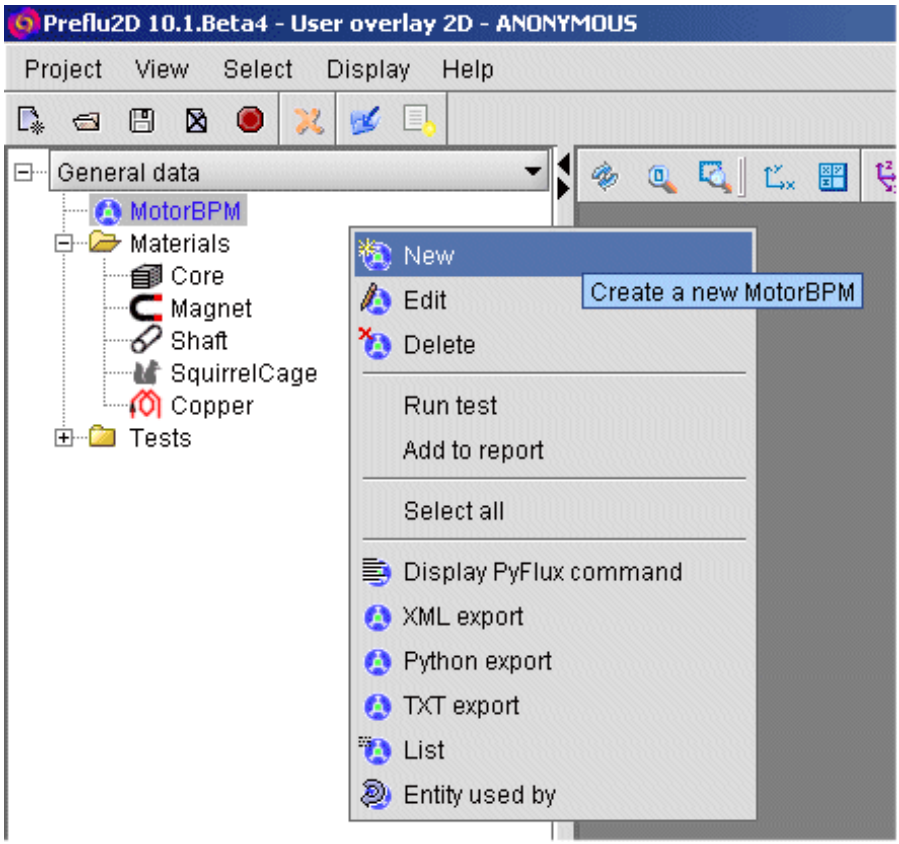


FIG. 5.12 – Interface dédiée à l'étude des moteurs dans Flux

Les templates

Après avoir défini le modèle de l'extension, on génère des templates python. Une classe python est créée par entité présente dans l'arbre. Nous avons souhaité une structuration des templates la plus simple possible. Ces classes sont composées de quatre méthodes à remplir en Python permettant d'associer un comportement (c'est à dire des commandes Flux) à la création, modification et destruction des objets métiers (Figure 5.13).

```python
class FMotor:

    def __init__(self):
        pass

    ################################################################################
    # Method:       # check(self, model)
    # Purpose:      # check the user parameters values
    # Parameters:   # model: the user parameter values
    # Returns:      # None
    ################################################################################
    def check(self, model):
        pass

    ################################################################################
    # Method:       # build(self, model)
    # Purpose:      # define the actions to do in Flux during the object's creation
    # Parameters:   # model: the user parameter values
    # Returns:      # None
    ################################################################################
    def build(self, model):
        pass

    ################################################################################
    # Method:       # modify(self, model, changes)
    # Purpose:      # define the actions to do in Flux during the object's modifications
    # Parameters:   # model   : the user parameter values
    #               # changes : the user parameter changes
    # Returns:      # None
    ################################################################################
    def modify(self, model, changes):
        pass

    ################################################################################
    # Method:       # delete(self)
    # Purpose:      # define the actions to do in Flux during the object's deletion
    # Parameters:   # None
    # Returns:      # None
    ################################################################################
    def delete(self):
```

FIG. 5.13 – Template python générée

La méthode *check(model)* Appelée par la FluxCore avant chaque création ou modification d'un objet métier, la méthode *check()* vérifie si les valeurs des paramètres

renseignés par l'utilisateur (qui sont fournis en argument de la méthode dans l'objet *model*) sont corrects et n'introduisent pas de problèmes. Une erreur avec un message expliquant le ou les paramètres qui posent problème peut être affichée à l'utilisateur et la création ou la modification de l'objet n'est pas réalisée. Pour les objets moteur, cette méthode contient un ensemble de règles géométriques assurant une bonne construction. Elles vérifient notamment que les côtes indiquées n'entrainent pas d'intersection de ligne ou d'incohérences comme un rayon de l'arbre plus grand que le rayon extérieur du rotor. Ces règles sont très importantes car elles assurent à l'utilisateur un bon fonctionnement du logiciel et le guident quant aux erreurs qui surviennent. Pour le développement des objets moteur, pour s'assurer que ces règles définissent correctement l'ensemble des moteurs traçables, nous avons mis en place des procédures de tests aléatoires qui remplissent la boîte moteur et tente de construire la géométrie. Si la géométrie ne se trace pas et qu'aucun message d'erreur n'est apparu, on détecte qu'il manque une règle de géométrie.

```
################################################################################
# Method:       # check(self, model)
# Purpose:      # check the user parameters values
# Parameters:   # model: the user parameter values
# Returns:      # None
################################################################################
def check(self, model):

    if model.rotor.external_radius <= 0:
        raise UserObjectValidityException("External rotor radius must be positive")

    if model.rotor.shaft_radius <= 0:
        raise UserObjectValidityException("Shaft radius must be positive")

    if model.rotor.magnet_thickness <= 0:
        raise UserObjectValidityException("Magnet thickness must be positive")

    if model.rotor.external_radius <= model.rotor.shaft_radius + model.rotor.magnet_thickness:
        raise UserObjectValidityException("External rotor radius is too small")
```

FIG. 5.14 – Exemple de code d'une méthode check

La méthode *build(model)* Cette méthode contient le code Python des actions à réaliser dans Flux à la création de l'objet. Pour les objets moteur, cette méthode contient le code python permettant de construire la géométrie et le maillage du moteur à partir des valeurs des paramètres contenus dans *model*. Si le nombre de lignes de code est important, une méthode comme build() peut faire appel à d'autres méthodes ou d'autres objets Python créés par le développeur. On peut donc structurer le code Python et ainsi le rendre plus lisible et plus maintenable (Figure 5.15).

```
################################################################################
# Method:       # build(self, model)
# Purpose:      # define the actions to do in Flux during the object's creation
# Parameters:   # model: the user parameter values
# Returns:      # None
################################################################################
def build(self, model):

    self.createRotorParameters()
    self.createRotorCoordinateSys()
    self.createRotorPoints()
    self.createRotorLines()
    self.createRotorRegionFaces()
    buildFaces()
    self.createMeshPoints()
    self.applyRotorSymmetries()

################################################################################
# Method:       # createRotorPoints(self)
# Purpose:      # create rotor points
# Parameters:   # None
# Returns:      # None
################################################################################
def createRotorPoints(self):
    self.PR1=self.createPoint('0','0')
    self.PR2=self.createPoint('RADSH','0')
    self.PR3=self.createPoint('RAD1-LM','0')
    self.PR4=self.createPoint('RAD1','0')
    self.PR5=self.createPoint('RADSH','-PHI')
    self.PR3=self.createPoint('RAD1-LM','-PHI')
    self.PR4=self.createPoint('RAD1','-PHI')
```

FIG. 5.15 – Exemple de code de la méthode build

La méthode *delete()* Cette méthode contient le code Python des actions à réaliser dans Flux pour la destruction d'un objet. Pour les objets moteur, il s'agit de détruire la géométrie créée en détruisant par exemple, le repère de définition de la géométrie (Figure 5.16).

La méthode *modify(model, changes)* Cette méthode contient le code à réaliser lors d'un changement de paramètres. Bien sûr l'approche la plus simple pour gérer une modification consiste à détruire et reconstruire l'objet en faisant appel successivement aux méthodes *delete()* et *build()*. Cependant, la FluxCore fournit également en argument l'objet *changes* qui contient les demandes de changements élémentaires qui ont été souhaités. Ainsi on peut modifier de manière plus fine la géométrie. On peut par exemple simplement modifier la valeur d'un paramètre géométrique au lieu de tout reconstruire afin d'optimiser le temps du changement (5.17).

```
##########################################################################
# Method:       # delete(self)
# Purpose:      # define the actions to do in Flux during the object's deletion
# Parameters:   # None
# Returns:      # None
##########################################################################
def delete(self):

    #delete coord system
    CoordSys['ROTOR_COORD'].deleteForce()

    #delete MeshPoints
    self.deleteMeshPoints()

    #delete Parameters
    self.deleteRotorParameters()
```

FIG. 5.16 – Exemple de code de la méthode delete

```
##########################################################################
# Method:       # modify(self, model, changes)
# Purpose:      # define the actions to do in Flux during the object's modifications
# Parameters:   # model   : the user parameter values
#               # changes : the user parameter changes
# Returns:      # None
##########################################################################
def modify(self, model, changes):

    self.deleteMesh()

    for change in changes:

        if change.isConcerning('external_radius') :
            ParameterGeom['ROTOR_EXTERNAL_RADIUS'].expression = str(change.getNewValue())

        elif change.isConcerning('shaft_radius') :
            ParameterGeom['SHAFT_RADIUS'].expression = str(change.getNewValue())

        elif change.isConcerning('nb_poles') :
            self.deleteRotorSymmetries()
            ParameterGeom['NB_POLES'].expression = str(change.getNewValue())
            self.applyRotorSymmetries()

    self.meshFaces()
```

FIG. 5.17 – Exemple de code de la méthode modify

Les commandes

Des templates pour les commandes sont également générées. Elles se présentent sous la forme d'un fichier qui contient une fonction Python correspondant à la commande et ses arguments.

5.3.3 Un générateur de bibliothèques métiers : XBuilder

Le modèle d'un contexte métier est beaucoup plus léger que celui d'une application complète. En effet le modèle est limité à la description du modèle de données métier, des commandes, des menus et de l'arbre qui seront chargés dans le contexte vide de Flux prévu à cet effet. Il est inutile de décrire des afficheurs ou un modèle graphique puisqu'ils existent dans Flux. C'est pourquoi un logiciel spécifique, appelé XBuilder, a été développé pour décrire ces modèles métiers. Basé sur les compétences de FluxBuilder, ce logiciel est beaucoup plus simple d'accès et ne contient que les informations strictement nécessaires. Il bénéficie également d'un éditeur Python pour remplir les templates sous une seule interface de développement (Figure 5.18). Cedrat envisage de commercialiser ce logiciel prochainement, ce qui permettra de proposer aux utilisateurs avancés de Flux de développer leur propre personnalisation de Flux.

5.3.4 Sauvegarde

Pour avoir une intégration totale de ces objets métiers dans un projet Flux nous avons travaillé sur la sauvegarde de ces objets. Au chargement d'un contexte métier, un objet est créé. Celui-ci contient l'emplacement du modèle et du code correspondant au contexte métier. De plus à la sauvegarde, les objets métiers créés sont sérialisés et stockés dans cet objet. Ainsi à l'ouverture d'un projet on charge de manière habituelle le projet Flux, ensuite on lit l'objet ce qui permet de recharger le contexte et de désérialiser les objets métiers.

5.4 Conclusion

J'ai donc présenté brièvement le fonctionnement de la plateforme logicielle FluxFactory. Elle permet, par la définition d'un modèle basé sur le langage UML, de générer rapidement l'interface et la gestion des données d'un logiciel de simulation. Elle permet également une implémentation java ou fortran des données. Le logiciel Flux est construit sur ce principe. Cette plateforme assure à nos logiciels, la réduction des coûts de développement et de maintenance et une bonne évolutivité.

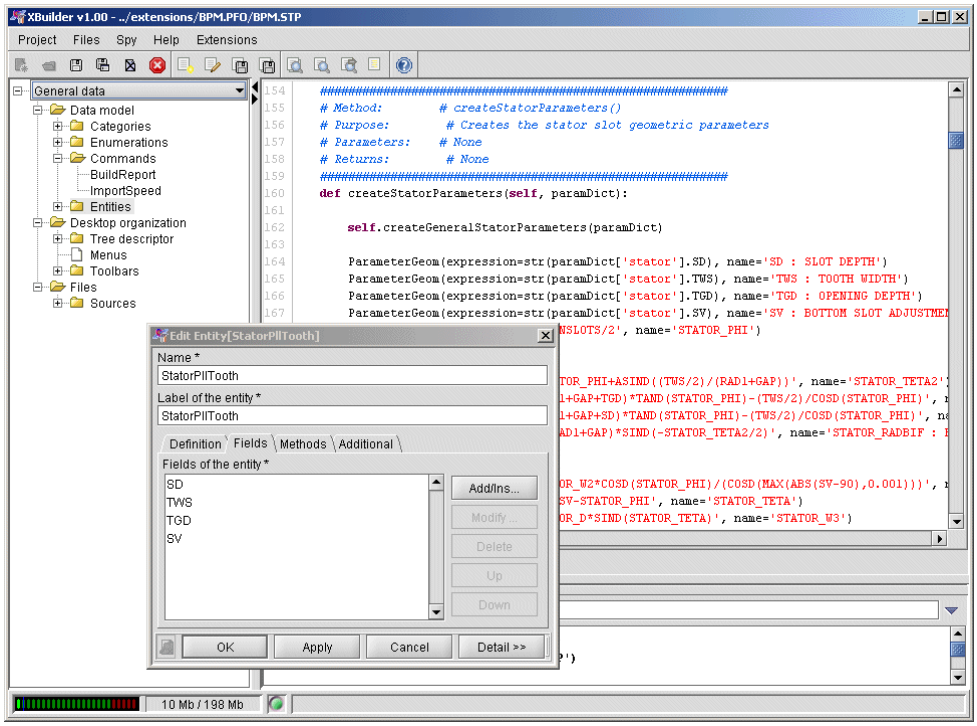


FIG. 5.18 – XBuilder : Définition du modèle de données et des templates python d'un contexte métier

Pour répondre au besoin de personnalisation du logiciel Flux, toujours dans l'optique de simplifier et d'accélérer son utilisation, nous avons développé la notion de contexte métier. Ce développement a consisté à ajouter dans Flux une commande permettant de charger dynamiquement un modèle métier. Cette fonctionnalité ajoute dans Flux de nouveaux objets et construit une nouvelle interface mieux adaptée à un métier précis. De plus nous avons introduit dans la FluxFactory la possibilité d'implémenter ces objets métier dans le langage de commande de Flux (Python) et ainsi de décrire les actions à réaliser dans Flux à la manipulation de ces nouveaux objets. Grâce au logiciel XBuilder le développement d'un contexte métier est relativement simple et devient accessible à un utilisateur avancé. Le développement d'une IHM dédiée utilisant la puissance de calcul de Flux n'est plus réservée aux seuls développeurs de Flux, ce qui devrait faciliter la création d'un grand nombre de personnalisations métier dans l'avenir.

Chapitre 6

Modéliser le métier moteur dans Flux

6.1 Introduction

Après avoir présenté le principe de la FluxFactory et le développement réalisé sur la notion de contextes métier, nous pouvons maintenant envisager la façon dont va se mettre en place le développement du logiciel FluxMotor. La première étape est de personnaliser Flux au métier de la conception et de l'analyse des moteurs. Pour cela nous avons développé des contextes métier pour différents types de moteurs. Certains ont été commercialisés pour leur partie géométrique. Dans ce chapitre je présenterai la mise en place du contexte dédié à l'étude des moteurs brushless à aimants et notamment le modèle de données réalisé à partir de la spécification présentée dans la première partie.

6.2 Le modèle UML des moteurs à aimants

6.2.1 L'entité moteur

Dans un premier temps l'objectif de ce contexte est de permettre à l'utilisateur de créer des géométries de moteurs à aimants dans Flux à partir de paramètres généraux. Pour définir quels sont ces paramètres, nous avons créé dans le modèle une entité *MotorBPM* dont une partie du modèle est représenté Figure 6.1. Cette entité regroupe les paramètres généraux du moteur comme la longueur d'entrefer (GAP) ou la longueur de fer (*stack_length*). L'entité *MotorBPM* contient également des champs liés à une définition éléments finis comme la densité de maillage (*mesh_factor*) ou le paramétrage de la boîte infinie (*InfiniteBox*) comme on a pu le spécifier dans la première partie. Pour décrire les topologies de rotor, de stator, le modèle de bobinage ou l'existence ou non d'excentricités, l'entité moteur contient des champs de type entité abstraite.

Ces entités sont *Rotor*, *Stator*, *Winding* et *Excentricity*. Elles se dérivent en plusieurs sous-types qui définissent les paramètres des différentes topologies.

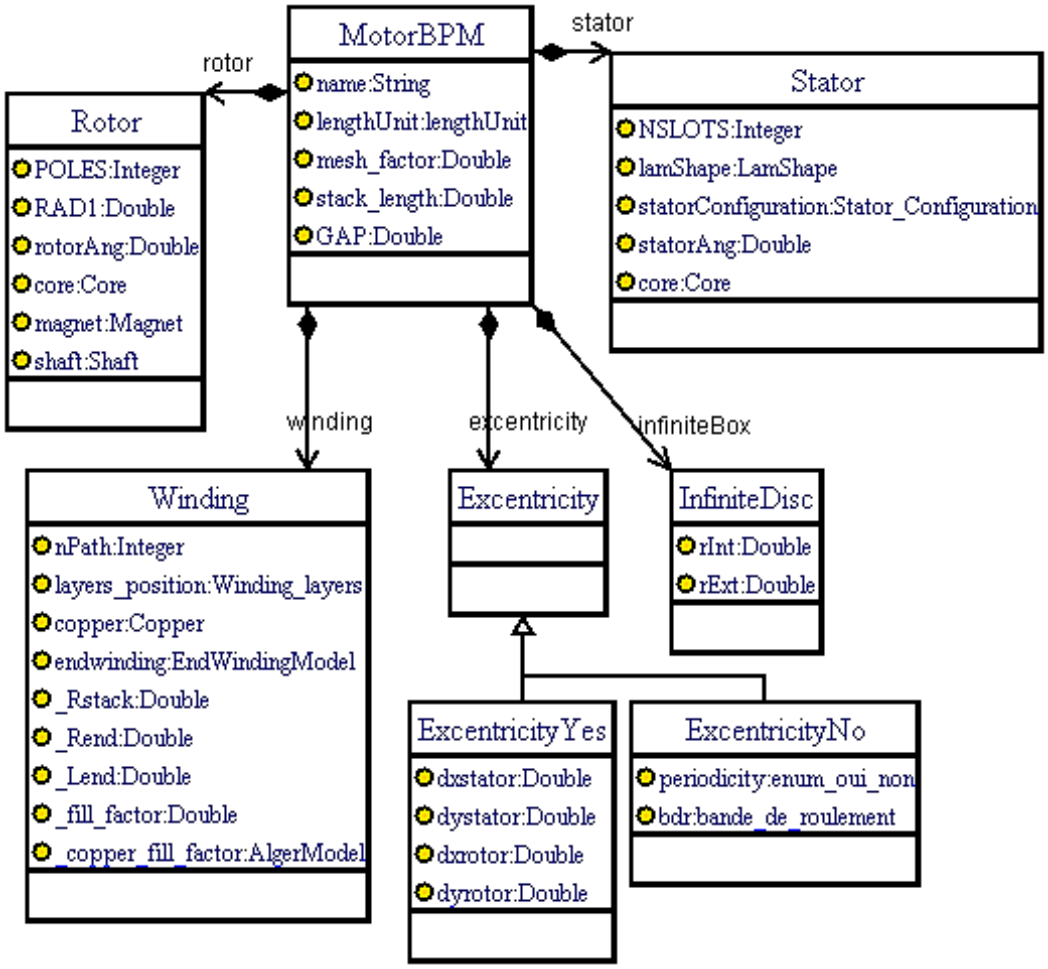


FIG. 6.1 – L'entité moteur et ses attributs dans le modèle de la BPM Overlay

6.2.2 Les topologies rotor

Nous avons vu dans la première partie que les rotors à aimants se déclinent en plusieurs familles de topologies, et en différents enfoncements des aimants au sein de chaque famille. L'entité Rotor se dérive donc en sous-types correspondants aux différentes familles de rotor. Ces sous-types contiennent le champ *magnetType* de type entité abstraite qui se dérive en plusieurs sous-types correspondants aux enfoncements des aimants. Une petite partie de ce modèle est représentée Figure 6.2. On peut également retrouver dans l'entité *Rotor,* les matériaux fer (*Core*), aimant (*Magnet*) et le

matériau de l'arbre (*Shaft*) en association, ce qui signifie que les matériaux seront définis de manière séparée et utilisables par différents moteurs.

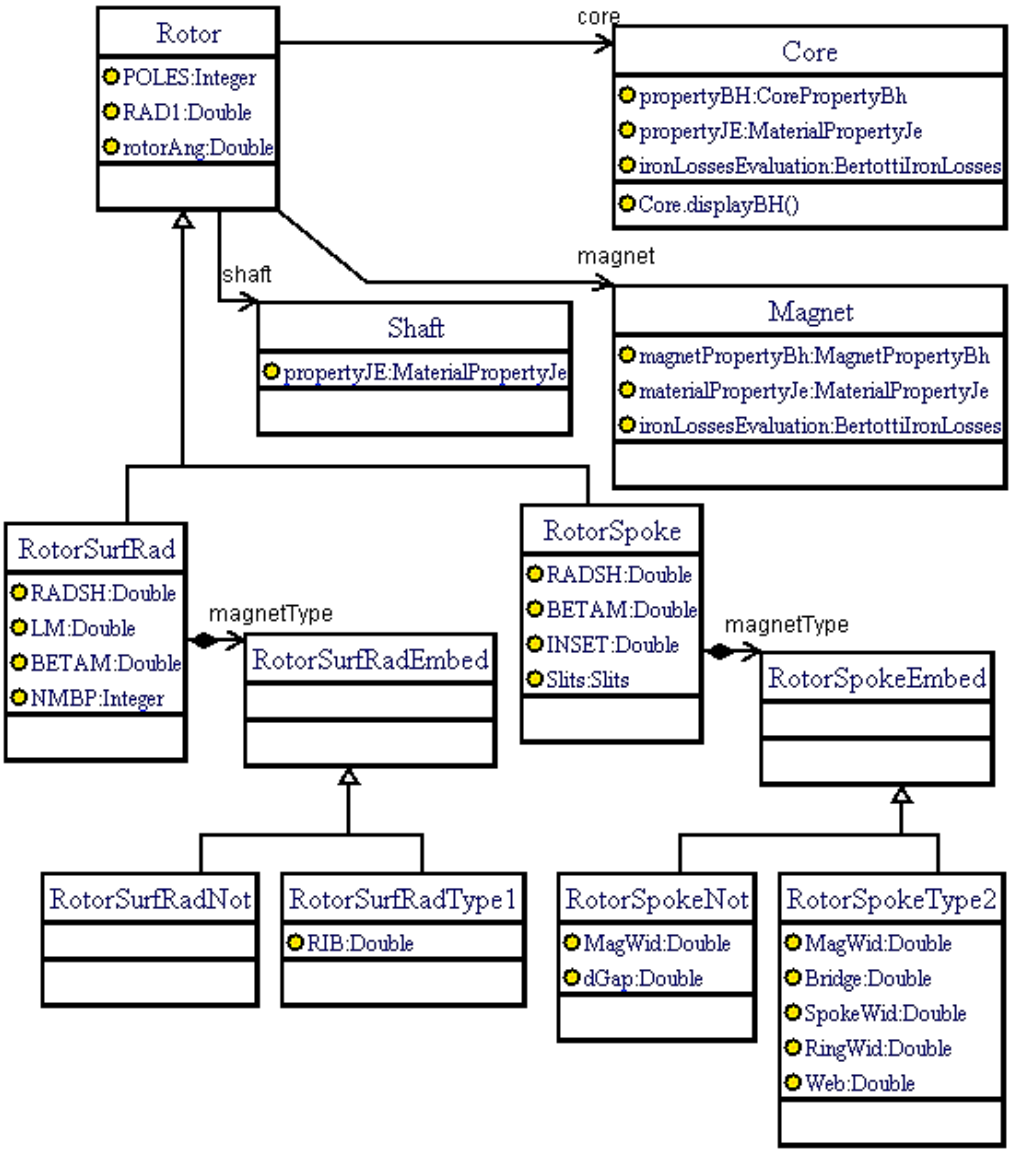


FIG. 6.2 – La classification des topologies rotor dans le modèle UML (4 représentées, 36 en réalité)

6.2.3 Les topologies stator

La modélisation UML d'un stator est représentée Figure 6.3. L'entité *Stator* possède des champs généraux comme le nombre d'encoches (*NSLOTS*) ou la présence ou

non de dents bifurquées (*statorConfiguration*). L'entité abstraite *Stator* se dérive en plusieurs sous-types correspondant aux différentes formes d'encoches (*StatorSquare*, *StatorPllSlot*, *StatorRound*). Chacune des formes d'encoches possède ses propres paramètres (*SD* : slot depth/profondeur d'encoche, *SO* : slot opening/ouverture d'encoche, etc...). L'entité *Stator* comporte également un champ de type *LamShape* ("Lamination shape") représentant la forme de la carcasse et qui se décline en plusieurs formes (*Circle* : circulaire ; *RectRound* : rectangulaire avec extrémités arrondies).

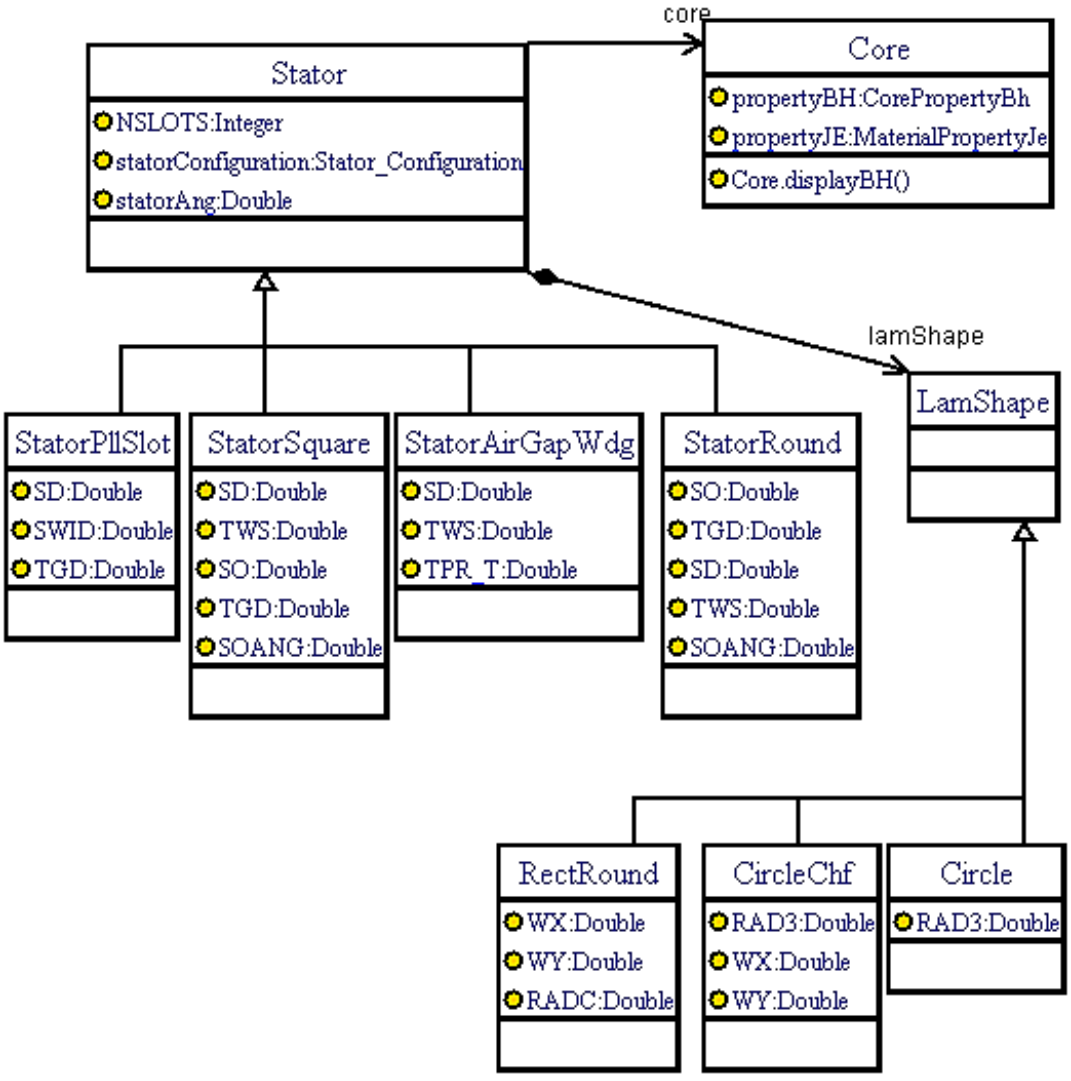


FIG. 6.3 – Diagramme UML de la définition des stators

6.3 Le modèle UML du bobinage

6.3.1 Les types de bobinages

Le modèle UML du bobinage est représenté Figure 6.4. On retrouve dans ce modèle la possibilité de choisir un bobinage personnalisé ou classique avec la présence des entités *CustomWinding* et *ClassicalWinding* qui héritent de l'entité *Winding*. Avec un bobinage personnalisé, on définit chacune des phases (*WindingPhase*) constituées de bobines (*WindingCoil*) définies par une encoche aller, une encoche retour et un nombre de spires. Avec un bobinage classique, on a le choix d'un type de bobinage (*WindingType*) qui peut être un bobinage imbriqué (*LapWinding*), concentrique (*ConcentricWinding*) ou à pas fractionnaire (*FracSlotWinding*). A la construction d'un bobinage classique, on évalue le tableau des phases, c'est pourquoi l'entité *ClassicalWinding* possède un attribut *WindingPhase*, mais cet attribut est calculé par programme et non fourni par l'utilisateur. Les attributs évaluées par programme sont marqués d'un underscore. Ainsi on peut noter les attributs *_Rstack*, *_Rend* et *_Lend* dans l'entité *Winding* qui stockent les résistances de phases et inductances de têtes de bobines qui seront calculées par le logiciel.

6.3.2 Les modèles de têtes de bobines

La définition du modèle utilisé pour le calcul des têtes de bobines est représenté par l'attribut de type *EndWindingModel*. Il se décline en cinq sous-types correspondant aux modèles présentés dans la première partie (Alger, Kostenko, etc...).

6.4 Le modèle UML des matériaux

Les matériaux peuvent être décrits de manière indépendante. Dans le modèle, les entités qui décrivent la définition des matériaux héritent toutes de l'entité MotorMaterial. Cette entité contient un attribut *mass* qui représente la masse volumique. Il permettra notamment de calculer l'inertie du rotor.

6.4.1 Le cuivre

L'entité *Copper* hérite, comme tous les matériaux, de l'entité *MotorMaterial*. Deux entités héritent de l'entité cuivre : *CopperAWG* et *CopperUserValue*. Dans un cas on définit le cuivre en choisissant le numéro d'une bobine de l'American Wire Gage (dont la classification sera donnée dans une image d'aide), dans l'autre on définit une résistivité, un coefficient de température et une section. La section peut être circulaire ou

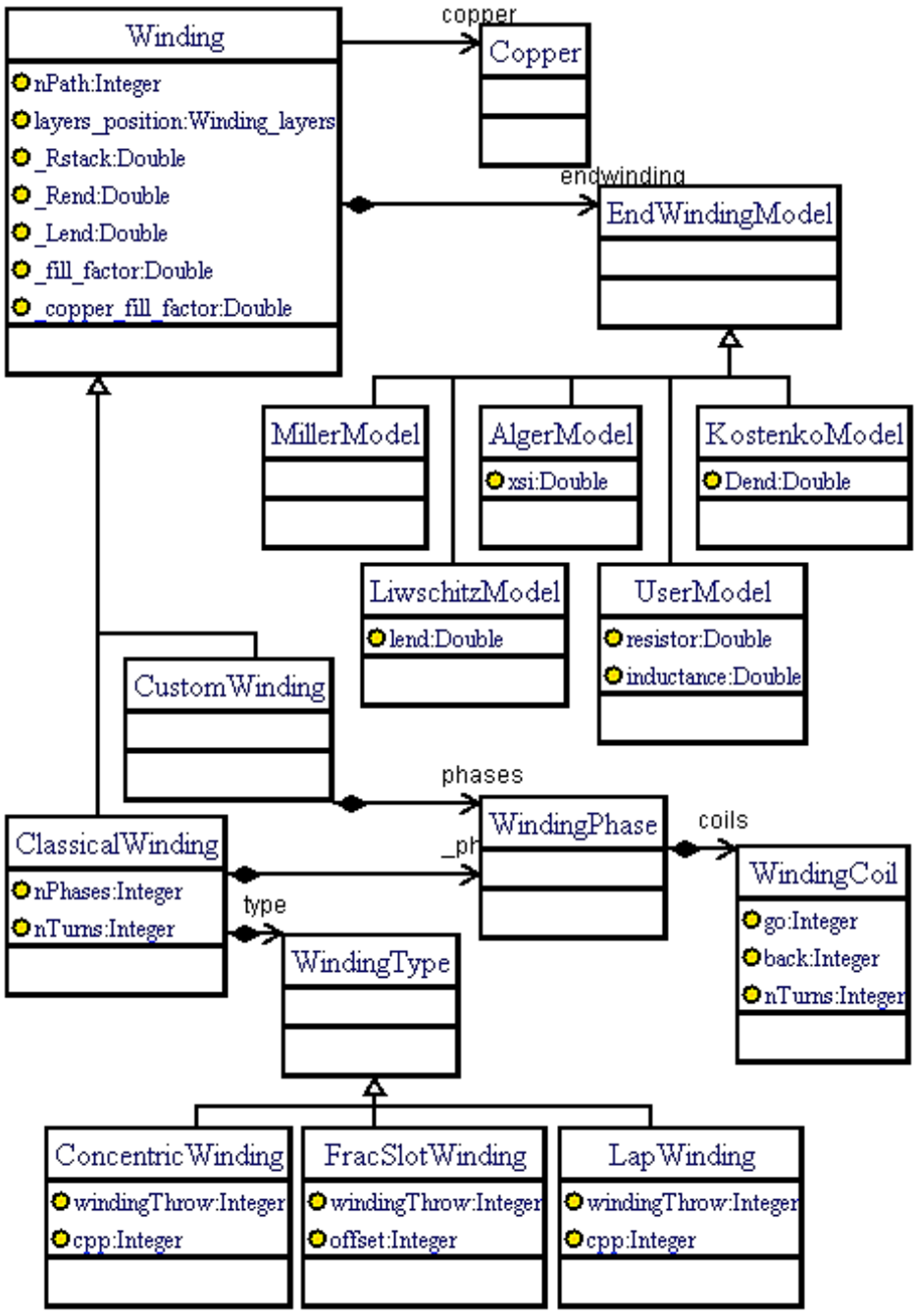


Fig. 6.4 – Modèle UML du bobinage dans la BPM Overlay

rectangulaire (ce qui est modélisé par la présence des entités *CircularCopperSection* et *RectangularCopperSection*) et on définit l'épaisseur de l'isolant (modélisé par le champ *insulator_thickness*). Les coefficients de foisonnement seront évalués à la construction du moteur et stockés dans les champs *_fill_factor* et *_copper_fill_factor* de l'entité *Winding* (Figure 6.4).

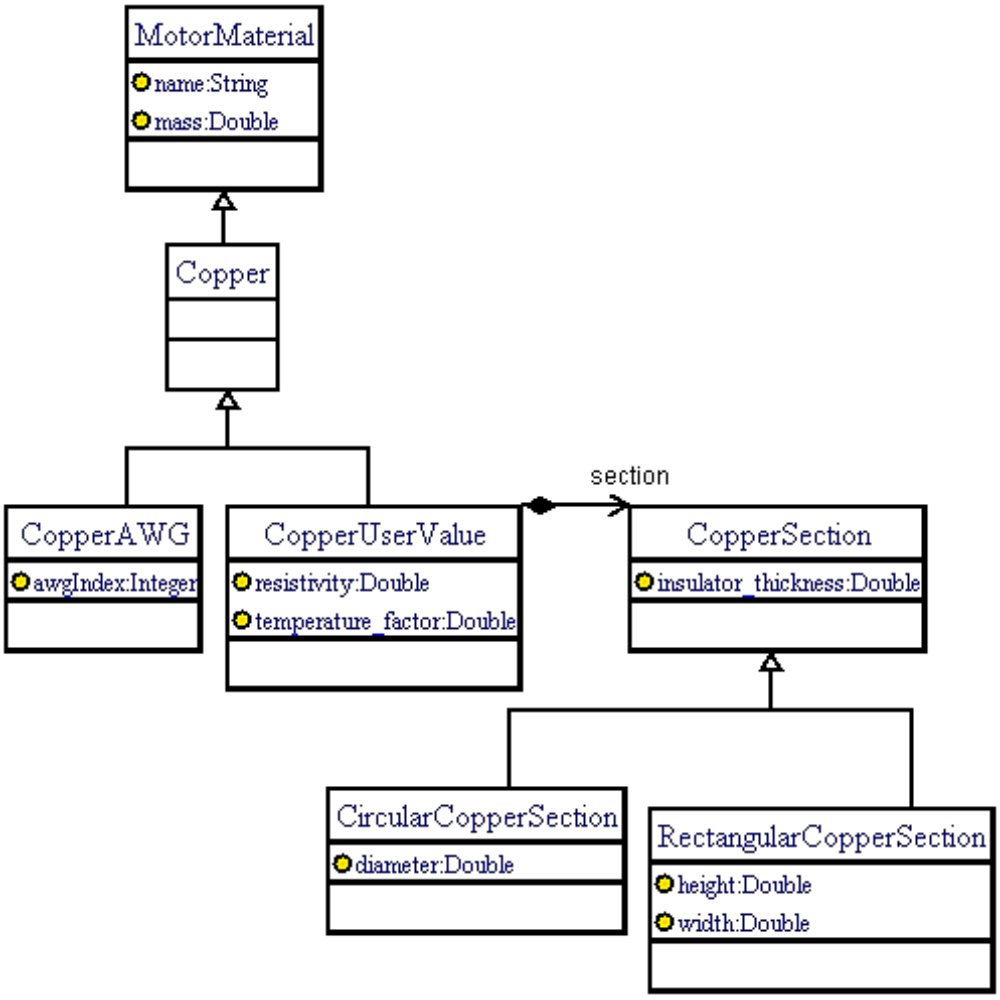


FIG. 6.5 – Modèle UML du matériau cuivre

6.4.2 Les matériaux rotor et stator

Le modèle UML des matériaux utilisés pour les carcasses rotor et stator est représenté par l'entité Core Figure 6.6. L'entité Core a un attribut propriété B(H) (*CorePropertyBh*) et un attribut propriété J(E) (*MaterialPropertyJe*). Plusieurs modèles de

courbes B(H) sont proposés. On définit également les coefficients de Bertotti pour le calcul des pertes fer dans l'entité *BertottiIronLosses*.

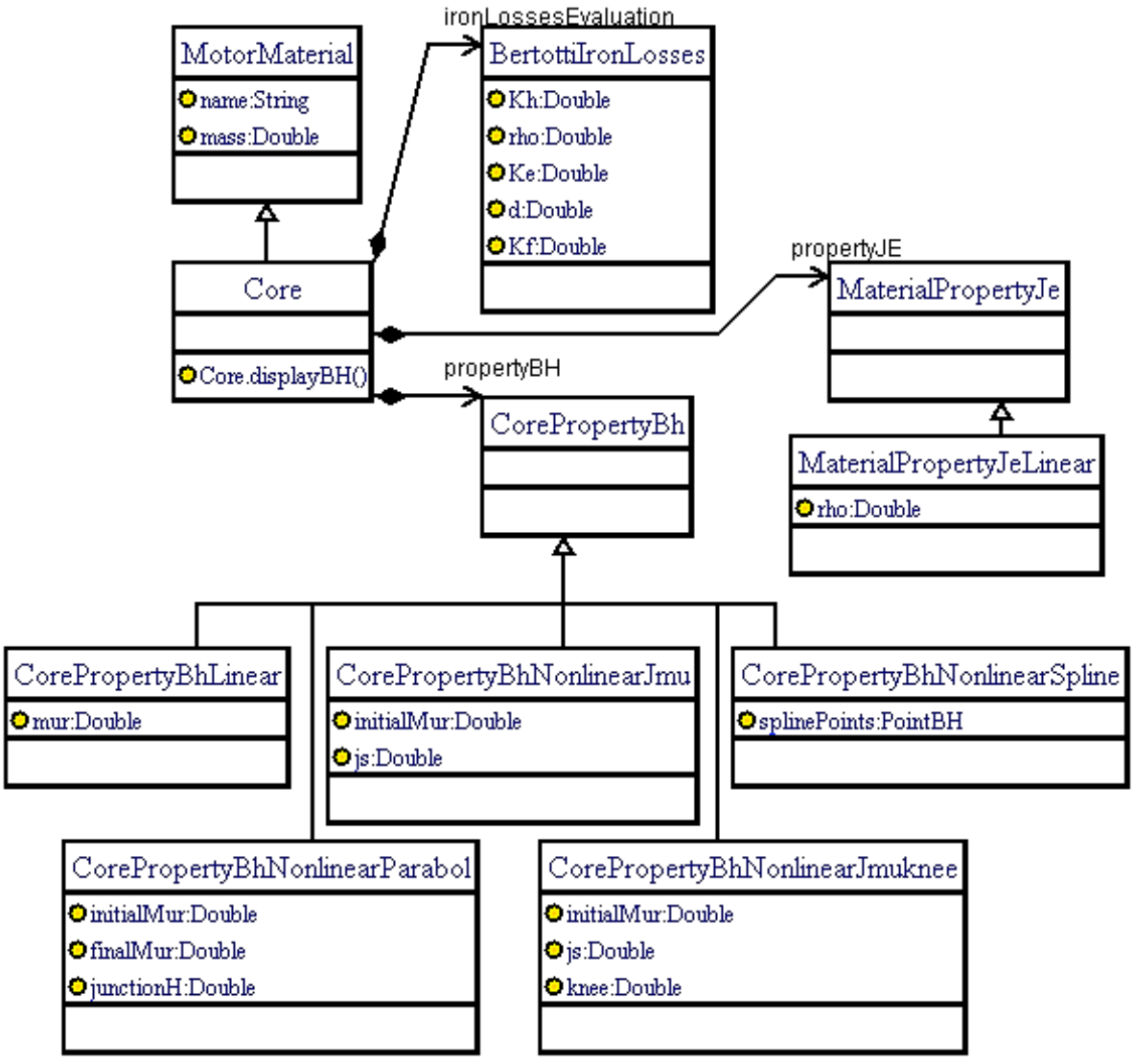


FIG. 6.6 – Modèle UML des matériaux de la carcasse rotor et stator

6.4.3 Les aimants

Le modèle des aimants est très proche de celui des matériaux de la carcasse. Cependant, les modèles de courbes B(H) proposés sont différents.

6.4.4 L'IHM

Une fois le modèle UML décrit, les paramètres d'un moteur et des matériaux sont définis. Pour améliorer l'interface, nous avons ajouté des icônes sur les entités, des pages d'aide sur les attributs et rangés certains attributs dans des onglets. De plus nous avons défini dans XBuilder un modèle d'arbre, dans lequel nous avons placé l'entité MotorBPM et un répertoire Matériaux dans lequel nous avons placé les entités Copper, Core, Magnet, etc...

On peut charger ce modèle d'extension dans Flux. Le logiciel interprête ce modèle et fait apparaître un nouveau contexte dans le quel on retrouve l'arbre décrit. En cliquant dans l'arbre on peut par exemple créer un moteur. La boîte de dialogue de création de moteur apparaît alors. Cette boîte correspond au modèle de données décrit dans XBuilder.

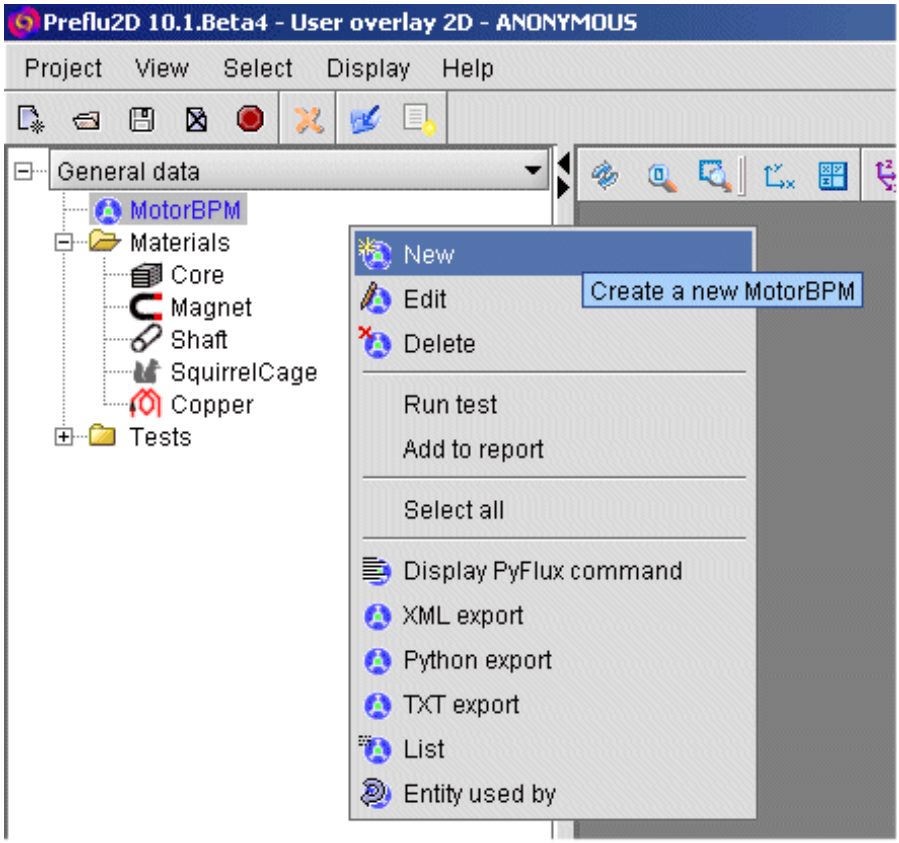


FIG. 6.7 – Arbre de l'extension BPM Overlay

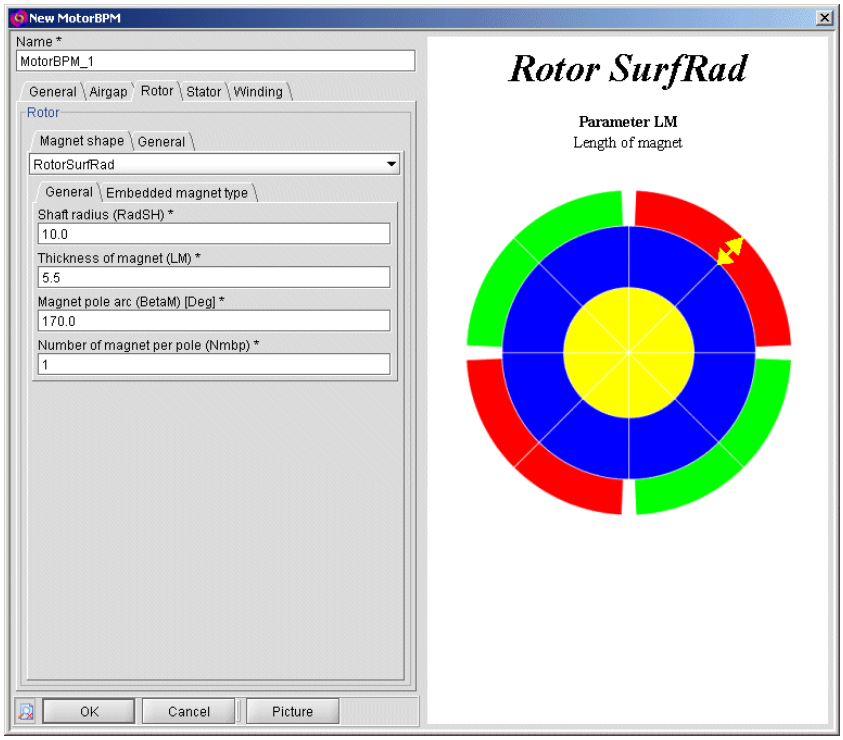
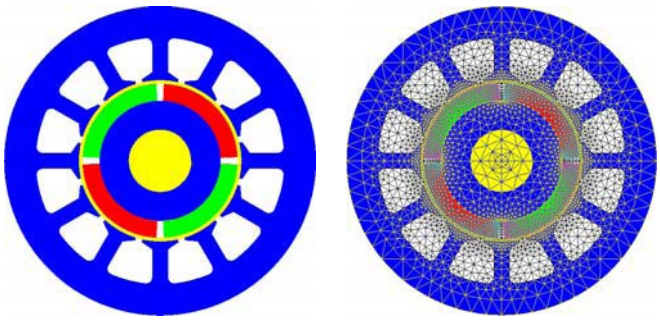


FIG. 6.8 – Boîte de dialogue de définition d'un moteur sans balais à aimants



TAB. 6.1 – Exemple de moteur BPM créé dans le contexte métier

6.4.5 Les templates Python

Cette IHM permet de créer des objets moteur et matériaux, mais n'implique aucune action dans le logiciel Flux. Nous avons donc réalisé une implémentation Python des comportements métier. Pour l'objet MotorBPM, la template décrit les actions permettant de tracer la géométrie, de définir les densités de maillage et d'affecter les régions. Pour les matériaux, les templates créent un matériau Flux correspondant aux paramètres rentrés par l'utilisateur. Certains paramètres sont évalués. Ainsi on calcule dans les templates python les valeurs des composants circuit (résistances des phases, inductances des têtes de bobines).

6.5 Le modèle UML des essais normalisés

Le contexte métier permet donc pour le moment à l'utilisateur de définir son moteur et ses matériaux, ce qui implique la construction de sa géométrie et de son maillage dans Flux. Une fois ce moteur construit nous chercherons à réaliser des études sur ce moteur. Pour faciliter l'utilisation, nous souhaitons proposer des études précâblées qui ont été décrites dans la première partie. Nous avons donc défini, dans le modèle, des entités définissant les paramètres de ces essais. Toutes ces entités héritent de l'entité *Test* qui contient la méthode *run()* permettant d'automatiser l'affectation de la physique et de lancer le calcul. Une partie du modèle des essais qui reprend la spécification de la première partie est représentée Figure 6.9.

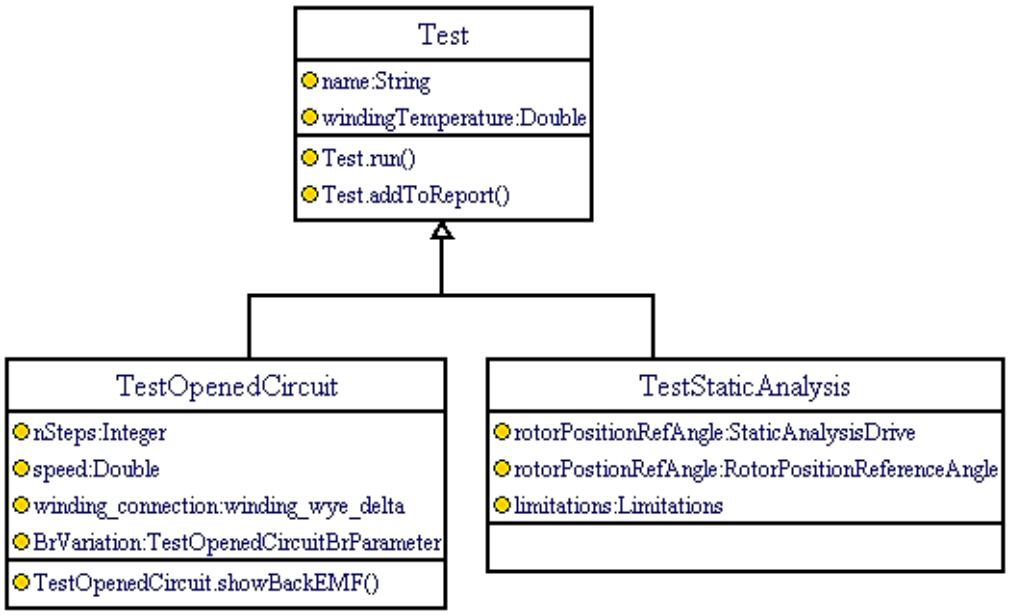


FIG. 6.9 – Modèles des essais

En plus d'automatiser le calcul, nous souhaitons faciliter l'exploitation des résultats. Nous avons donc modélisé les exploitations standards comme la fem induite, le couple de détente ou les schémas équivalents. A la fin d'un calcul ces résultats sont créés automatiquement par les templates python. A partir de l'arbre l'utilisateur peut afficher ou cacher les résultats. Ces résultats font référence au Moteur et à l'essai qui ont permis de le calculer. Toutes les entités résultats héritent de l'entité Result (Figure 6.10). Cette entité contient deux méthodes. La méthode *show()* permet d'afficher un résultat. La méthode *addToReport()* permet d'ajouter la synthèse de ce résultat dans un rapport.

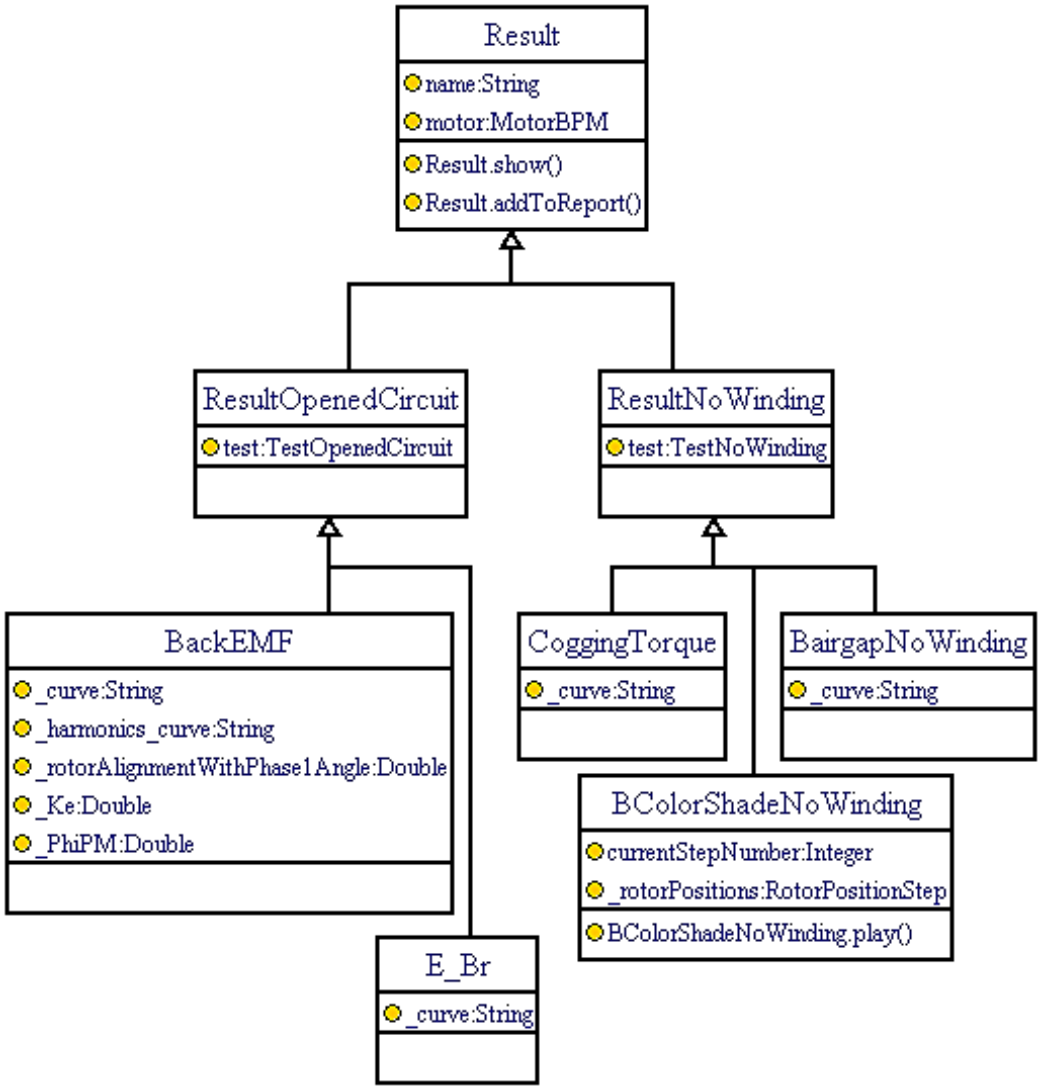


FIG. 6.10 – Modèle des résultats

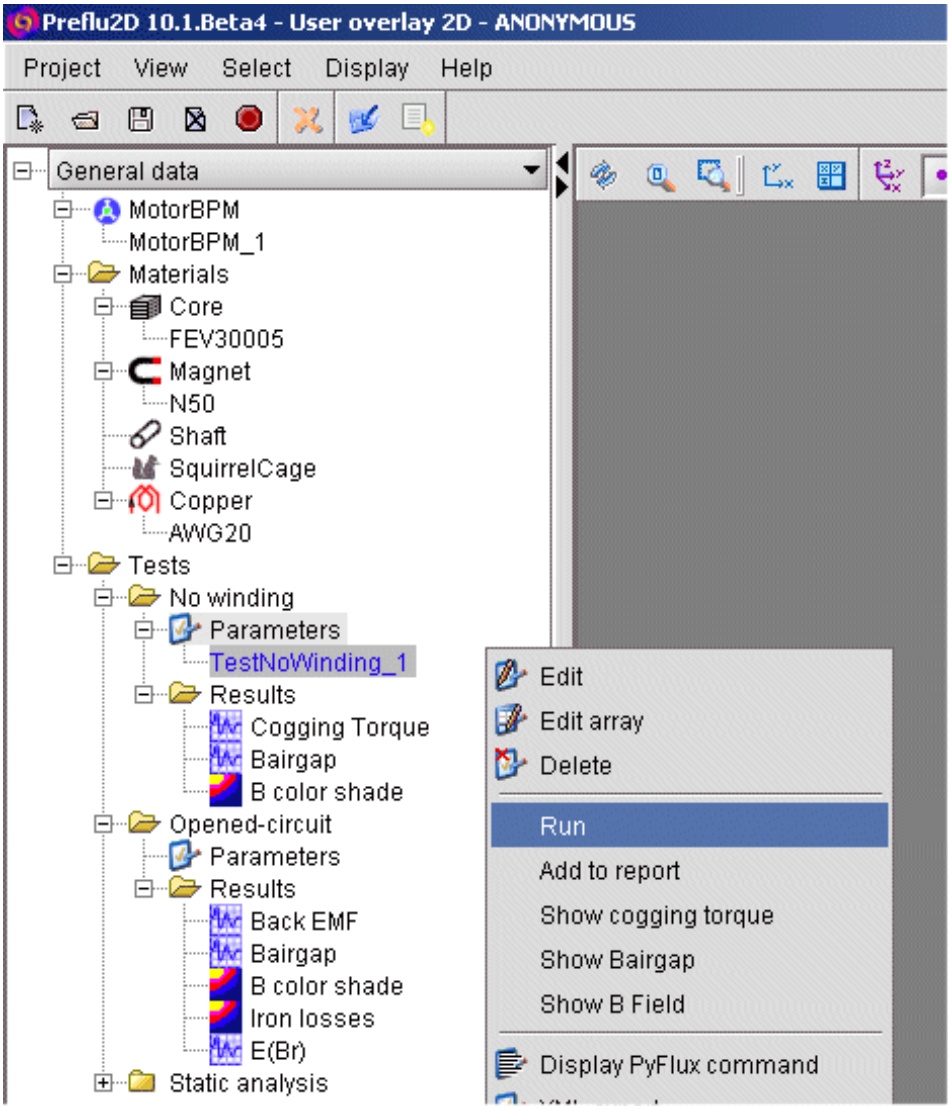


FIG. 6.11 – Lancement du calcul d'un essai sans bobinage - Modélisation des résultats dans l'arbre

6.6 Les autres moteurs

Basées sur les mêmes principes que la bibliothèque de moteurs sans balais à aimants permanents (BPM), d'autres bibliothèques moteur ont été développées pendant la thèse. Il s'agit de bibliothèques géométriques pour les machines asynchrones, les machines à courant continu à aimants, les machines à reluctances variables.

Dans le cadre d'un projet de recherche nommé Arema, dont Valéo est le leader et auquel Cedrat participe, une bibliothèque complète comprenant la définition métier d'un moteur à griffes et ses méthodes d'analyse est développée. Pour ces travaux, la partie CAO est très importante, de même que la qualité du maillage et la possibilité de résoudre de gros problèmes avec prise en compte des courants de Foucault. La réalisation d'une bibliothèque pour les machines BPM 3D à flux axial est envisagée. Les bibliothèques 3D peuvent représenter un réel atout. Les outils analytiques se limitent souvent au 2D et la qualité du logiciel généraliste éléments finis nécessaire pour faire tourner ces simulations de moteurs en 3D est importante. Enfin c'est en 3D que les problèmes sont les plus longs et les plus difficiles à décrire, l'approche métier est donc d'autant plus intéressante.

Bibliothèques moteur	Géométrie	Physique	Essais	Résultats
BPM (Brushless Permanent Magnet)	X	X	à vide, en charge, analyse statique, fréquentiel, court-circuit	couple, tensions, courants, dégradés de B, de H, rendement, pertes joules, pertes fer, inductances, circuit équivalent
BPMOR (Brushless Permanent Magnet Outer Rotor)	X	X	à vide, en charge, analyse statique, fréquentiel, court-circuit	couple, tensions, courants, dégradés de B, de H, rendement, pertes joules, pertes fer, inductances, circuit équivalent
IM (Induction Machine)	X			
IMOR (Induction Machine Outer Rotor)				
DCM (Permanent Magnet DC Machine)	X			
SRM (Switch Reluctance Machine)	X			
WFDC (Woundfield DC machine)				
ClawPole Machine (3D)	en cours	en cours	à vide, à rotor enlevé, en charge	modèle de potier, couple, tensions, courants, dégradés
Axial flux BPM machine (3D)				

FIG. 6.12 – Synthèse des bibliothèques moteur développées

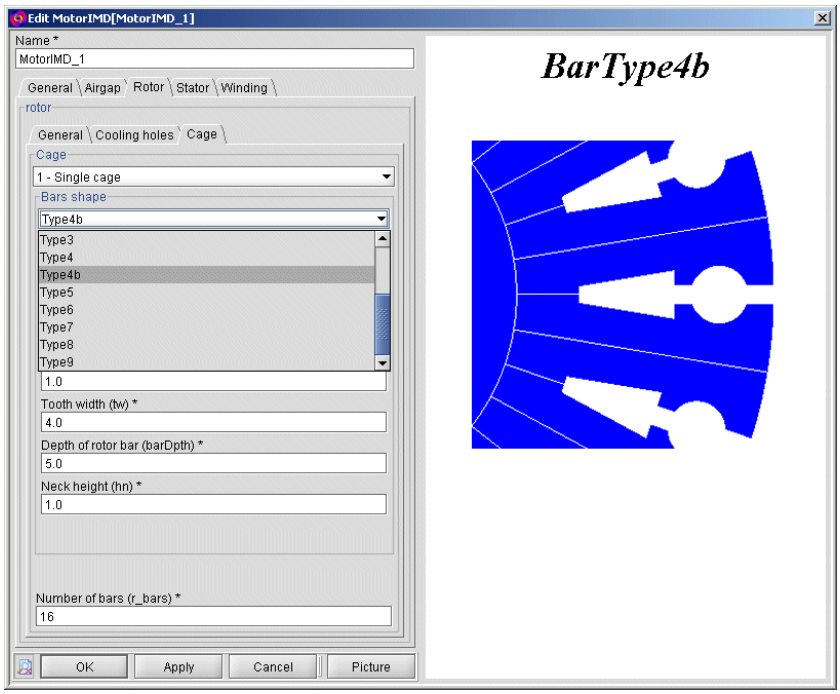


FIG. 6.13 – Définition d'une machine asynchrone

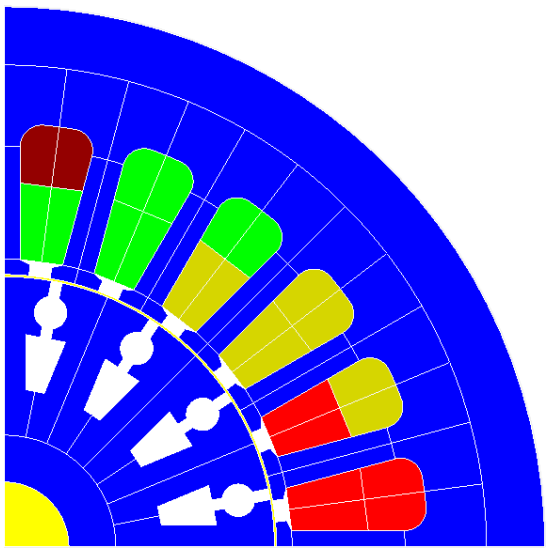


FIG. 6.14 – Exemple de machine asynchrone générée dans Flux

6.7 Conclusion

Dans ce chapitre j'ai présenté la façon dont on a structuré les données du contexte dédié à l'étude des moteurs brushless à aimants. Il permet la définition d'un objet moteur. Une fois le moteur défini, des objets correspondants aux études précâblées peuvent être définies par l'utilisateur. A la fin du calcul d'une étude, les résultats sont directement visualisables via une simple méthode. Une dernière méthode permet l'ajout des données dans un rapport, mais cette fonctionnalité sera expliquée plus en détail dans le prochain chapitre.

Avec la technologie des contextes métier, nous avons donc répondu à notre objectif de personnalisation de Flux au métier moteur. Son utilisation est plus simple et plus rapide. De plus, Flux est maintenant capable de dialoguer facilement avec des logiciels dédiés au métier du moteur, puisqu'il partagent avec eux les mêmes paramètres métier. Une fonction d'import d'un moteur défini dans le logiciel SPEED a donc pu être développée facilement dans Flux. Cette fonction lit les paramètres d'un moteur dans SPEED et crée une ligne python permettant la construction du moteur dans le contexte dédié.

Cependant, cette méthodologie n'est pas suffisante à la réalisation d'un logiciel moteur. En effet le logiciel Flux est mono-projet, ce qui veut dire qu'il ne peut gérer dans un projet qu'une seule géométrie et une seule application physique. Les objets du contexte étant internes au projet Flux, l'utilisation est limitée à la construction d'un seul moteur et au calcul d'une seule étude.

Chapitre 7

L'application moteur

7.1 Introduction

Les contextes métier développés pour les machines tournantes rendent le logiciel Flux plus facile de prise en main. Son utilisation ne nécessite pas de connaissances particulières en éléments finis. L'introduction d'un modèle dédié aux machines électriques rend aisée la communication avec les autres outils de la conception. Grâce à l'import SPEED, on peut rapidement passer d'une étude de prédimensionnement à une étude éléments finis. Cependant, Flux et ses bibliothèques ne permettent l'automatisation que d'un moteur et d'une étude par projet. Or le concepteur cherche à caractériser un moteur complètement et donc réaliser plusieurs études. De plus il cherche à comparer les performances respectives de plusieurs dimensionnements ou types de machines.

C'est pour répondre à toutes ces attentes que nous proposons le développement de FluxMotor. Ce nouveau logiciel est capable de capitaliser les résultats de plusieurs études réalisées sur différents moteurs. Il utilise Flux en tant que serveur de calcul et masque la complexité liée à la gestion des projets Flux. Le développement d'un tel logiciel comporte plusieurs étapes que je présenterai dans ce chapitre :

- Création du modèle de FluxMotor avec FluxBuilder et validation de son interface.
- Mise en place d'un serveur de calcul Flux.
- Ecriture d'un gestionnaire de serveur Flux.
- Récupération des résultats.

7.2 Le modèle de l'application

FluxMotor contrairement aux contextes métier constitue un nouveau logiciel. Il a donc été construit avec FluxBuilder. Son modèle de données est basé sur ceux des contextes dédiés aux moteurs dans Flux. Pour le compléter, nous avons élaboré un modèle graphique proche de celui de Flux. Il permet de représenter la géométrie, le

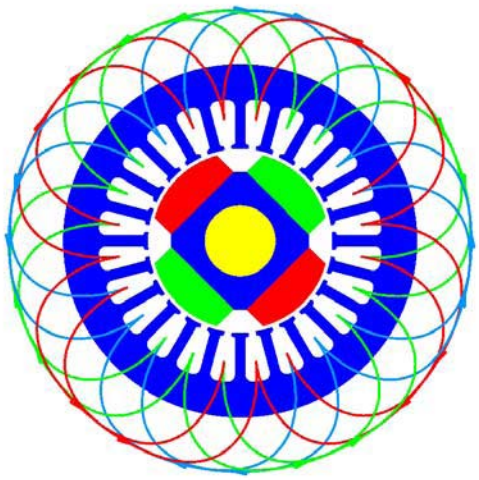


FIG. 7.1 – Représentation du bobinage sur la géométrie du moteur

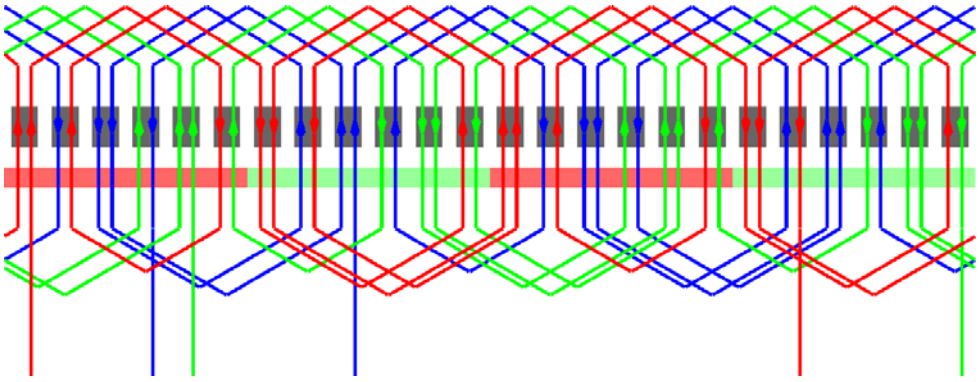


FIG. 7.2 – Représentation panoramique du bobinage

maillage ou les dégradés qui seront réccupérés à partir de Flux. Des représentations graphiques spécifiques au bobinage des moteurs ont également été ajoutées (Figures 7.1, 7.2). Le logiciel FluxMotor bénéficie également de l'afficheur courbe et un afficheur rapport a été introduit.

7.3 L'architecture Client/Serveur

Une fois le modèle de données défini et l'interface graphique validée, nous devons implémenter ces objets en java de manière à leur donner un comportement. Pour les objets moteur, le principe de l'algorithme consiste à piloter Flux de manière à ce qu'il ouvre le contexte dédié correspondant et qu'il crée le moteur. Une fois le moteur créé

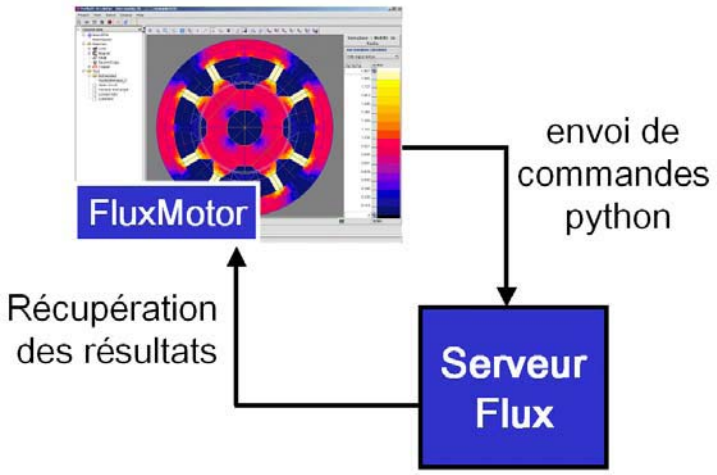


FIG. 7.3 – Principe du pilotage de Flux par FluxMotor

dans Flux, FluxMotor recupère les informations graphiques. Au lancement d'un calcul c'est le même principe : FluxMotor pilote Flux pour lancer le calcul et récupère les résultats. Puisque FluxMotor bénéficie du même modèle que les contextes moteur de Flux, la ligne de commande Python permettant de créer un moteur est la même dans les deux logiciels. Le pilotage de Flux consiste donc simplement à lui envoyer les lignes python de création d'un moteur ou de lancement d'un calcul.

Pour permettre cette implémentation du modèle de FluxMotor nous devons donc développer un serveur Flux répondant à deux services fondamentaux qui sont :
- Interpréter et exécuter une ligne Python
- Retourner les résultats

Maintenant que les services sont définis, nous devons choisir une architecture et une technologie de communication. Pour cela, nous devons rappeler que le développement d'un serveur Flux sera réutilisé par d'autres logiciels métier et par des logiciels d'optimisation comme GOT. En effet ces logiciels peuvent demander de nombreux calculs à Flux et nous devons anticiper le besoin de faire du calcul distribué, c'est à dire répartir les calculs sur un parc d'ordinateurs sur lesquels tournent des serveurs Flux. C'est pourquoi nous avons choisi d'utiliser une architecture Client/Serveur. Dans cette architecture deux programmes tournent en même temps : le programme Client (FluxMotor) qui est demandeur de services, et le programme serveur (Flux) qui réalise des services et retournent le résultat au client. Ces programmes peuvent communiquer entre eux en local (sur une même machine) ou à distance (via un réseau). La technologie utilisée est le RMI (Remote Method Invocation)[35] qui est fournie dans le langage de programmation Java.

Cette architecture doit répondre à plusieures exigences :

- Elle doit être évolutive, c'est à dire qu'elle doit être capable de supporter les évolutions futures du logiciel comme l'introduction de nouveaux services.
- Elle doit également permettre le pilotage des autres logiciels basés sur le principe de la FluxFactory (Flux2D, Flux3D, Inca, GOT, etc...)
- Le serveur Flux ne présente pas d'interface grahique.
- Elle doit nécessiter le moins de modifications possibles sur le code actuel. Elle doit notamment permettre de continuer à développer en parallèle Flux en local et Flux en Client/Serveur. Autrement dit le choix entre un fonctionnement client/serveur ou local doit se faire au lancement du logiciel.

Afin de présenter l'architecture Client/Serveur mise en place, je dois d'abord présenter l'architecture du logiciel Flux. Cette architecture est primordiale puisque c'est elle qui doit évoluer en client/serveur. C'est pour une meilleure maintenance, une meilleure lisibilité du programme et donc une meilleure évolutivité que le logiciel Flux, et plus particulièrement le FluxCore, est architecturé en modules regroupant chacune des fonctionnalités spécifiques (Figure 7.4). Un module est un ensemble de composants partageant le même espace de nommage. Tout module possède une frontière avec le monde extérieur bien définie et limitée.

Les modules du FluxCore existant aujourd'hui sont :
- **Le Frontal (le bureau) :** Ce module est responsable de l'affichage graphique.
- **Le GUI Generator (GUI = Graphic User Interface) :** Ce module s'occupe de la construction de fenêtres graphiques (affichées ensuite par le frontal)
- **Le TUI Interpreter (TUI = Textual User Interface) :** Ce module interprète les saisies réalisées par l'utilisateur dans les boîtes et les lignes de commandes python
- **Le Kernel (le noyau) :** Ce module supervise l'application et gère la base de données

D'autres modules sont quant à eux plus spécifiques à l'application Flux (ils se trouvent dans la surcharge) :
- **Le Modeler 3D :** Il gère les visualisations 3 dimensions de Flux
- **La Physique :** Il contient les équations de la physique
- **Le Solver :** Il réalise les calculs pour la résolution

On constate également, à la Figure 7.4, l'utilisation de deux bases de données pour l'application : la « méta base » qui contient le modèle du logiciel, et la base de données « projet », correspondant aux données créées par l'utilisateur.

Les modules communiquent entre eux via un bus. Chacun d'entre eux est demandeur (client) de services et fournisseur (serveur) de services. On peut distinguer cependant la notion de services et de notifications. Dans le cas de la notification, un module envoie une information à un ou des modules sans attendre de valeur de retour. Au démarrage de Flux, les modules se lancent puis se connectent au bus qui fait lui même les liaisons entre les demandeurs et les fournisseurs d'un même service ou d'une même notification.

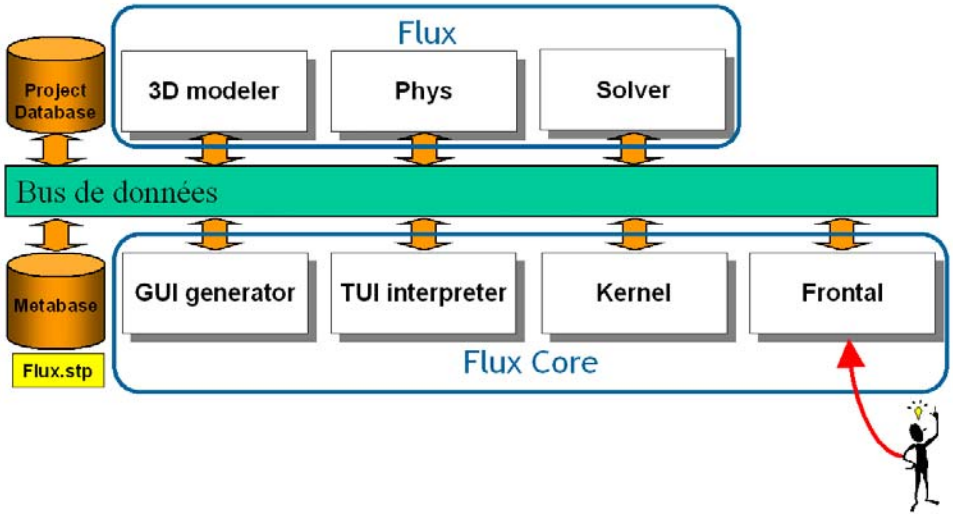


FIG. 7.4 – Architecture modulaire du logiciel Flux

A partir de cette architecture et pour réaliser le logiciel FluxServeur, nous devons adapter le programme de lancement de Flux au fonctionnement Serveur. D'abord, le module frontal qui constitue l'affichage graphique n'est d'aucune utilité. Nous ne lancerons donc pas ce module au démarrage de Flux en mode Serveur. Cela n'affectera pas le fonctionnement des autres modules, car le module frontal n'est pas fournisseur de services (il est seulement notifié ou demandeur de services). De plus, nous devons ajouter un module capable de répondre aux deux services fondamentaux du serveur Flux que nous avons identifiés. Nous avons donc développé un nouveau module appelé ProxyServeur. Ce module donne accès aux deux services du serveur sur un port de la machine. Au démarrage, ce module se connecte également sur le bus de manière à rediriger vers le TUI Interpreter et le Kernel chacun des services qu'il offre. Grâce à ces développements le logiciel Flux peut être lancé en tant que serveur et propose les deux services spécifiés. Pour faciliter le pilotage de Flux, nous avons également développé un code Java qui sert de poignée pour le logiciel Client (Handle) et qui masque la complexité du code d'appel à distance des services proposés par le serveur Flux (Figure 7.5). Cette architecture est évolutive puisque nous pouvons facilement ajouter de nouveaux services dans le module ProxyServeur et dans la poignée Client. De plus elle est applicable à toutes les logiciels construits à partir de la FluxFactory.

7.4 La gestion transparente des fichiers

Une fois le serveur Flux en place, nous pouvons commencer l'implémentation du modèle du logiciel FluxMotor. Cette implémentation consiste à envoyer au serveur Flux les lignes de commande Python de création de moteur et du lancement d'un calcul. Cependant, comme FluxMotor permet de construire plusieurs moteur et de réaliser

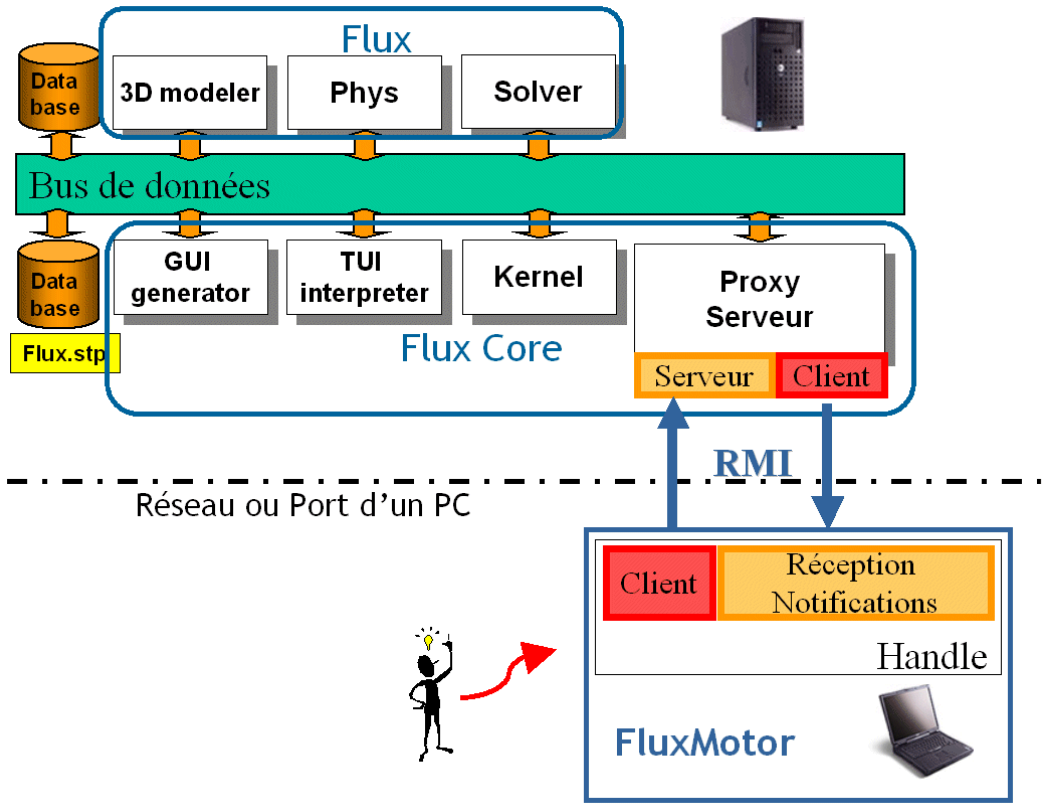


FIG. 7.5 – Pilotage du logiciel Flux par FluxMotor

plusieurs études sur chacun d'eux, nous devons implémenter une gestion des projets Flux. Sous le projet FluxMotor, on sauvegarde un fichier par géométrie de moteur et un fichier par étude. Ainsi à l'édition d'un moteur ou à l'affichage des résultats d'un essai, on pourra réouvrir sur le serveur le projet Flux correspondant. Grâce à cette mécanique, la gestion des fichiers Flux est complètement masquée à l'utilisateur et il peut conserver sous un même projet FluxMotor l'ensemble des études qu'il a réalisé pendant la conception.

7.5 Les rapports

Puisque l'application FluxMotor permet de conserver les différentes études réalisées pendant la conception, nous avons souhaité travailler sur la présentation des données et notamment sur la notion de rapport automatique. Pour que ces rapports puissent être utilisables et modifiables par l'utilisateur, nous devons générer des rapports dans des formats standards comme word, open office, pdf ou html. Cette notion de rapport doit être réutilisable pour toute personnalisation métier de Flux. Ainsi l'approche métier permet de réaliser rapidement une étude et de générer un rapport.

Dans FluxMotor, la notion de rapport se présente sous la forme d'une méthode "ajouter au rapport" applicable à chacun des objets créés par l'utilisateur. Pour implémenter cette méthode, une première solution consiste à coder, pour chacun des objets, l'écriture directe d'un fichier au format html ou word xml. Cependant cette méthode est difficilement modifiable et n'est pas accessible à l'utilisateur. De plus ce code ne profite pas à l'ensemble des logiciels de la FluxFactory.

C'est pourquoi nous avons choisi d'utiliser une technologie appelée transformation xml [37]. Cette technique est divisée en deux partie. Dans un premier temps, une méthode d'export permet d'écrire sous forme xml [36] n'importe quel objet créé par l'utilisateur. Ce fichier xml contient toutes les données sources nécessaires au rapport et les balises correspondent au modèle de l'objet. A cet export xml on peut associer une feuille de style au format xsl. Cette feuille de style définit le format de sortie et la façon dont se présente les données dans le rapport. Ainsi un module de transformation xml permet de générer le rapport. Les formats de sortie proposés par la transformation xml sont le format txt, html et xml document. Le format xml document [38] est un standard xml permettant de décrire assez simplement un document. Il est beaucoup plus simple par exemple que les formats xml word ou open office. De plus, des modules open source permettent à partir d'un fichier xml document de générer du word, du latex, de l'open office ou du pdf.

Cet export xml a été développé dans la FluxCore. Il est donc présent dans FluxMotor mais aussi dans Flux et ses contextes métiers. Avec cette technologie, introduire la notion de rapport automatique dans un logiciel de la FluxFactory ou d'un contexte métier consiste à écrire une feuille de style pour la mise en forme des données. Le code est donc extérieur au code du logiciel. De plus cette feuille de style peut être remplacée si un utilisateur souhaite avoir sa propre feuille de style. Ainsi il peut mettre ses rapports à la norme des documents de sa société par exemple.

7.6 Exemple d'utilisation de FluxMotor

Après avoir présenté les différents développements qui ont été entrepris pour FluxMotor, je propose maintenant de les illustrer rapidement au travers d'un cas d'application :

La société Cedrat Technologie participe à un projet européen appelé MuFly. Elle aide à travers des mesures et des études à déterminer le moteur qui entrainera l'hélice d'un petit hélicoptère avec caméra embarquée. Ces travaux pourront d'ailleurs donné lieu à des études de conception pour améliorer les performances des moteurs existant. Ce projet a donné lieu à des simulations éléments finis dont une partie constituera ici notre cas d'application. L'objectif est de comparer rapidement les performances de deux moteurs brushless à rotor extérieur. Ces deux moteurs ont le même stator mais la forme des aimants au rotor diffère (Figures 7.7 et 7.8). Pour discuter de l'influence des aimants, un essai à vide sera d'abord simulé pour chacun des deux moteurs avant de passer à l'essai en charge.

7.6.1 La géométrie

Grâce à une définition métier des moteurs, basée sur le choix des topologies et le réglage des paramètes associés, les géométries sont créées en quelques secondes dans FluxMotor. La définition métier du bobinage propose des algorithmes qui calculent la répartition des bobines dans les encoches. Dans le cas présent, les moteurs ont 12 pôles et 9 encoches. On utilise donc l'algorithme de bobinage à pas fractionnaire. Deux paramètres suffisent alors à définir le bobinage :

- le pas du bobinage qui est égal à 1 (bobinage par dent)
- le décalage entre les phases qui ici est égal à 2 encoches (pour une bonne répartition spatiale des phases on utilise souvent la formule : $offset = \frac{2}{3} * \frac{Nslots}{Npoles} + k * \frac{2*Nslots}{Npoles}, \ \ k = 1, 2, 3, ...$)

Cette définition métier permet de créer deux projets Flux qui contiennent déjà : la géométrie du moteur, le maillage et la répartition des régions. Par rapport à une utilisation classique de Flux, plusieurs tâches ont été automatisées, ce qui représente un gain de temps non négligeable pour l'utilisateur (Figure 7.6).

Construction de la géométrie d'un moteur			
Flux		**FluxMotor**	
Calcul des coordonnées	1/2 heure	Choix de la forme des aimants	10s
Définition des paramètres	10 min	Choix de la forme d'encoche	10s
Saisie des coordonnées des points	10 min	Saisie des côtes du moteur	30s
Construction des lignes Construction des faces	10 min	Définition métier du bobinage	10s
Regroupement des faces par région (rotor, stator, distribution du bobinage)	5 min		
Affectation des densité de maillage sur la géométrie	15 min		
Création des transformation Propagation de la demi encoche et du demi-aimant Création des périodicités	10 min		
TOTAL	**1 h 30**	**TOTAL**	**1 min**

FIG. 7.6 – Construction de la géométrie d'un moteur dans Flux et dans FluxMotor

7.6.2 Les matériaux

Une fois les géométries définies, l'utilisateur définit les matériaux des carcasses rotor et stator (Figure 7.9), des aimants et du fil de cuivre utilisé pour le bobinage (le nombre de spires par bobines est renseigné dans la définition métier de la répartition du bobinage).

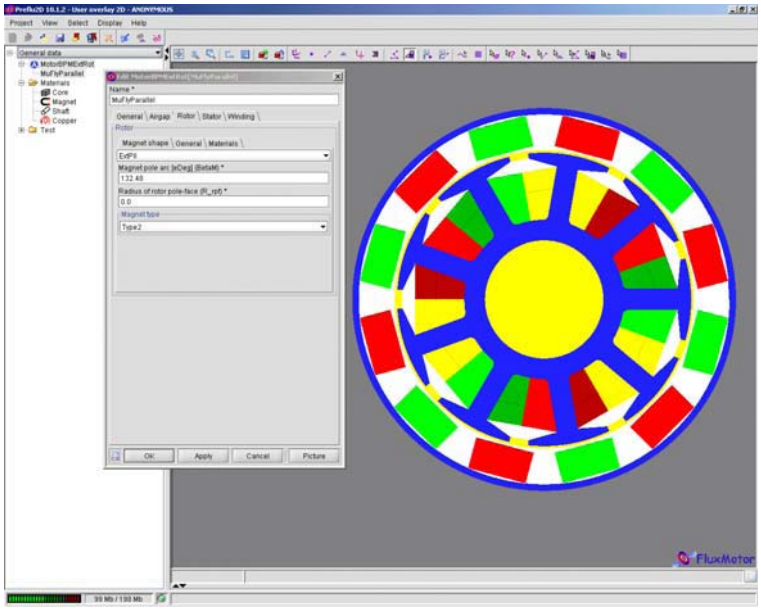


FIG. 7.7 – Définition métier du moteur brushless à rotor extérieur à aimants droits

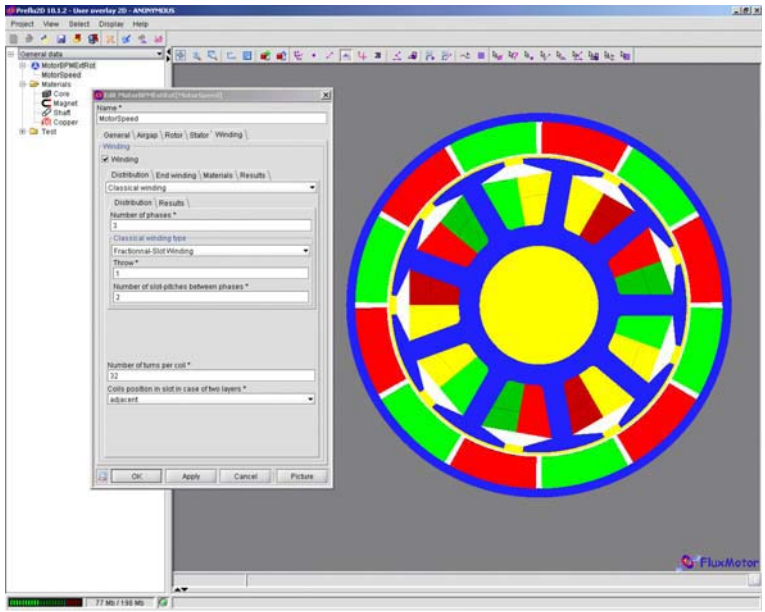


FIG. 7.8 – Définition métier du moteur brushless à rotor extérieur à aimantation radiale

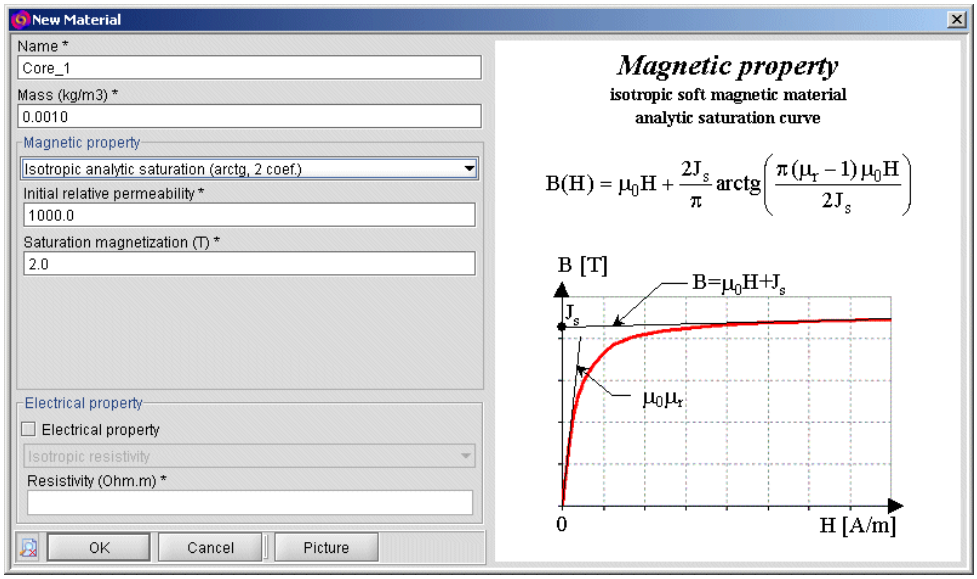


FIG. 7.9 – Définition d'un matériau magnétique pour les carcasses rotor et/ou stator dans FluxMotor

7.6.3 Les études

Dans FluxMotor, les essais que l'on peut réaliser sur un moteur sont des objets manipulables par l'utilisateur et donc rapidement définis (Figures 7.10 et 7.11)

L'essai à vide

Après avoir défini les matériaux, trois paramètres suffisent à définir l'essai à vide. Ces paramètres sont :

- La vitesse
- Le nombre de pas de temps calculés sur la période électrique
- Le type de connexion entre les phases (étoile ou triangle)

Une fois l'essai créé, on peut l'appliquer à un moteur, ce qui lance automatiquement le calcul sur le serveur Flux. A la fin du calcul, les résultats attendus comme la fem sont directement accessibles dans l'arbre. Le même essai est ensuite appliqué à l'autre moteur. FluxMotor permettant de conserver les résultats sous un même projet, on peut comparer les fem des deux moteurs (Figure 7.12).

On constate que la fem du moteur à aimants courbes (en noir à la figure 7.12) a une plus grande amplitude, ce à quoi on pouvait s'attendre puisque le volume des aimants est plus important. Elle est de forme plus trapézoïdale que la fem à aimants droits. Le moteur à aimants courbes devrait donc avoir un meilleur couple moyen lors de l'essai

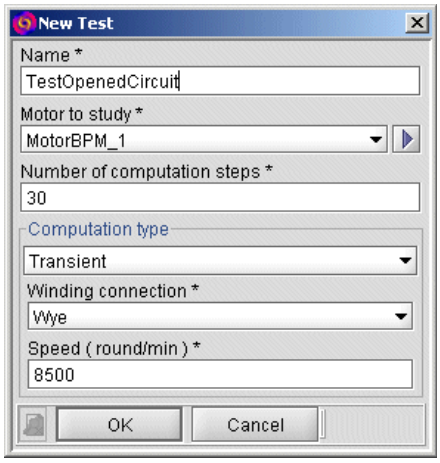


FIG. 7.10 – Définition d'un essai à vide dans FluxMotor

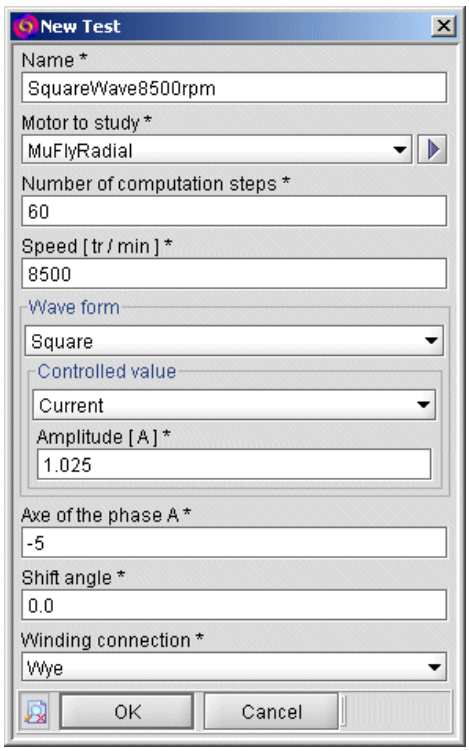


FIG. 7.11 – Definition de l'essai en charge dans FluxMotor

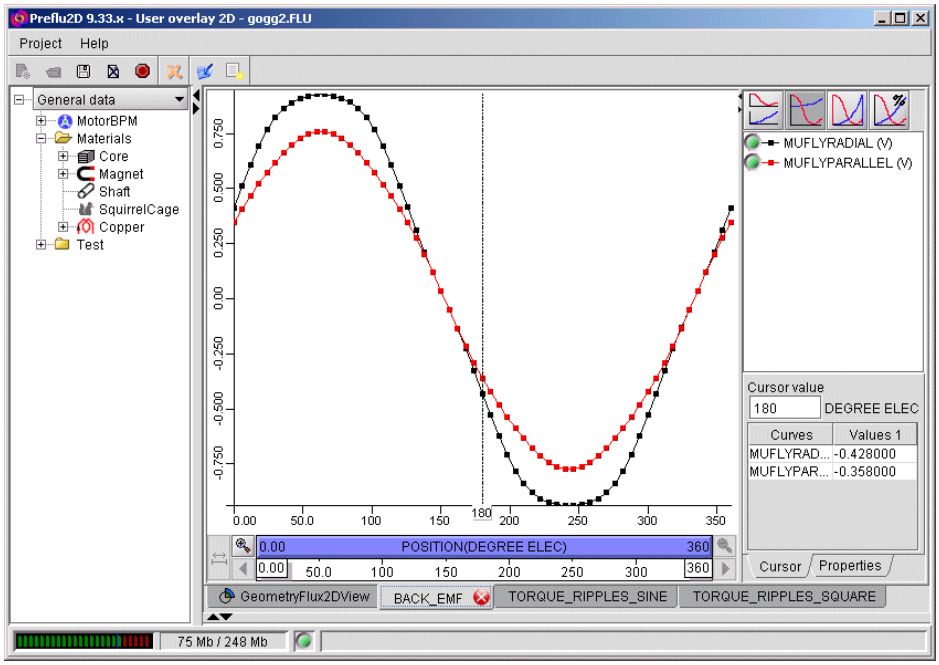


FIG. 7.12 – Comparaison des fem du moteur à aimants droits et du moteur à aimants courbes.

en charge. Le moteur à aimants droits ayant une fem pratiquement sinusoïdale devrait être mieux adapté à un fonctionnement en régime synchrone.

L'essai en charge

Passons maintenant à l'essai en charge. Deux essais seront réalisés à vitesse nominale :

- L'un avec une alimentation des phases en courants en créneaux (fonctionnement BDC)
- L'autre avec une alimentation des phases en courants sinusoïdaux (fonctionnement PMSM)

La même amplitude des courants sera utilisée dans les deux essais.

Les paramètres nécessaires à la définition d'un essai en charge (Fig 7.11) sont :

- Le nombre de calculs réalisés sur la période électrique
- La vitesse
- La forme des courants et leur amplitude
- Le type de branchement entre phases (étoile ou triangle)

Comme pour l'essai à vide, on compare les résultats obtenus par les deux moteurs. Les résultats obtenus confirment le fait que le moteur à aimants courbes a un couple

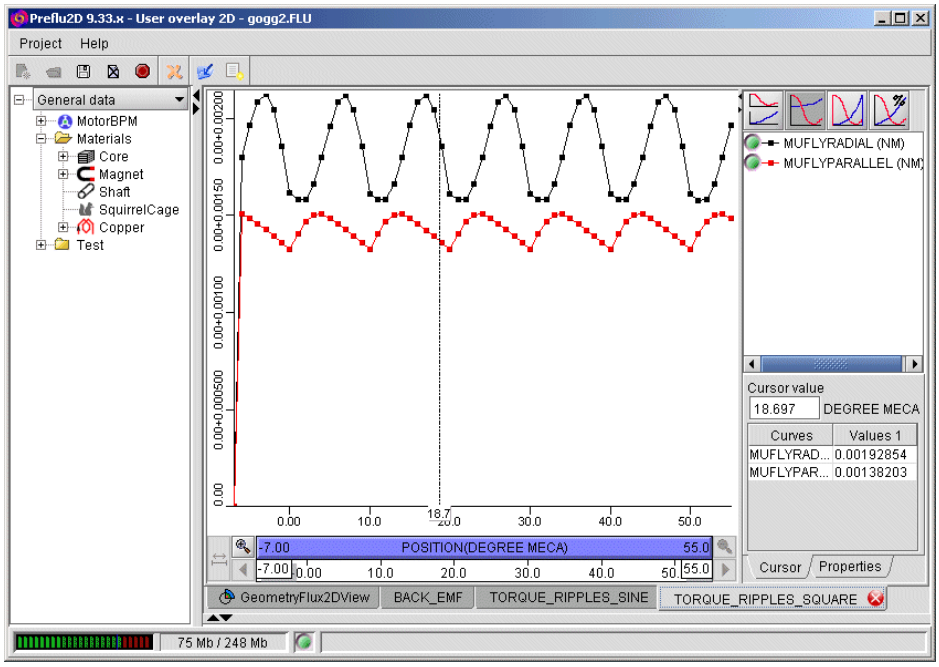


FIG. 7.13 – Ondulations de couple obtenues par les moteurs lors d'un essai en charge avec alimentation en créneaux

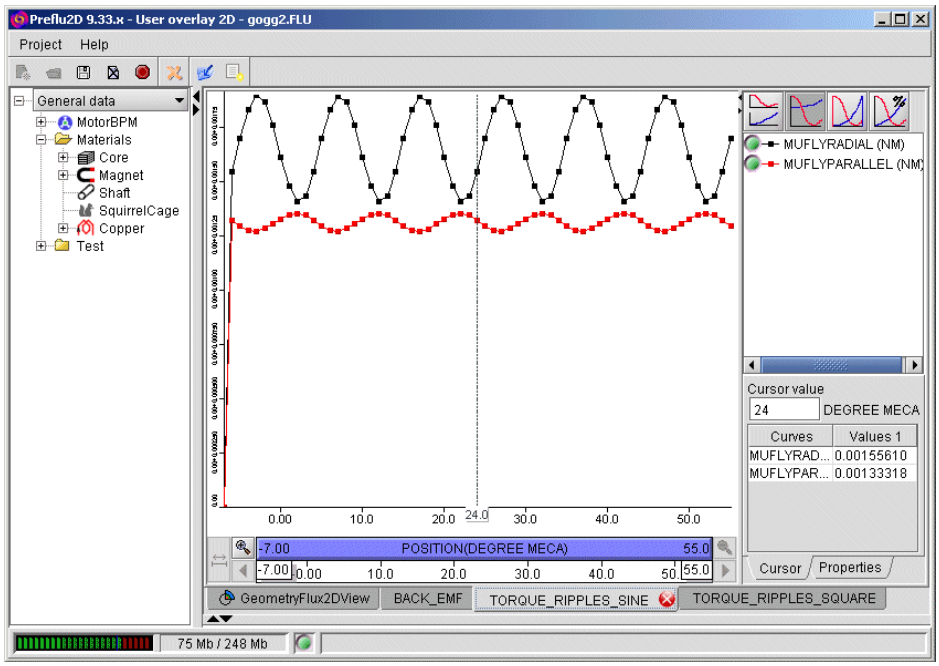


FIG. 7.14 – Ondulations de couple obtenues par les moteurs lors d'un essai en charge avec alimentation en courants sinusoïdaux

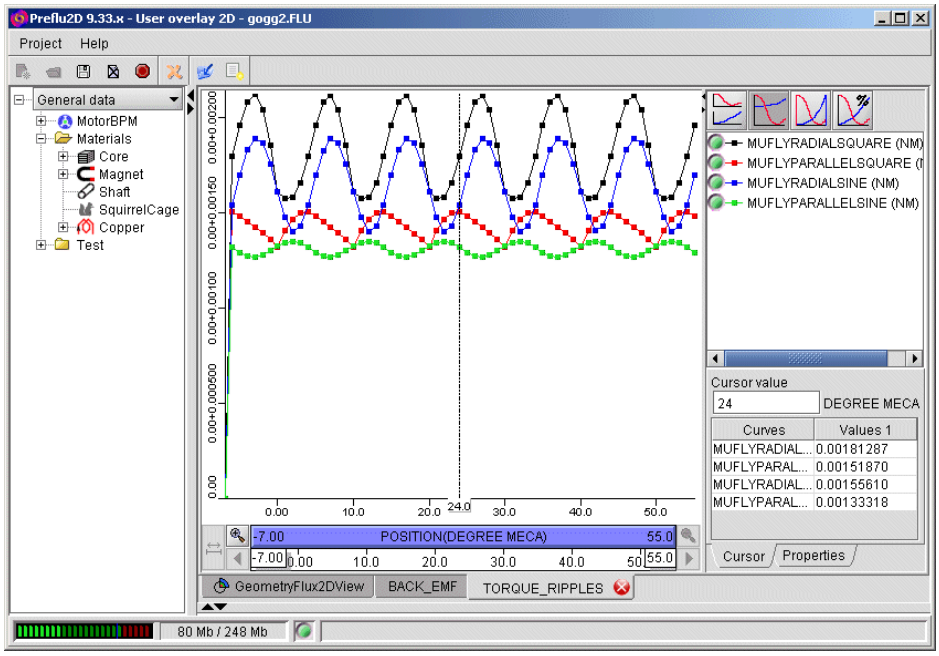


FIG. 7.15 – Comparaison des ondulations de couple obtenues pendant les deux essais

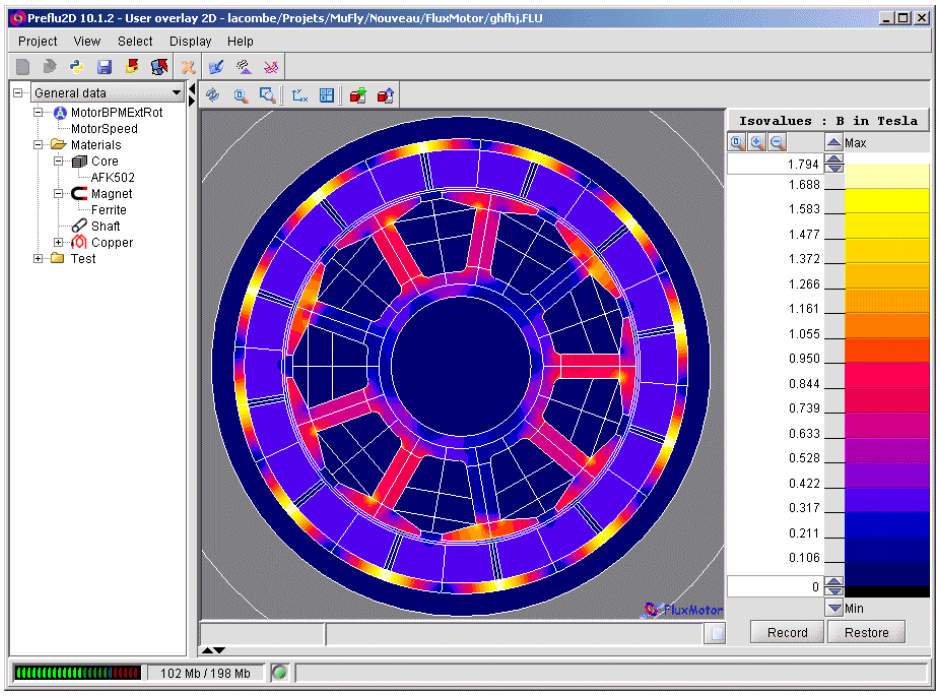


FIG. 7.16 – Visualisation des dégradés de B lors de l'essai en charge

moyen en charge plus important que le moteur à aimants droits. Cependant il entraîne des ondulations de couple plus fortes. Avec une alimentation sinusoïdale, les couples moyens sont légèrement plus faibles, mais les ondulations de couple diminuent encore pour le moteur à aimants droits.

A travers ces quelques calculs, on constate que l'approche métier facilite l'utilisation et réduit le temps de réalisation d'une étude (Figure 7.17). La définition du calcul de l'essai en charge d'un moteur en éléments finis qui prenait presque deux heures dans une utilisation classique de Flux (en comptant la définition de la géométrie) prend maintenant moins de 5 minutes. De plus, conserver les résultats de plusieures machines lors des essais permet de faciliter la comparaison des performances et aide à capitaliser la connaissance métier.

Définition d'un essai			
Flux		**FluxMotor**	
Création d'une application transitoire		Définition des matériaux	**30 s**
Matériaux	**3 min**	Création d'un essai en charge	**30 s**
Définition des matériaux			
Affectation des matériaux aux régions			
Orientation des aimants			
Circuit	**10 min**		
Création d'un circuit			
Calcul des résistances de phase			
Calcul des inductances des têtes de bobines			
Formules pour définir la forme et l'amplitude des courants			
Définition des liens entre composant circuit et région élement finis (bobinage)			
Scénario	**2 min**		
Définition des ensembles mécaniques pour le mouvement			
Définition d'un scénario de résolution avec vitesse imposée sur une période électrique			
TOTAL	**0 h 20**	**TOTAL**	**1 min**

FIG. 7.17 – Définition d'un essai en charge dans Flux et FluxMotor

Pendant une phase de conception, ce gain de temps peut être utilisé par le concepteur pour explorer plus de solutions. Le paramétrage métier permet de changer rapidement des paramètres discrets comme la topologies du rotor ou le nombre d'encoches du stator.

On peut penser que cette approche facilitera également la définition métier d'un problème d'optimisation. Sur ce sujet une thèse va commencer cette année. Le travail devrait consister, dans un premier temps, à développer de nouveaux algorithmes dans le logiciel d'optimisation GOT. Ces algorithmes devraient permettre de construire des surfaces de réponse plus facilement exploitables en utilisant la connaissance de modèles analytiques simples des dispositifs étudiés. Dans un deuxième temps, ces outils d'optimisation seront intégrés dans des logiciels métier tels que FluxMotor.

7.7 Conclusion

Dans ce chapitre, les travaux réalisés sur le superviseur FluxMotor ont été présentés. Ce logiciel pilote Flux et capitalise l'ensemble des études réalisées pendant une conception sous un seul projet. Le logiciel devrait proposer par la suite d'autres types de machines. Cette architecture permet également d'envisager le pilotage d'autres logiciels et des bibliothèques de machines 3D (machines à griffes, moteur à Flux axial, calcul 3D des inductances des têtes de bobines) pourront être introduites. Cela nous permettra aussi de proposer des méthodes d'optimisation. Nous envisageons d'enrichir le modèle du logiciel FluxMotor de manière à ce qu'il puisse proposer la définition d'un problème d'optimisation de manière métier, c'est à dire basée sur la modélisation des paramètres métier du moteur et de ses résultats. L'optimisation sera alors réalisée par pilotage du logiciel d'optimisation GOT (logiciel de la FluxFactory). Ce travail fera l'objet d'une nouvelle thèse.

Enfin ces travaux ont eu un intérêt pour Flux, puisqu'une architecture Client/Serveur a été développée et permet à un logiciel extérieur de piloter le logiciel Flux à distance ou en local via une API Java simple.

Chapitre 8

Autres exemples de bibliothèques métier

8.1 Introduction

Les technologies mises en place avec les contextes métiers et le pilotage de Flux en Client/Serveur sont réutilisables pour d'autres métiers et facilitent la démarche de couplage avec un autre logiciel métier. Dans la première étape, Flux s'approprie le langage du logiciel métier. Ensuite le couplage devient plus facile, puisqu'il constitue de la simple lecture/écriture de données de haut niveau. La démarche de personnalisation du logiciel Flux dans le cadre du projet FluxMotor a donc pu être réutilisée pour d'autre projets.

8.2 Les actionneurs linéaires

Dans le cadre d'un stage de master réalisé au G2Elab et en partenariat avec Schneider, le prototype d'un contexte dédié à l'étude des actionneurs linéaires a été développé (Figure 8.1). Cette bibliothèque permet la definition d'actionneurs simples comme des contacteurs en U, E, etc... des noyaux plongeurs, des déclencheurs. Des études permettent d'automatiser le calcul des courbes de force en fonction du courant et de la tension pour un export sous forme de tables vers les logiciels de la simulation système. Une étude permet d'enchainer plusieurs calculs transitoires pour déterminer l'echelon de tension minimal qui permet la fermeture du contacteur, ceci par un processus de dichotomie. Cette démarche pourrait être relativement longue avec une manipulation à la main de Flux. A ces actionneurs simples, Schneider envisage d'ajouter leurs propres produits. Les performances et caractéristiques de leurs produits pourront ainsi être rapidement retrouvées. De même leur comportements dans un nouvel environnement pourra être étudié plus facilement grâce à l'export vers la simulation système.

Du point de vue de Schneider, une telle application représente de nombreux avantages :

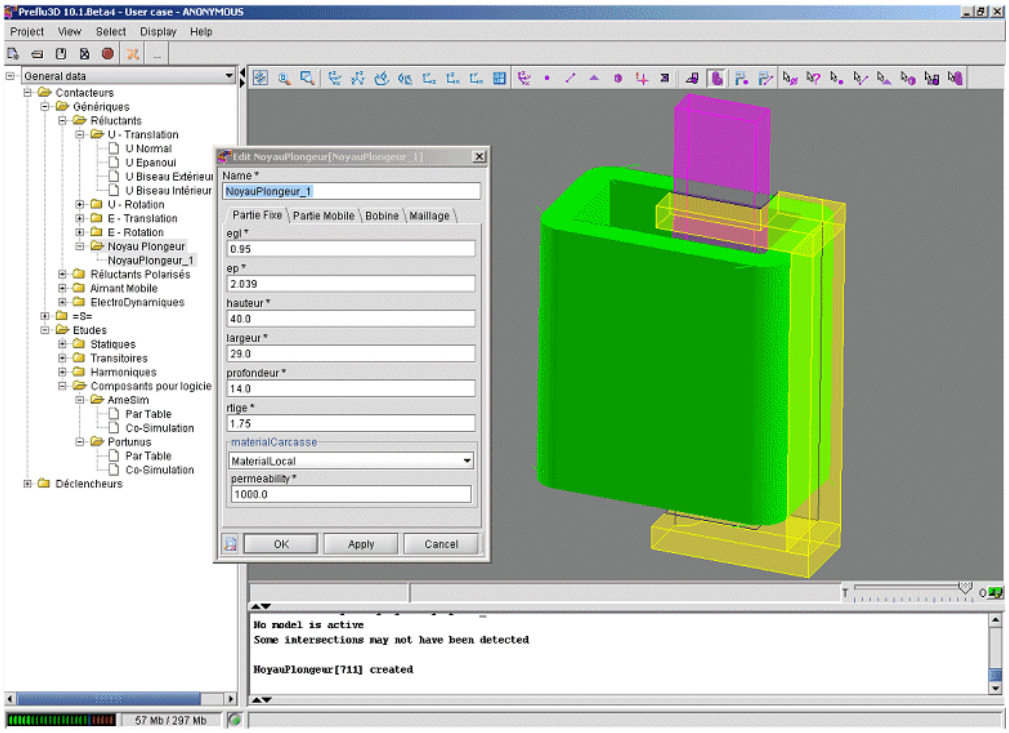


FIG. 8.1 – Contexte dédié à l'étude des actionneurs linéaires

- L'utilisation de Flux est plus simple et plus efficace (Les difficultés d'utilisation sont cachées derrière l'interface métier pour des utilisateurs non spécialistes)
- Elle accélère la recherche de solutions techniques (La définition d'un dispositifs et d'une étude est automatisée)
- Elle aide à la modélisation pour l'approche système (Des exports vers les logiciels de la simulation système sont prédéfinis)
- Elle capitalise de la connaissance (sur les dispositifs et sur les méthodes d'analyse par éléments finis)

La spécification a été réalisée à Schneider. Le stagiaire a ensuite pu se former à XBuilder et à Flux au G2ELab et a développé un prototype. Cela illustre bien l'intérêt de disposer de XBuilder pour réaliser rapidement des contextes mais aussi pour les faire évoluer en interne avec ses propres produits (qui sont soumis à une certaine confidentialité).

8.3 Le contrôle non destructif

CIVA [44] est un logiciel métier pour le contrôle non destructif. Il propose un module ultra sons, et un module courants de Foucault par des méthodes de calcul semi analytiques. CIVA est développé par le CEA et commercialisé par le groupe Cedrat.

Dans le cadre du projet Mutsic, Cedrat a réalisé un prototype permettant le calcul par éléments finis des études du module courants de Foucault. Pour faciliter le couplage avec Flux, un contexte métier qui contient le même modèle de définition des données que CIVA a été développé. Le modèle métier contient la definition d'une pièce, d'un défaut, des capteurs constituant la sonde, puis du déplacement et de l'alimentation de la sonde. Ensuite pour réaliser le couplage avec CIVA, Flux est utilisé comme serveur de calcul et l'architecture client/serveur présentée dans le chapitre précédent est réutilisée.

Grâce aux contextes métiers ce projet a été une réussite et a donné lieu à un nouveau projet plus ambitieux appelé PLAYA. Cet exemple illustre les possibilités qu'offrent l'approche métier pour se coupler avec d'autres logiciels.

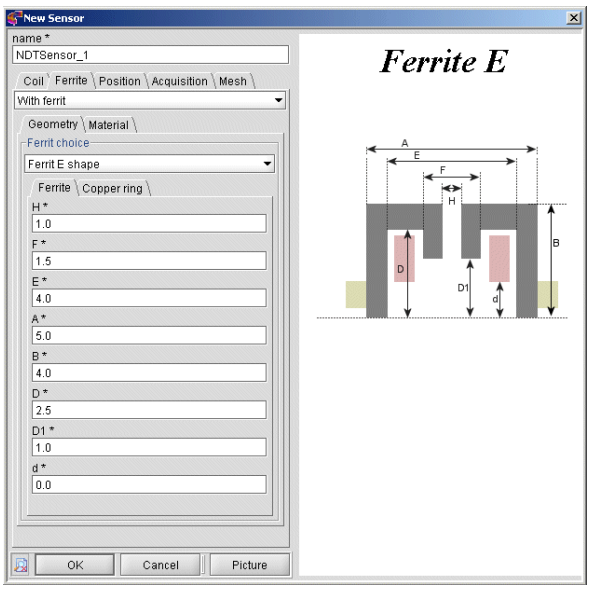


FIG. 8.2 – Définition d'un capteur (Correspond au modèle UML de la figure 8.4)

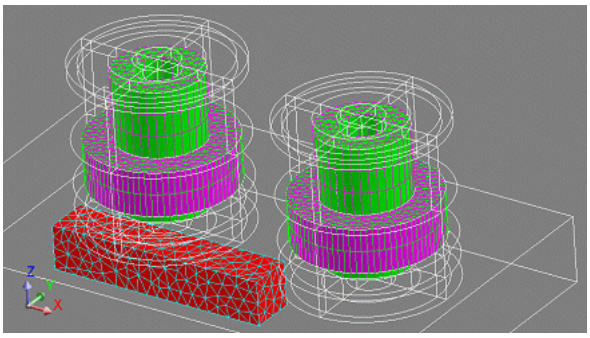


FIG. 8.3 – Une sonde avec une bobine emmettrice et une bobine réceptrice sur le défaut d'une plaque

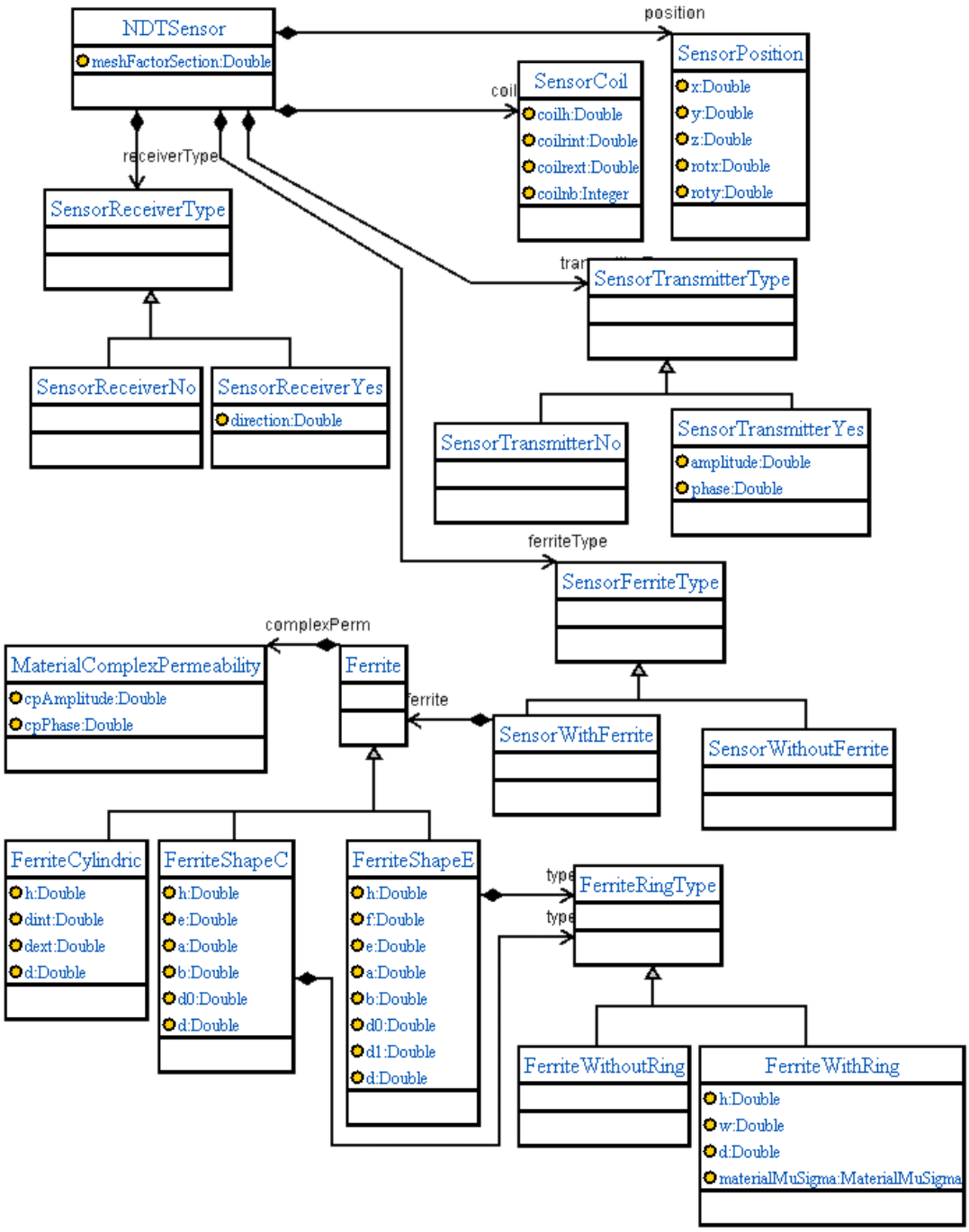


FIG. 8.4 – Partie du modèle du contexte dédié au contrôle non destructif (Modèles d'un capteur)

8.4 Conclusion

D'autres exemples d'application métier ont été présentées dans ce chapitre. Leur développement a été facilité par l'introduction dans Flux de la notion de contexte métier. Avec le logiciel XBuilder, les utilisateurs peuvent développer leurs propres contextes métier ou simplement compléter des contextes existant.

Les contextes métier sont particulièrement bien adaptés pour des travaux ou des calculs répétitifs et permettent de diminuer le temps de définition d'un problème. De plus Flux devient plus accessible et peut être utilisé par des personnes peu ou pas familières avec les éléments finis.

Cette technologie a également facilité le couplage de Flux avec d'autres logiciels. D'abord Flux peut obtenir le même modèle de données et donc les mêmes paramètres par le développement d'un contexte métier. Ensuite, avec le développement du FluxServeur, Flux peut être piloté en client/serveur et recevoir les lignes python contenant la valeur des paramètres métier. C'est ce qui a été fait avec les logiciels CIVA et FluxMotor.

Conclusion

Les logiciels de calcul par éléments finis se sont imposés pour la conception des machines électriques. Ils permettent une évaluation plus précise des performances de la machine et de son rendement. Cependant leur interface généraliste demande souvent du temps pour la définition du problème et de la formation, ce qui limite encore leur utilisation. Le travail effectué a visé à mieux intégrer le logiciel Flux dans le processus de conception des machines électriques.

Ce travail a aboutit à la réalisation de contextes dédiés aux machines électriques dans Flux. Même si le travail s'est concentré sur les machines sans balais à aimants permanents, le développement a commencé pour les machines asynchrones, les machines à courant continu et les machines à griffes. On envisage de continuer avec les moteurs à reluctance variable et les moteurs à flux axial.

Pour des utilisateurs non spécialistes, ces contextes donnent accès aux méthodes de calcul par éléments finis à travers une définition métier du problème à résoudre plus simple et plus rapide. Ils accélèrent également dans une phase de conception la recherche d'une solution technique. De plus ils capitalisent une certaine connaissance du métier moteur et proposent à l'utilisateur un certain nombre de topologies et de méthodes d'analyses classiques des machines électriques. Enfin l'introduction de modèle de données métier permet un couplage avec les autres logiciels de la conception. Une fonction d'import de moteurs SPEED a été développée et permet de passer plus facilement du prédimensionnement au dimensionnement. De plus les études classiques permettent l'identification des schémas équivalents des moteurs, et un export au format VHDL-AMS vers les logiciels de la simulation système est prévu.

Le prototype d'un logiciel dédié à la conception des moteurs appelé FluxMotor a été développé. Il utilise Flux et ses contextes dédiés comme serveur de calcul et des outils pour la visualisation du bobinage ou la génération de rapports ont été réalisés.Grâce à une approche multi-projet, l'utilisateur peut capitaliser l'ensemble des moteurs étudiés et les résultats obtenus pendant le processus conception. L'architecture proposée pour ce logiciel prévoit le pilotage d'autres logiciels utiles à la conception. Dans un premier temps, les outils d'optimisation seront étudiés. En effet ces outils sont de plus en plus utilisés pendant la conception mais sont encore difficiles d'accès. FluxMotor proposera une définition métier et donc plus simple du problème d'optimisation. Basé sur une architecture client/serveur, il pourra distribuer les calculs nécessaires à l'optimisation sur plusieures machines. Cependant l'optimisation sur des modèles numériques pose encore

des problèmes de coût et des modèles compacts type surface de réponse sont souvent utilisés. Grâce à notre environnement métier, on pourra étudier des méthodes utilisant la connaissance de modèles analytiques simples pour améliorer la construction des surfaces de réponse. Ces travaux sur l'optimisation devraient nous amener à réfléchir à la modélisation du processus de conception et à la capitalisation de l'historique du dimensionnement. L'élaboration d'un modèle commun pour les logiciels moteur (partage des même paramètres) et les divers couplages entre logiciels devraient conduire à une interface unique pour la conception des machines électriques.

La méthodologie de personnalisation du logiciel Flux développée dans le cadre de ce projet moteur est réutilisable pour d'autres métiers. La notion de contexte métier chargeable dynamiquement propose à l'utilisateur d'introduire de nouveaux objets dans Flux et une nouvelle interface. Ces contextes ont l'avantage de simplifier l'utilisation du logiciel Flux. De plus ils capitalisent de la connaissance métier. Ils nous aident également à mieux comprendre le métier des utilisateurs et ainsi à mieux anticiper leurs attentes. Avec le développement de XBuilder, il est maintenant possible pour un expert Flux de développer sa propre personnalisation. Ceci devrait faciliter le développement de nombreuses interfaces métier dans l'avenir. Grâce à ce générateur d'applications métier et à l'architecture pilotée par modèle, une grande partie du code est factorisé dans la machine virtuelle. Ainsi, malgré une probable augmentation du nombre de logiciels métiers, le coût de maintenance est maîtrisé. Enfin cette approche facilite le couplage de Flux avec les autres logiciels puisqu'elle permet de partager le même modèle de donnée. Ceci devrait permettre de mieux insérer le logiciel Flux dans des PLM (Product Life Management) par exemple.

Bibliographie

[1] Design of Brushless Permanent-Magnet Motors
J.R. Hendershot Jr., TJE Miller (1994)

[2] Guy Seguier, Francis Novotel,
Electrotechnique Industrielle, Technique et Documentation, 1996

Standards

[3] IEEE Standard 115-1995
"Test Procedures for Synchronous Machines"

[4] Commission Electrotechnique Internationale (CEI), Publication 34-4, 1985
"Machines électriques tournates. Partie 4 : Méthodes pour la détermination à partir d'essais des grandeurs des machines synchrones."

Modèles équivalents et calcul des paramètres

[5] Jacek F. Gieras, Ezio Santini, Mitchell Wing, "Calculation of synchronous reactances of Small Permanent-Magnet Alternating-Current Motors : Comparison of Analytical Approach and Finite Element Method with Measurements", IEEE Transaction on Magnetics, vol 34, no 5, September 1998

[6] Hans-Peter Nee, Louis Lefevre, Peter Thelin, Juliette Soulard
"Determination of d and q Reactances of Permanent-Magnet Synchronous Motors Without Measurements of the Rotor Position"
IEEE Transactions on Industry Applications, vol 36, no5 September/October 2000

[7] Fidel Fernandez, Aurelio Garcia-Cerrada, Roberto Faure
"Determination of Parameters in Interior Permanent-Magnet Synchronous Motors With Iron Losses Without Torque Measurement"
IEEE Transactions on Industry Applications, vol 37, no 5, Spetember/October 2001

[8] P. Zhou, M. A. Rahman, M. A. Jabbar, Field Circuit Analysis of Permanent Magnet Synchronous Motors
IEEE Transactions on Magnetics, vol 30, no 4, July 1994

[9] Bernd Klöckl, Measurements Based Parameter Determination of Permanent Magnet Synchronous Machines,
Diploma Thesis, Graz University of Technology, 2001

[10] A. B. Dekordi, A. M. Gole, T. L. Maguire,
"Permanent Magnet Synchronous Machine Model for Real-Time Simulation"
International Conference on Power Systems Transients (IPST'05), Paper IPST05-159, Montreal June 2005

[11] Tomy Sebastian, Gordon R. Slemon
"Transient Modeling and Performance of Variable-Speed Permanent-Magnet Motors"
IEEE Transactions on Industry Applications, vol 25, no 1, January/February 1989

[12] Stefan Henneberger, Uwe Pahner, Kay Hameyer, Ronnie Belmans
"Computation of a Highly Saturated Permanent Magnet Synchronous Motor for a Hybrid Electric Vehicule"
IEEE Transactions on Magnetics, vol 33, no 5, Spetember 1997

[13] Y.-K. Chin, J. Soulard, "Modeling of Iron Losses in Permanent Magnet Synchronous Motors with Field-Weakening Capabiblity for Electric Vehicules
International Journal of Automotive Technology, vol 4, no 2, p 87-94, 2003

[14] Longya Xu, Xingyi Xu, Thomas A. Lipo, Donald W. Novotny,
"Vector Control of a Synchronous Reluctance Motor Including Saturation and Iron Losses"
IEEE Transactions on Industry Applications, vol 27, no 5, September/October 2001

[15] Julius Luukko, "Direct Torque control of permanent magnet synchronous machines - analysis and implementation"
Lappeenranta University of Technology, Finland, 2000

[16] Albert Foggia,
"Méthodes de calcul des inductances de fuites",
Techniques de l'ingénieur, traité génie électrique, D3 440, 2-1999

Bobinage

[17] Jacques Saint-Michel, "Bobinage des machines tournantes à courant alternatif"
Techniques de l'ingénieur, traité génie électrique, D3 420

[18] Flux 3D application technical paper, End winding characterization

[19] Stéphanie Richard, "Etude électromagnétique des parties frontales des alternateurs en régime permanents et transitoires"
Thèse de doctorat, Institut National Polytechnique de Grenoble, 1994

[20] P.L. Alger, "Induction Machines - Their Behaviour and Uses",
Gordon and Breach, Science publisher inc., New York, 1970

[21] S. Williamson, M.C. Begg,
"Calculation of the resistance of induction motor end rings",
IEEE Proceeding, Part B, vol 133, 1986

[22] P. Trickey, "Induction motor ring width",
IEEE Trans. Amer. Inst. Elect. Eng., vol 55, pp. 144-150, 1936

Défluxage

[23] Shigeo Morimoto, Tomohiro Ueno, Masayuki Sanada, Yoji Takeda, Takao Hirasa, "Variable Speed Drive System of Interior Permanent Magnet Synchronous Motors for Constant power Operation"
PCC Yokohama, TH0406-9/93

[24] Nicola Bianchi, Silverio Bolognani,
"Parameters and Volt-Ampere Ratings of a Synchronous Motor Drive For Flux-Weakening Applications",
IEEE Transactions on Power Electronics, vol 12, no 5, Spetember 1995

[25] Ayman M. EL-Refaie, Donald W. Novotny, Thomas M. Jahns,
"A Simple Model For Flux Weakening in Surface PM Synchronous Machines Using Back-to-Back Thyristors"
IEEE Power Electronics Letters, vol 2, no 2, June 2004

Optimisation

[26] Laurent Jolly, M. A. Jabbar, Liu Quinghua
"Design Optimization of Permanent Magnet Motor Using Response Surface Methodology and Genetics Algorithms"
IEEE Transactions on Magnetics, vol 41, no 10, October 2005

[27] Jean-Louis Coulomb, "Optimisation", chapitre 8 de "Electromagnétisme et problèmes couplés", "Electromagnétisme et éléments finis 3D", EGEM, Hermes, 2002

[28] J. W. Bandler, Q. S. Cheng, S. A. Darkoury, A. S. Mohamed, M. H. Bakr, K. Madsen, Jacob Sondergaard,
"Space Mapping : The State of the Art",
IEEE Transactions microwave Theory tech., vol 52, no 1, January 2004

[29] A. Kechroud, "Association d'un modèle précis mais coûteux et d'un modèle approché mais rapide pour accélérer l'optimisation d'une soupape magnétique",
Mémoire Master 2 Recherche, G2Elab, 2006

Informatique

[30] Yves Maréchal, "Vers une nouvelle génération de logiciels de simulation pour l'électromagnétisme et les disciplines connexes",
Habilitation à diriger des recherches, Institut National Polytechnique de Grenoble, 2001

[31] Yves Souchard, "Réalisation d'une plate-forme informatique dédiée au métier du génie électrique autour des logiciels Flux. Application à la réalisation de logiciels métiers."
Thèse de doctorat, Institut National Polytechnique de Grenoble 2005

[32] Unified Modeling Language, V2.0, http ://www.uml.org

[33] Model Driven Architecture, http ://www.omg.org/mda

[34] Meta Object Facilities, V1.4, http ://www.omg.org/mof

[35] Remote Method Invocation,
http ://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

[36] XML en concentré, éditions O'reilly, 2005

[37] XSL Transformation, http ://www.w3.org/TR/xslt

[38] Extensible StyleSheet Language (XSL), http ://www.w3.org/TR/xsl

[39] IEEE Standard VHDL-AMS Reference Manual, IEEE Std 1076.1-1999, IEEE, 1999

Logiciels

[40] Flux, a finite element simulation environment for Electrical Engineering, http ://www.cedrat.com

[41] Portunus, a system simulation software dedicated to control command, http ://www.adapted-solutions.com

[42] SPEED, Scottish Power Electronics ans Electric Drives consortium,
Speed Laboratory, Glasgow, Scotland, http ://www.speedlab.co.uk

[43] MotorCAD, http ://www.motor-design.com/

[44] CIVA, htttp ://www-civa.cea.fr

Publications

[45] G. Lacombe, Y. Soucahrd, C. Chantrel, E. LeCathelinais, G. Jérôme, X. Brunotte, Y. Marechal, J. Wallace, D. Dorell,
"Benefits of Model Driven Architecture for simulation software design, application for dedicated motor tool"
Conference on the computation of Electromagnetic Fields, Compumag 2005, Shenyang, Liaoning, China, june 2005

[46] G. Lacombe, A. Foggia, Y. Marechal, X. Brunotte,
"From general finite element software to engineering-focused software"
International Symposium on Electric and Magnetic Fields, EMF 2006, Aussois, France, june 2006

[47] G. Lacombe, A. Foggia, Y. Marechal, X. Brunotte,
"Innovating engineering-focused software solution on rotating electrical machines"
International Conference on Electrical Machine, ICEM 2006, Chania, Crete

[48] G. Lacombe, A. Foggia, Y. Marechal, X. Brunotte, P. Wendling,
"From general finite element simulation software to engineering-focused software,
Example for brushless permanent magnet motor design"
IEEE Transactions on magnetics, vol 43, no 4, april 2007

[49] G. Lacombe, A. Foggia, Y. Marechal, X. Brunotte,
"Innovating engineering-focused software for rotating electrical machines design"
Recent Advances in Aerospace Actuation Systems and Components, Toulouse, France, june 2007

Résumé :

Les logiciels Flux sont des logiciels de modélisation des phénomènes électromagnétiques par la méthode des éléments finis. Ils sont particuliérement bien adaptés pour l'analyse fine des machines électriques tournantes. Cependant leur interface généraliste rend la définition d'un calcul assez longue ce qui limite leur utilisation. Les travaux présentés proposent de faire une analyse du métier de la conception des machines tournantes, de manière à définir et réaliser un nouveau logiciel dédié, basé sur les capacités de calcul de Flux . En capitalisant un certains nombre de concepts métier comme des topologies de moteur ou des méthodes d'analyses classiques, ce logiciel donne un accès plus simple et plus rapide aux outils numériques dans le processus de conception des moteurs. De plus, les concepts informatiques utilisés ont permis l'élaboration d'une méthodologie de personnalisation du logiciel Flux réutilisable pour d'autre métiers et accessible à un utilisateur avancé.

Mots clés :

Logiciel de simulation métier, Conception des machines électriques, Eléments finis, Architecture pilotée par modèle.

Abstract:

Flux is finite element software for electromagnetic simulation. It is well-adapted for accurate analysis of electrical rotating machine. However its use is still limited by a generalist point of view in the graphical user interface and a long computation's definition time. Our work proposes to analyse electrical machine design process, and then to define and realize a new dedicated software based on Flux computation capabilities. This software capitalizes dedicated concepts like motor topologies or classical methods of analysis and gives an easier and faster access to numerical methods in the design process of electrical machine. This paper also deals with the programming concepts used for the development of this dedicated software. Thanks to this work, the process to customize Flux software is reusable for other jobs and can be used by an advanced user of Flux.

Keywords:

Engineering-focused software, Rotating electrical machine design, Finite Elements, Model Driven Architecture.