Contribution à la simulation de la stimulation magnétique transcrânienne: vers une approche dirigée par les modèles

Sébastien Luquet

▶ **To cite this version:**

Sébastien Luquet. Contribution à la simulation de la stimulation magnétique transcrânienne: vers une approche dirigée par les modèles. Bio-informatique [q-bio.QM]. Université Blaise Pascal - Clermont-Ferrand II, 2009. Français. NNT : 2009CLF22001 . tel-00724476

HAL Id: tel-00724476

https://theses.hal.science/tel-00724476

Submitted on 21 Aug 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : D. U : 2001
EDSPIC : 471

Université Blaise Pascal

THÈSE

pour obtenir le grade de

Docteur de L'Université Blaise Pascal
Ecole Doctorale Sciences Pour L'Ingénieur

Discipline Informatique

présentée par

M. Sébastien Luquet

Contribution à la simulation de la Stimulation Magnétique Transcrânienne : vers une approche dirigée par les modèles

---

sous la direction des Professeurs

Vincent Barra et David Hill

Soutenue publiquement le 14 décembre 2009

Rapporteurs :	M. Jean Bézivin, Professeur, Université de Nantes
	M. Eric Ramat, Professeur, Université du Littoral
Directeurs :	M. Vincent Barra, Professeur, Université Blaise Pascal
	M. David Hill, Professeur, Université Blaise Pascal
Examinateurs :	M. Alain Quilliot, Professeur, Université Blaise Pascal
Co-encadrant :	M. Eric Innocenti, Maître de Conférences, Université de Corse.
Invité :	M. Christophe Haug, Soluscience

N° d'ordre : D. U : 2001
EDSPIC : 471

Université Blaise Pascal

THÈSE

pour obtenir le grade de

Docteur de L'Université Blaise Pascal
Ecole Doctorale Sciences Pour L'Ingénieur

Discipline Informatique

présentée par

M. Sébastien Luquet

Contribution à la simulation de la Stimulation Magnétique Transcrânienne : vers une approche dirigée par les modèles

---

sous la direction des Professeurs

Vincent Barra et David Hill

Soutenue publiquement le 14 décembre 2009

Rapporteurs :	M. Jean Bézivin, Professeur, Université de Nantes
	M. Eric Ramat, Professeur, Université du Littoral
Directeurs :	M. Vincent Barra, Professeur, Université Blaise Pascal
	M. David Hill, Professeur, Université Blaise Pascal
Examinateurs :	M. Alain Quilliot, Professeur, Université Blaise Pascal
Co-encadrant :	M. Eric Innocenti, Maître de Conférences, Université de Corse.
Invité :	M. Christophe Haug, Soluscience

À mes Grands-parents

Remerciements

Je voulais remercier toutes les personnes qui m'ont soutenu et sans qui je n'aurais jamais soutenu.

Introduction

Le cerveau fonctionne sur la base d'impulsions électriques. L'électromagnétisme prouve qu'à un courant électrique est associé un champ magnétique. En conséquence, tout champ magnétique induit un courant et inversement. C'est sur la base de cette caractéristique que la Stimulation Magnétique Transcrânienne dite *SMT* permet de traiter des pathologies du cortex cérébral. Cette technique peut se substituer dans certaines mesures à la technique dite des électrochocs ou *ECT* qui est invasive, douloureuse et risquée. La SMT nécessite d'utiliser une bobine électrique qui parcourue par un courant de forte intensité pendant un temps très court génère un champ magnétique qui influe sur le fonctionnement neuronal du patient. La bobine doit pour cela être placée à proximité du cortex cérébral, contre la tête du patient. La difficulté réside dans la capacité à connaître de manière assez précise la zone du cortex cérébral à stimuler. Aussi, avant de pouvoir envisager une séance SMT, il convient de calculer précisément l'orientation et le positionnement des bobines de l'appareillage. Il est courant que cela se fasse de manière empirique, il n'existe pas d'outils informatiques permettant d'estimer orientation et position. En effet, la SMT est encore une technique ancrée dans le domaine de la recherche, bien qu'elle semble offrir des possibilités intéressantes notamment dans le traitement de pathologies telles que les dépressions, les douleurs chroniques, certaines formes de schizophrénies, etc. L'objectif général de ce travail est de proposer un simulateur informatique permettant d'appréhender le travail préparatoire d'une séance de SMT afin de faciliter le placement de la bobine sur le cortex cérébral d'un patient. Ce travail est composé des sous-objectifs suivants :

- participer à l'amélioration des connaissances des mécanismes de la SMT qui sont encore de nos jours mal connus
- faciliter l'utilisation de la SMT en traitement de routine grâce à l'utilisation de l'outil informatique
- fournir un outil de visualisation 3D adapté pour le milieu médical.

Dans un premier temps, nous présentons la démarche pragmatique utilisée afin d'obtenir un résultat utilisable à moyen terme. Nous proposons ensuite une démarche de

rétro-ingénierie et de refactoring car celle-ci s'est révélée indispensable a posteriori. Le manuscrit est organisé en six chapitres.

Dans un premier chapitre, nous présentons les fondements théoriques du domaine d'application de la présente étude. Nous expliquons les mécanismes fondamentaux liés à l'activité neuronale ainsi que les principales techniques qui sont utilisées en stimulation neuronale. Parmi celles-ci, nous expliquons en quoi la Stimulation Magnétique Transcrânienne (SMT) est une technique intéressante et nous présentons quelques-unes de ses applications médicales. Nous rappelons les principes généraux de l'électromagnétisme et nous présentons les modèles physiques de la littérature qui décrivent le fonctionnement de la SMT.

Dans un deuxième chapitre, nous présentons les concepts et les enjeux de l'Ingénierie Dirigée par les Modèles (IDM), fondement de notre démarche de rétro-ingénierie et de refactoring que nous développerons in fine dans le chapitre 6. Nous allons pour cela proposer des modèles et des méta-modèles qui permettront de faire évoluer l'architecture logicielle développée.

Dans un troisième chapitre, nous présentons les modèles et méthodes proposés dans la littérature pour calculer les effets de la stimulation magnétique transcrânienne. Nous présentons les différents travaux théoriques à partir desquels nous allons définir un modèle de calcul du champ électromagnétique. La physique du modèle envisagé est détaillée.

Dans un quatrième chapitre, nous présentons comment nous avons réalisé le module de calcul et la méthode mathématique retenue.

Dans un cinquième chapitre, nous expliquons comment le noyau de calcul a été intégré dans un logiciel de visualisation 3D, afin de faciliter la lecture et l'interprétation des résultats des expériences. Nous traitons de l'intégration des éléments logiciels, des bibliothèques graphiques, des bibliothèques de calculs, et des implémentations d'algorithmes de traitement d'image.

Dans un sixième chapitre, nous présentons les modèles et méta-modèles qui pourraient être utilisés pour effectuer une refactorisation du simulateur obtenu à la fin de son premier cycle de développement.

Enfin, une conclusion revient sur le travail réalisé et les perspectives présentées.

Table des matières

Remerciements **7**

Introduction **8**

Table des matières **10**

Table des figures **15**

Glossaire **17**

1 Stimulation Neuronale **19**

1.1 Support et principes de l'activité neuronale 19

1.2 Les techniques de stimulation neuronale 24

1.2.1 Les Electrochocs 24

1.2.2 La stimulation du cortex moteur par l'implantation d'électrodes 24

1.2.3 La Stimulation Magnétique Transcrânienne 25

1.2.4 Utilisation de la SMT en recherche médicale 26

1.3 Principes d'électromagnétisme 29

1.4 Principe de fonctionnement et modèles physiques de la SMT 31

1.4.1 Niveau macroscopique 32

1.4.2 Niveau microscopique 33

1.5 Conclusion 33

2 L'ingénierie dirigée par les modèles **35**

2.1 Présentation de l'ingénierie dirigée par les modèles 35

2.1.1 Le modèle 36

2.1.2 Le méta-modèle 37
2.1.3 La transformation 38
2.1.4 Application de l'IDM dans le contexte de l'étude 38
2.2 IDM et refactorisation (*refactoring*) 39
2.2.1 Les outils de rétro-ingénierie 40
2.2.2 La programmation globale 40
2.3 Conclusion sur la démarche de rétro-ingénierie adoptée 41

3 Modélisation et calcul des effets de la SMT 43
3.1 Calcul des effets de la stimulation 44
3.1.1 Modèle sphérique 44
3.1.2 Méthode des éléments finis 45
3.1.3 Méthode d'impédance 3D 46
3.2 Aide à la conduite de séance de SMT 48
3.2.1 Appareil de fixation 49
3.2.2 Recalage par dispositif optique 49
3.2.3 Neuro-navigation 49
3.3 Modélisation des effets de la stimulation sur les neurones 50
3.4 Validation 50
3.4.1 Confrontation de résultats 50
3.4.2 Mesure des effets de la SMT par l'IRM 51
3.4.3 Comparaison des effets de la stimulation 51
3.5 Discussion 52
3.6 Conclusion 53

4 Le noyau de calcul du simulateur 55
4.1 De la forme des bobines 55
4.1.1 Bobine simple 56
4.1.2 Bobine en forme de 8 56
4.1.3 Bobine en forme de double cône 59
4.1.4 Autres formes de bobines 59
4.2 Modélisation par équation paramétrique 59

4.2.1 Intégration Numérique 60
4.2.2 Implémentation 62
4.2.3 Exemple d'utilisation 64
4.3 Conclusion 66

5 Le simulateur 69
5.1 Contexte de développement 69
5.1.1 Cadre de développement de l'interface graphique 69
5.1.2 De l'Interface Graphique et de la ligne de commande 70
5.1.3 De la gestion de versions 74
5.1.4 Intégration Logicielle 83
5.2 Modélisation de la Bobine 83
5.2.1 Arbre de Design 83
5.2.2 Exemple de modélisation : Une bobine en forme de 8 85
5.2.3 Exemple proche de la réalité physique 86
5.3 Modèle de cerveau et gestion des maillages 89
5.3.1 Marching-cubes 89
5.3.2 Placement relatif du maillage et de la bobine 94
5.3.3 Application de champ sur le modèle de surface du cortex . . . 94
5.4 Résultats 94
5.4.1 Calcul dans un plan 94
5.4.2 Importance de la distance locale 95
5.4.3 Diffusion 97
5.4.4 Quelques métriques sur le logiciel réalisé 99
5.5 Conclusion 101

6 Modèles et Méta-modèles pour la refactorisation (*refactoring*) 103
6.1 Génération de code et gestion de versions 104
6.1.1 Limitations liées à l'utilisation des générateurs de code . . . 104
6.1.2 Limitation liée au contenu des modèles générés 106
6.1.3 Description de la démarche de refactorisation du code 107
6.2 La méta-programmation 108

6.2.1 Principe de la méta-programmation 108
6.2.2 Avantages et inconvénients de la méta-programmation 111
6.2.3 Conclusion sur l'utilisation de la méta-programmation 112
6.3 Le modèle source de l'application 113
6.3.1 Le modèle source de l'application 114
6.3.2 Gestion des versions du modèle source de l'application 115
6.4 Intérêt et mise en œuvre des tests unitaires 122
6.4.1 Principe des tests unitaires 122
6.4.2 Importance des tests unitaires 123
6.5 Persistance, sérialisation et Object-Relational Mapping 124
6.5.1 Object Relational Mapping 125
6.5.2 Avantages, inconvénients et limites d'un ORM 126
6.5.3 Conclusion sur l'utilisation d'un ORM dans le cadre de l'IDM 128
6.6 Conclusion 129

Conclusion **131**

Bibliographie **136**

Annexe **150**

Table des figures

1.1 La structure d'un neurone moteur 20
1.2 Communication neuronale 22
1.3 Propagation d'un influx nerveux 22
1.4 La synapse chimique 23
1.5 La synapse électrique 23
1.6 Stimulation neuronale à partir d'ECT 25
1.7 La Stimulation Magnétique Transcrânienne 26
1.8 Un exemple de protocole de SMT répétée 28
1.9 Modélisation d'une fibre nerveuse par un câble électrique 32

2.1 Modèle et système modélisé 36
2.2 Le méta-modèle et modèle 37
2.3 Transformation 38
2.4 Cartographie simplifiée des outils de génie logiciel 41

3.1 Réseau de résistances 47
3.2 Boucle de courant et loi de Kirchoff 47

4.1 Module du champ magnétique généré par une bobine simple 57
4.2 Module du champ magnétique généré par une bobine double 58
4.3 Extrait du diagramme UML général 62
4.4 Vecteur champ potentiel magnétique 64
4.5 Cartographie de champ magnétique 65

5.1 Les 3 parties de l'interface graphique 87
5.2 Une modélisation de la bobine Medtronic MC-B70 88

5.3 Reconstruction du cortex à partir d'image IRM 90
5.4 Pas de focalisation dans certains cas pathologiques 96
5.5 Meilleure focalisation en tenant la bobine à *l'envers* 97
5.6 Visualisation de 3 coupes orthogonales d'une image IRM en 3D . . . 98
5.7 Extrait du graphique des dépendances 100

6.1 Assistant de création de classe d'Eclipse 105
6.2 Déclaration du composant graphique wxWColor 106
6.3 Déclaration du composant graphique wxWColor corrigée 106
6.4 Gestion de code généré sous forme de *vendor branch* 109
6.5 Rejouer des personnalisations de code 110
6.6 Initialisation d'une variable par le calcul d'une factorielle 110
6.7 Exemple de méta-programmation en Ruby 111
6.8 Ce listing donne le code équivalent après évaluation du listing 6.7 . . 111
6.9 Avantages et inconvénients liés de la méta-programmation 113
6.10 Exemple de Diagramme UML source d'application 114
6.11 Commande de génération et fichiers générés 115
6.12 Script de migration global de la base de données 116
6.13 Exemple de script de migration 117
6.14 Décoration des codes de l'application via le patron décorateur. 119
6.15 Définition d'un modèle **User** et de sa relation avec le modèle **Group** 119
6.16 Exemple de méthode ajoutée par méta-programmation 120
6.17 Diagramme UML généré à partir d'un fichier XMI/Ecore 121
6.18 Mise en place de tests sur une application 123
6.19 Sérialisation des paramètres des transformations géométriques 125
6.20 Correspondance Objet-Relationnel avec ActiveRecord 126
I Quelque part à la racine de l'arborescence d'un projet 151
II L'équivalent de la commande status 152
III Comparaison d'un fichier avant sa soumission au serveur 153
IV Annulation de modification 154

Glossaire

CVS : *Concurrent Versions System*. Logiciel de gestion de versions.

DTD : *Document Type Definition*. Document permettant de décrire le modèle d'un document XML. Si leur syntaxe est différente, la *DTD* remplit le même rôle que le *XSD*

EDP : Equations aux Dérivées Partielles

EEG : ElectroEncéphaloGraphie. Examen fonctionnel explorant l'activité électrique produite spontanément par les cellules nerveuses.

GUI : *Graphical User Interface*. Interface graphique qui permet à l'utilisateur de manipuler ses objets graphiquement (par opposition au programme en ligne de commande).

IDE : *Integrated Development Environment*. Environnement de Développement Intégré.

IDM : Ingénierie Dirigée par les Modèles. Formalisme informatique unifiant de nombreux espaces technologiques via la définition des modèles, méta-modèles et transformations.

IHM : Interface Homme Machine. *cf.* GUI.

IRM : Imagerie par Résonance Magnétique. L'imagerie par résonance magnétique nucléaire est une technique d'imagerie médicale permettant d'avoir une vue 2D ou 3D d'une partie du corps.

JSON : *JavaScript Object Notation*. Format de représentation de données issu du langage de programmation JavaScript. Un fichier json est directement évaluable par un interpréteur JavaScript.

LCR : Liquide Céphalo-Rachidien.

MDA : *Model Driven Architecture*. Architecture Dirigée par les Modèles

MEG : *Magnétoencéphalographie*. La magnétoencéphalographie est une technique de mesure des champs magnétiques induits par l'activité électrique des neurones du cerveau.

ORM : *Object-Relational Mapping*. Technique de programmation informatique qui permet de faire la correspondance entre les concepts des bases de données relation-

nelles et les concepts de la programmation orientée objet.

SMT : Stimulation Magnétique Transcrânienne.

SVK : Logiciel de gestion de versions décentralisée reposant sur SVN.

SVN : Ou SubVersion. Logiciel de gestion de versions. Successeur de CVS

SOAP : *Simple Object Access Protocol* est un protocole de communication basé sur XML

Thread : Fil d'exécution d'un programme. Un processus peut disposer de plusieurs fils d'exécution concurrents dans le même espace mémoire.

TMS : *Transcranial Magnetic Stimulation* voir SMT.

Voxel : cube élémentaire.

UML : *Unified Modeling Language*. Langage graphique de modélisation généraliste.

VTK : *Visualization ToolKit*. Bibliothèque de visualisation de données Open Source.

XML : *eXtensible Markup Language*. Langage de balisage extensible qui est devenu un standard pour l'échange d'informations entre systèmes d'information.

XSD : *XML Schema Description*. Le XSD permet de définir la structure d'un document XML.

YAML : *YAML Ain't Markup language*. Langage de sérialisation de données.

Chapitre 1

Stimulation Neuronale

Nous présentons dans ce chapitre les fondements théoriques du domaine d'application de la présente étude. Dans une première partie, nous expliquons la structure et les mécanismes fondamentaux liés à l'activité neuronale, préalables indispensables à la compréhension générale du sujet de l'étude : *la stimulation neuronale*. Nous expliquons plus particulièrement, comment la perturbation des processus électriques présents dans le cerveau engendre une modification du comportement et des sensations d'un individu. Il en résulte que la modification des processus électriques neuronaux peut être utilisée à des fins médicales, notamment pour soigner certaines pathologies. Dans une deuxième partie, nous présentons les techniques existantes qui se fondent sur ce principe et qui sont utilisées dans le domaine. Parmi celles-ci, nous expliquons en quoi la *Stimulation Magnétique Transcrânienne* (SMT) est une technique d'avenir et nous présentons quelques-unes de ses applications médicales. Dans une troisième partie, nous rappelons rapidement les principes généraux de l'électromagnétisme, afin d'aider à la compréhension du fonctionnement interne de la SMT et des modèles physiques de la littérature qui en découlent, que nous détaillons plus avant dans une quatrième partie. Enfin, une conclusion générale vient clore ce chapitre.

1.1 Support et principes de l'activité neuronale

Les *neurones* constituent le support de *l'activité neuronale*. Ce sont des cellules nerveuses excitables qui génèrent et transmettent des signaux électriques. Ils consti-

tuent les unités fonctionnelles du système nerveux. Pour cela, ils acheminent des messages sous forme d'*influx nerveux* entre les différentes parties du corps. C'est la *membrane plasmique* des neurones qui est le siège du déclenchement et de la propagation des influx nerveux.

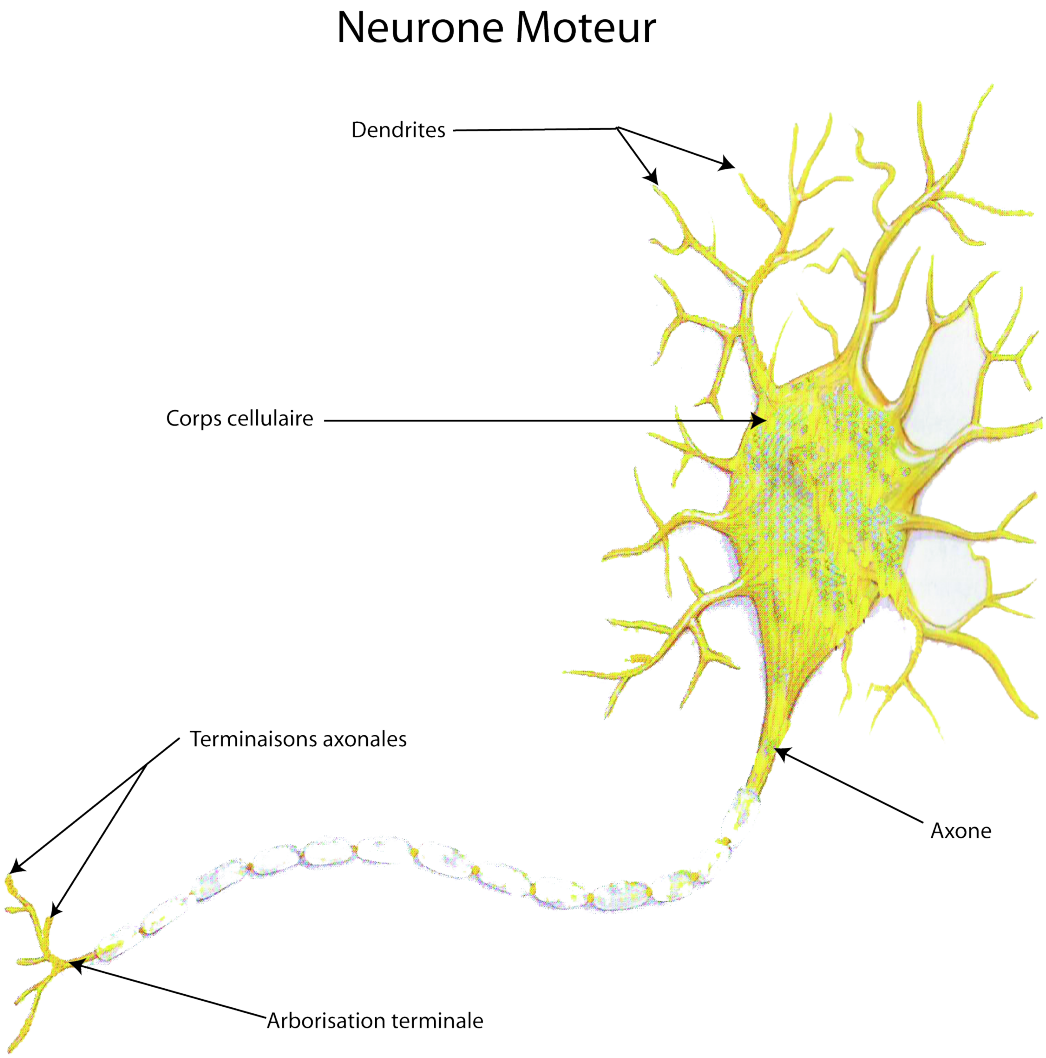

FIGURE 1.1 – La structure d'un neurone moteur

Bien qu'il existe de nombreux neurones différents, il est d'usage de décrire les prolongements neuronaux à partir de l'exemple du *neurone moteur*. Les neurones possèdent deux types de prolongements : les *axones* et les *dendrites* (cf. Figure 1.1). Les dendrites des neurones sont des prolongements courts aux ramifications diffuses qui forment la structure réceptrice chargée de la collecte de l'information. Chaque

neurone est muni d'un axone appelé neurofibre pour les plus longs, qui achemine les influx nerveux. L'extrémité d'un axone se divise en de très nombreuses ramifications qui constituent l'*arborisation terminale* (*terminaisons axonales*). L'influx nerveux est produit dans la partie haute de l'axone appelée *zone gâchette* et conduit jusqu'aux *terminaisons axonales* qui forment la structure sécrétrice des neurones. L'influx nerveux entraîne alors la libération dans l'espace extracellulaire de neurotransmetteurs qui sont des substances chimiques emmagasinées dans les terminaisons axonales. Ainsi, les neurotransmetteurs excitent ou inhibent les signaux entre les neurones. L'activité cérébrale qui nous permet de nous mouvoir, de sentir, de penser est intimement liée à cet ensemble complexe de processus électrico-chimiques (cf. Figure 1.1).

Les neurones sont sensibles aux *stimuli*. Lorsqu'un neurone reçoit un stimulus, il produit un influx électrique et le conduit le long de son axone. Ce phénomène électrique est à la base du fonctionnement du système nerveux. Les stimuli modifient la perméabilité aux ions de la membrane du neurone en ouvrant des canaux voltage dépendants situés sur la membrane plasmique.

Ainsi, le fonctionnement du système nerveux repose sur la circulation de l'information dans des réseaux de chaînes de neurones reliés par des *synapses*. Une synapse (sunapsis = liaison, point de jonction) permet le transfert de l'information d'un neurone à un autre. Il existe deux types de synapses : les *synapses électriques* et les *synapses chimiques*. Les synapses électriques ont pour rôle de permettre la circulation des ions entre les neurones. Les synapses chimiques ont pour rôle de libérer et de recevoir des *neurotransmetteurs* chimiques. Ainsi, la communication neuronale repose sur des processus électriques et des processus chimiques qui sont localisés au niveau des synapses. D'infimes impulsions électriques se propagent continuellement le long des axones neuronaux et transitent d'un neurone à l'autre au niveau des synapses via ces neurotransmetteurs. La figure 1.2 décrit cette communication neuronale.

Il ne s'agit pas ici d'un déplacement d'électrons classiques, comme dans le cas d'un courant électrique, mais d'un signal électrique qui se propage de proche en proche, grâce à la modification de la concentration des ions présents de part et

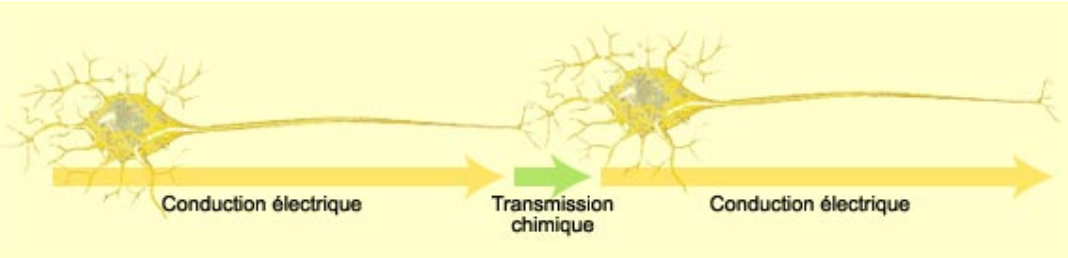

FIGURE 1.2 – La communication neuronale repose sur des conductions électriques le long des axones, et chimiques (ou électriques) au niveau des synapses. Source des dessins : Cerveau à tous les niveaux.

d'autre de la membrane des cellules. Cela crée une différence de potentiel qui engendre une dépolarisation qui se déplace le long de l'axone.

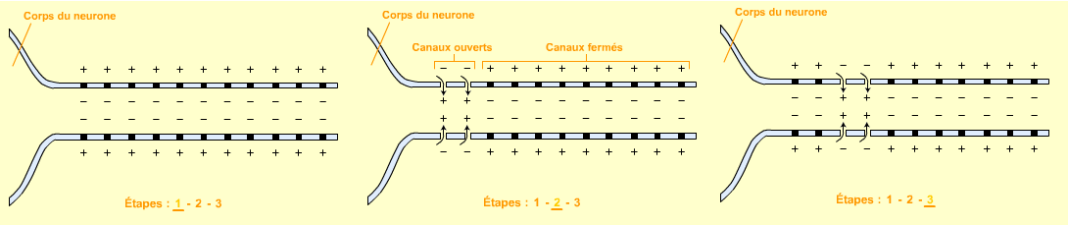

FIGURE 1.3 – Propagation d'un influx nerveux : la dépolarisation se déplace le long de l'axone. Source des dessins : Cerveau à tous les niveaux

De cette dépolarisation naît une différence de potentiel qui constitue l'influx nerveux qui circule le long d'un axone jusqu'à arriver à une synapse. Le système nerveux est essentiellement constitué de synapses chimiques dans lesquelles la transmission de l'information se fait via des neurotransmetteurs (cf. fig. 1.4). Les neurotransmetteurs libérés dans la *fente synaptique* permettent de propager l'information au neurone suivant.

Dans le cas des synapses électriques, les neurones sont en contact direct via des jonctions communicantes appelées *nexus* qui permettent aux ions de se propager directement d'une cellule à une autre. Il n'y a, dans ce cas, aucune action de neurotransmetteur quant au processus de communication neuronale.

Au niveau du neurone, il y a intégration de l'ensemble des quantités électriques provenant des différentes synapses qui peuvent être aussi bien excitatrices qu'inhibi-

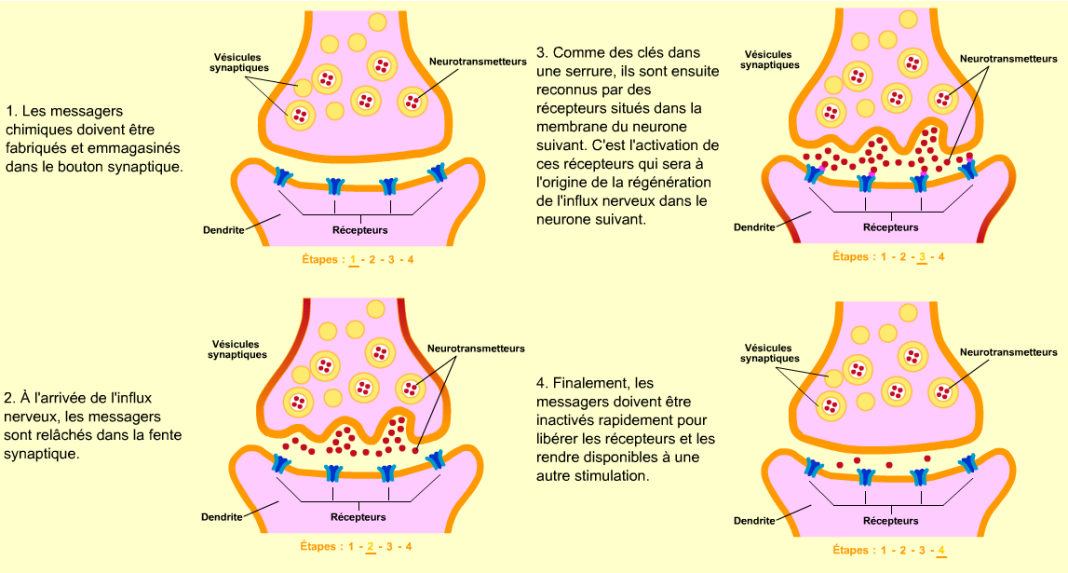

FIGURE 1.4 – La synapse chimique. Les neurotransmetteurs libérés dans la fente synaptique permettent de propager l'information au neurone suivant. Source des dessins : Cerveau à tous les niveaux

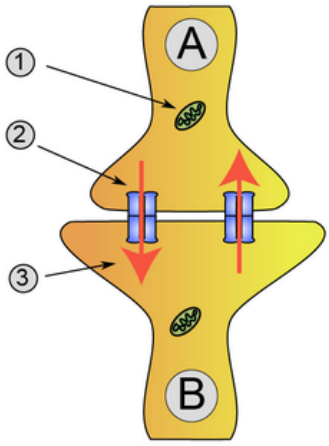

FIGURE 1.5 – La synapse électrique : 1. Mitochondrie, 2. Nexus, 3. Courant Ionique. L'information électrique passe d'un neurone à l'autre via des nexus sans faire appel aux neurotransmetteurs. Source de l'image : Wikipedia

trices. On parle de potentiel d'action qui se propage le long de l'axone, si celui-ci est supérieur au seuil de dépolarisation. Le lecteur intéressé par le sujet pourra consulter les ouvrages [Bronzino, 1995] et [Marieb et al., 1993] qui proposent des explications

plus détaillées.

La modification de l'activité cérébrale ou stimulation neuronale, engendrée par la perturbation des informations électriques qui circulent dans le cerveau peut modifier les sensations, les réactions et les mouvements de l'individu. De ce fait, il est possible de reproduire temporairement les symptômes de certaines maladies. Cette caractéristique est d'autant plus intéressante qu'elle permettrait dans le cas de certaines pathologies, d'en réguler les effets.

Après avoir décrit la nature de l'activité neuronale, nous présentons dans la partie suivante les techniques de stimulation neuronale existantes.

1.2 Les techniques de stimulation neuronale

1.2.1 Les Electrochocs

La modification de l'activité cérébrale est déjà utilisée en médecine via des techniques comme les électrochocs. Cela permet de soigner des pathologies telles que les douleurs chroniques ou les troubles de l'humeur.

Les électrochocs ou *ECT* (ElectroConvulsivoThérapie) consistent à placer des électrodes sur le scalp du patient. Il convient alors de faire circuler des courants électriques entre ces électrodes, ce qui engendre une modification globale de l'activité cérébrale à proximité de ces dernières. Ce traitement est cependant assez douloureux et n'est plus utilisé aujourd'hui qu'après anesthésie générale.

1.2.2 La stimulation du cortex moteur par l'implantation d'électrodes

Pour d'atteindre les zones du cerveau les plus profondes afin de réguler par exemple des mouvements anormaux (Parkinson, dystonies, ...), il peut être envisagé d'implanter les électrodes directement dans le cerveau du patient. Suivant la pathologie, la forme des électrodes et leur positionnement peuvent être adaptés. Les électrodes sont alors reliées à un stimulateur électrique placé dans la poitrine du patient afin de délivrer des impulsions et prévenir la crise.

Comme pour l'ECT, cette technique n'est pas dénuée de risque puisqu'il est

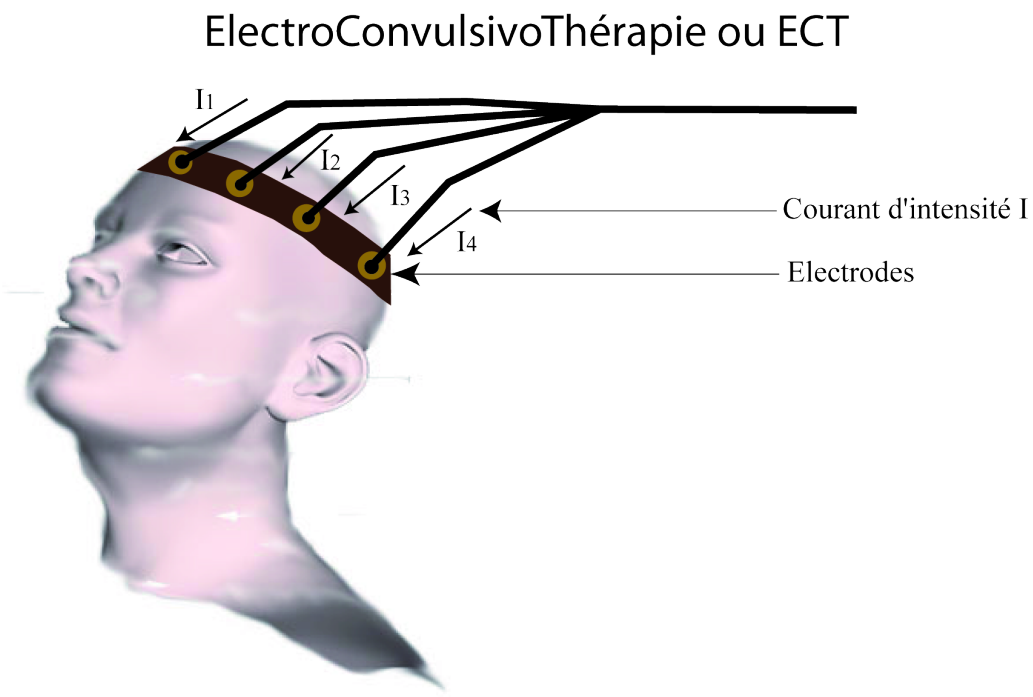

FIGURE 1.6 – Stimulation neuronale à partir d'ECT

nécessaire d'effectuer une opération sous anesthésie générale et d'ouvrir la boîte crânienne.

1.2.3 La Stimulation Magnétique Transcrânienne

La Stimulation Magnétique Transcrânienne ne présente pas les mêmes contraintes que les deux autres techniques que nous venons de décrire. Elle repose sur l'utilisation d'un champ magnétique qui, généré à proximité de la tête du patient, engendre une modification de l'activité cérébrale. L'opération s'effectue de façon non-invasive et est quasiment indolore. On peut alors envisager de l'utiliser en médecine ambulatoire pour soigner des pathologies telles que la dépression, pour laquelle l'utilisation de l'ECT présente plus d'effets secondaires désagréables que d'effets réellement bénéfiques. Les premières constatations des effets de la stimulation magnétique datent des expériences réalisées par d'Arsonval en 1896 [Arsonval, 1896] et la première utilisation clinique de la stimulation magnétique transcrânienne date de 1985 [Barker et al., 1985]

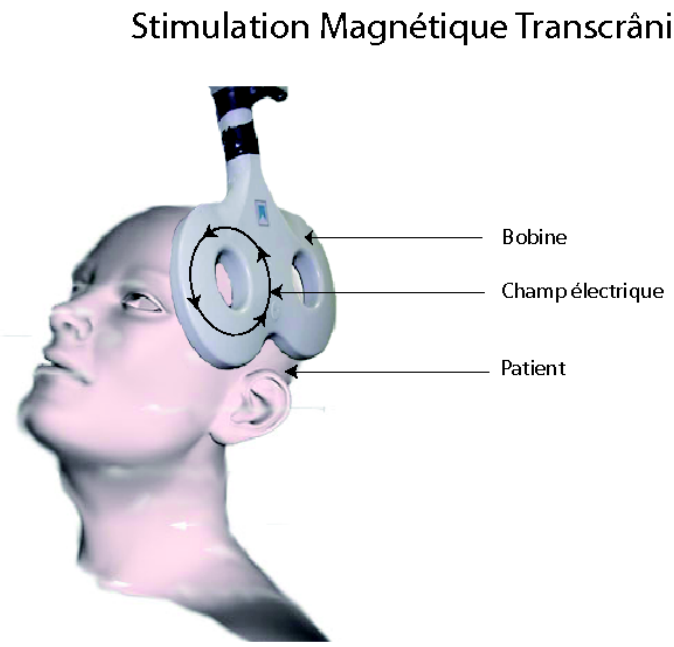

FIGURE 1.7 – La Stimulation Magnétique Transcrânienne

1.2.4 Utilisation de la SMT en recherche médicale

Les applications des effets de la SMT sont nombreuses, nous en présentons quelques unes. Pour une lecture plus complète, nous renvoyons volontiers le lecteur vers [George et al., 1999] qui est un état de l'art complet de toutes les utilisations. La SMT peut être utilisée dans le cas de paralysies partielle ou totale afin de tester les réflexes myotatiques. Un des exemples classiques de tests de réflexes ostéo-tendineux consiste à donner un coup sur le bas du genou (réflexe rotulien). La réponse réflexe est un mouvement de la partie inférieure de la jambe. Il est possible d'y substituer un test utilisant la SMT en créant une excitation sur la zone du cortex moteur correspondant aux muscles de la jambe. S'il y a réponse des muscles, cela tend à prouver que les connexions nerveuses sont assurées entre le cerveau et ces mêmes muscles, même si le patient ne peut pas les faire bouger consciemment [Wassermann

et al., 1992].

De nombreux travaux ont étudié la façon dont la SMT pouvait influencer la création des images sur la rétine et leur interprétation par le cerveau humain. Des stimulations affectant le cortex occipital peuvent provoquer de légers troubles momentanés de la vision. Il s'agit par exemple de l'apparition de taches dans le champ visuel du patient [Kamitani, 2001].

Les travaux [Epstein, 1996], [Pascual-Leone, 1991] et [Stewart et al., 2001] ont montré qu'il était possible d'interrompre la parole d'un patient en utilisant la SMT sur l'hémisphère du cortex cérébral qui la contrôle.

Il est également intéressant d'utiliser la SMT pour vérifier les chemins neuronaux entre le cerveau et un organe. Ainsi la SMT peut aider à la réalisation de cartographies du cerveau humain (cartographies somatotopiques) [George et al., 2003].

La SMT est également utilisée pour étudier la latence des réponses à des stimulations du cortex moteur. Quand un neurone est stimulé une première fois, il n'est pas rare que si la même stimulation est reproduite à l'identique, que celle-ci reste sans suite. En effet, lors du déclenchement d'un potentiel d'action, il y a de suite après, une période pendant laquelle le neurone se trouve dans un état réfractaire qui inhibe le départ d'éventuels nouveaux potentiels d'actions [Ruohonen et al., 1996]. Ainsi grâce à l'utilisation de la SMT en mode *double-impulsion* (c'est-à-dire deux impulsions, soit deux décharges de transistor dans la bobine, coup sur coup), il est possible de déterminer, en jouant sur la durée entre les deux impulsions, le temps des réponses observées.

En fait, l'utilisation d'une simple impulsion au-dessus de la tête du patient est a priori sans risque et ne présente que peu d'intérêt. Par contre, l'utilisation répétée peut provoquer des troubles chez le patient et il convient donc de respecter un certain nombre de précautions [Wassermann et al., 1996]. Aussi, précautions prises, les principales applications cliniques de la SMT reposent sur des séries d'impulsions multiples, on qualifie alors la SMT de répétée. Ses utilisations thérapeutiques sont présentées dans la sous-partie suivante.

1.2.4.1 Utilisation thérapeutique de la SMT

Les principales applications cliniques de la Stimulation Magnétique Transcrânienne utilisent les principes de la SMT répétée. Il s'agit d'effectuer des séries de stimulation afin que l'énergie délivrée au cours de la séance soit suffisamment importante pour avoir un effet thérapeutique. La figure 1.8 illustre un exemple de protocole de SMT répétée.

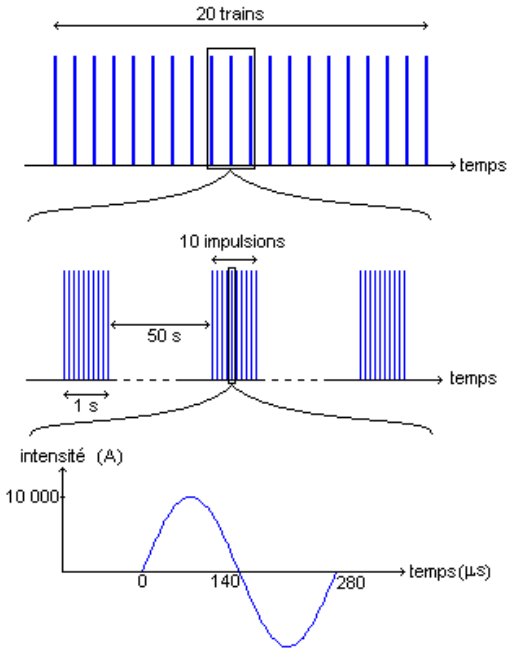

FIGURE 1.8 – Un exemple de protocole de SMT répétée. De bas en haut : on utilise une impulsion électrique sinusoïdale de forte amplitude (10 000A) ; les impulsions sont délivrées par groupe de dix (un train) pendant une seconde ; les trains sont espacés de 50s et on en délivre 20.

On utilise pour cela une impulsion électrique sinusoïdale de forte amplitude (10 000 A) ; les impulsions sont délivrées par groupe de dix, appelé 'train d'impulsions' pendant une seconde. Les trains sont espacés de 50 secondes et on en délivre 20.

Les effets anti-dépresseurs de la SMT ont donné lieu à de nombreuses études qui tendent à montrer une amélioration de l'état général d'un patient souffrant de dépression [Szuba et al., 2001], [George and Wassermann, 1994], [George et al., 1995].

La SMT est utilisée pour des patients souffrant du syndrome de Gilles de la Tourette

[Ziemann, 1997] et pour améliorer l'état des patients souffrant d'un stress post-traumatique [McCann et al., 1998]. Quelques études portent sur l'utilisation de la SMT pour améliorer l'état des patients souffrant de schizophrénie. Hoffman et al. montrent dans [Hoffman et al., 2000] que l'utilisation de la SMT répétée pouvait permettre une réduction des hallucinations auditives chez certains patients. Différentes études ont apporté des résultats contradictoires sur l'utilisation de la SMT pour soigner la maladie de Parkinson. Le lecteur interessé peut se référer à [Mally and Stone, 1999] et [Shimamoto et al., 2001] qui semblent montrer un effet bénéfique. Une autre application de ce domaine est l'utilisation de la SMT pour soigner la crampe de l'écrivain [Siebner et al., 1999]. Les études ayant pu montrer un effet positif de la SMT sur l'épilepsie sont peu nombreuses, citons [Wedegaertner et al., 1997] et [Theodore et al., 2002]). Cela est principalement dû au fait que les foyers responsables des crises sont souvent trop profonds dans le cerveau pour être atteints par la SMT qui produit des effets principalement à la surface du cortex cérébral. La SMT répétée est utilisée pour soigner des patients atteints de douleurs chroniques. On se référera aux deux études principales à ce sujet [Rollnik et al., 2002] et [Lefaucheur et al., 2001]. Enfin, l'utilisation préopératoire destinée à vérifier si le patient est sensible aux stimulations électromagnétiques avant l'implantation d'électrodes est envisagée dans certains travaux [Nguyen et al., 2003].

Notons finalement que quelque soit le type de SMT utilisé (répétée ou non), les effets à moyen et long terme n'ont pas encore pu être étudiés du fait de la récente utilisation de cette technique. Avant de détailler plus avant le fonctionnement interne de la SMT, il convient de rappeler les fondements physiques qui régissent ses lois.

1.3 Principes d'électromagnétisme

Avant d'aller plus loin, il est bon de rappeler les équations de Maxwell qui sont les bases de l'électromagnétisme. Ces équations régissent les relations entre les différents champs et potentiels électromagnétiques, aussi bien dans le vide que dans la matière.

Les équations dans le vide sont :

$$\text{div } \overrightarrow{E} \ = \ \frac{\rho}{\varepsilon_0} \tag{1.1}$$

L'équation de Maxwell-Gauss (1.1) exprime le fait qu'une charge électrique ρ est une source de champ électrique $\overrightarrow{E}$.

$$\text{div } \vec{B} \ = \ 0 \tag{1.2}$$

L'équation de conservation du flux (1.2) traduit le fait qu'il n'existe pas de monopole magnétique. Cela signifie que tout élément magnétique possède un pôle Nord et un pôle Sud.

$$\overrightarrow{\text{rot}} \ \overrightarrow{E} \ = \ - \ \frac{\partial \overrightarrow{B}}{\partial t} \tag{1.3}$$

L'équation de Maxwell-Faraday (1.3) traduit localement le phénomène d'induction électromagnétique découvert par Faraday : une modification dans le temps du champ magnétique $\overrightarrow{B}$ introduit un champ électrique $\overrightarrow{E}$.

$$\overrightarrow{\text{rot}} \ \overrightarrow{B} \ = \ \mu_0 \overrightarrow{j} \ + \ \varepsilon_0 \mu_0 \ \frac{\partial \overrightarrow{E}}{\partial t} \tag{1.4}$$

L'équation de Maxwell-Ampère (1.4) introduit le courant de déplacement $\overrightarrow{j}$ produit par une variation dans le temps du champ électromagnétique $\overrightarrow{B}$. Il s'agit d'une extension du théorème d'Ampère pour les régimes variables.

Le champ magnétique généré par un enroulement de fil de cuivre est donné par la loi de Biot & Savart. Celle-ci permet de calculer le champ magnétique généré dans le vide par un contour C à un endroit et à un moment donné :

$$\vec{B}(\vec{r}, t) = \frac{\mu_0}{4\pi} I(t) \oint_C \frac{d\vec{l} \times (\vec{r} - \vec{r'})}{|\vec{r} - \vec{r'}|^3} \tag{1.5}$$

où $\mu_0 = 4\pi 10^{-7} H$ est la perméabilité magnétique du vide, $I(t)$ l'intensité électrique parcourant l'enroulement électrique, $\vec{r'}$ un vecteur parcourant le contour C et dl un déplacement infinitésimal sur C.

Comme les champs magnétiques générés par la SMT reposent sur ces lois physiques, ces équations sont indispensables à la réalisation du modèle de simulation

développé et doivent faire partie intégrante du code objet sous-jacent. Nous allons maintenant présenter plus avant comment ces lois générales de la physique décrivent plus particulièrement le fonctionnement de la SMT.

1.4 Principe de fonctionnement et modèles physiques de la SMT

La SMT repose sur l'utilisation de champs magnétiques pour réaliser une stimulation neuronale. Le principe est relativement simple, même si la mise en application est délicate de par les ordres de grandeurs physiques utilisés. Une bobine, c'est-à-dire un enroulement de fils électriques, est placée contre la boîte crânienne d'un patient. Un courant de forte intensité, de l'ordre de 10 000 A y circule. Cela engendre un champ magnétique suffisamment important pour induire des modifications de l'activité cérébrale. Le courant ne circule que quelques microsecondes dans la bobine (environ 300 μs), ce qui reste du domaine de l'impulsion. Cependant, au niveau macroscopique, il y a génération d'un courant induit dans la tête du patient, dont le phénomène est décrit par l'équation 1.6

$$\frac{\partial \vec{B}}{\partial t} \propto \frac{\partial I}{\partial t} \tag{1.6}$$

D'après l'équation 1.6, c'est la variation du champ magnétique qui génère les courants induits, c'est la raison pour laquelle une intensité très importante est appliquée à la bobine pendant un temps très court. De tels ordres de grandeurs engendrent des puissances très importantes et entraînent par effet Joule un échauffement de l'appareil. C'est pour cela que celui-ci est donc souvent équipé d'un système de refroidissement.

La loi de Faraday vue précédemment (Equation 1.3) permet de déduire le champ électrique induit par le champ magnétique. Le champ électrique influe directement sur l'activité neuronale. Bien que cette influence puisse être envisagée de prime abord au niveau macroscopique, il est courant dans la littérature de considérer deux autres niveaux : le niveau semi-macroscopique et le niveau microscopique.

1.4.1 Niveau macroscopique

A un niveau macroscopique, les fibres nerveuses peuvent être considérées comme modélisables physiquement par un câble électrique ([Basser, 1993]). Il s'agit alors d'associer au câble représentant les fibres nerveuses des propriétés électriques comme la résistivité et la conductivité. Une modélisation possible est alors celle représentée sur la figure 1.9 ci-dessous.

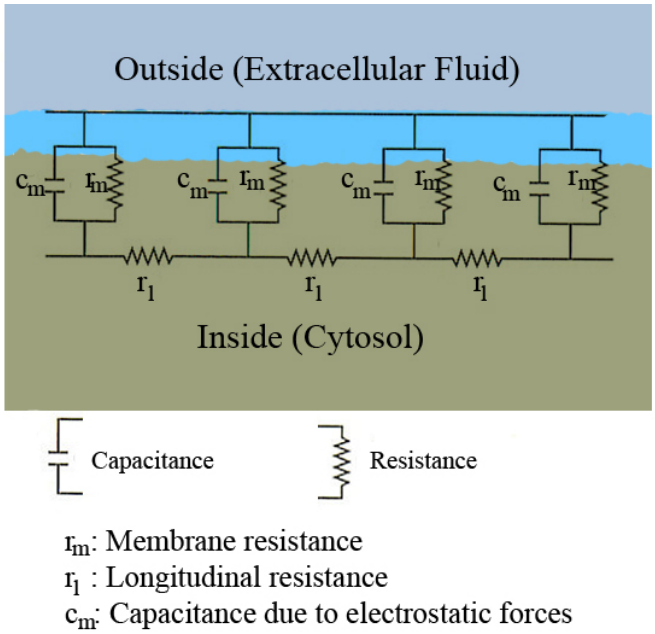

FIGURE 1.9 – Modélisation d'une fibre nerveuse par un câble électrique. Il s'agit d'un ensemble de cellules élémentaires composées de résistances et de condensateurs. Source de l'image : Wikipedia

L'introduction d'un tel circuit équivalent pour une fibre nerveuse provient d'observations expérimentales. Dès lors, il est possible d'utiliser les équations qui décrivent le comportement d'un câble passif lorsqu'il est soumis à un champ électromagnétique. Dans [Roth and Basser, 1990] et [Basser et al., 1992], les auteurs proposent de relier le potentiel membranaire au champ électrique par l'équation 1.7 :

$$\lambda^2 \frac{\partial^2 V}{\partial x^2} - V = \tau \frac{\partial V}{\partial t} + \lambda^2 E'_{||} \tag{1.7}$$

Où V est le potentiel transmembranaire, et $\lambda^2 = \frac{r_m}{r_l}$ et $\tau = c_m r_m$ et $E'_{||}$ est la variation du champ électrique dans la direction définie par le câble (fibre nerveuse).

[Ruohonen et al., 1996] proposent un modèle plus précis pour expliquer des phénomènes non prévus par la modélisation précédente. En effet, dans le cadre de stimulation magnétique des nerfs périphériques (faisceaux d'axones), la seule utilisation de $E'_{||} = -\frac{\partial E_x}{\partial x}$ dans l'équation de câble ne peut expliquer l'ensemble des positions de bobine conduisant à la génération du potentiel d'action évoqué. Ainsi, en analysant ces cas dits *inconsistants* que la modélisation précédente ne pouvait prédire, les auteurs introduisent un nouveau terme : la composante du champ électrique perpendiculaire au nerf $E_\perp$. L'équation générale devient alors :

$$\lambda^2 \frac{\partial^2 V}{\partial x^2} - V = \tau \frac{\partial V}{\partial t} + 2R(\alpha E'_{||} - E_\perp) \tag{1.8}$$

avec les mêmes notations que l'équation précédente et $\alpha = \frac{\lambda^2}{2R} = \frac{R_m}{4R_i}$

1.4.2 Niveau microscopique

Au niveau microscopique, on peut considérer que la diffusion des champs électromagnétiques dans le cortex cérébral crée des modifications des populations ioniques le long des gaines de myéline des axones. Ainsi soumis à des champs magnétiques, les populations en anions et en cations peuvent être modifiées. Si les concentrations sont suffisamment fortes, des dépolarisations peuvent se créer et générer des potentiels d'action qui vont dès lors se propager le long des axones. Les travaux de [Ruohonen, 1998] constituent une étude précise de l'orientation des champs par rapport aux gaines ainsi que la forme de celles-ci.

1.5 Conclusion

Bien que le principe général de la SMT soit simple, bien qu'elle soit une technique de stimulation neuronale non invasive et bien que le matériel qu'elle nécessite ne soit pas trop encombrant, son utilisation reste toutefois délicate. En effet, il est difficile pour un personnel soignant de connaître de manière précise la zone du cortex cérébral à stimuler et par voie de conséquence d'en estimer les effets qui sont générés chez un

patient. Aussi, avant de pouvoir envisager une séance de stimulation, il convient de calculer précisément l'orientation et le positionnement des bobines de l'appareillage. Lorsque cela se fait de manière empirique, le personnel soignant dispose la bobine de l'appareillage au-dessus de la zone qu'il souhaite stimuler, en se fondant sur sa connaissance de la cartographie du cortex cérébral et de repères anatomiques. Il procède alors à une stimulation. Après en avoir observé l'effet généré une première fois, il effectue un nouveau positionnement de la bobine et réitère l'opération autant de fois que nécessaire. C'est ainsi, en réalisant un balayage empirique au-dessus de la tête du patient, qu'il détermine l'emplacement qui produit la meilleure réponse au stimulus. On appelle cet emplacement *hot spot*. La bobine peut alors être fixée sur celui-ci, et il est alors possible de démarrer une séance de SMT.

L'objectif de ce travail de doctorat consiste à développer un outil de simulation permettant de mieux appréhender ce travail préparatoire afin de faciliter le placement de la bobine sur le cortex cérébral du patient. La nature prospective des développements entrepris au cours de ce travail n'a pas permis de mettre en place une démarche de conception logicielle linéaire et rigoureuse. En effet, il a fallu privilégier une démarche pragmatique afin d'obtenir un résultat utilisable à moyen, voire à court terme. Il apparaît aujourd'hui que si le logiciel est opérationnel, son évolutivité serait grandement améliorée par une refonte globale. C'est ainsi qu'une partie de ce travail doit être placée dans une démarche de refactoring et de rétro-ingénierie que nous développerons à la fin de ce manuscrit. Cette démarche doit se faire dans un cadre d'ingénierie dirigée par les modèles (IDM) dont il convient de présenter les enjeux et les concepts fondamentaux dans le chapitre suivant.

Chapitre 2

L'ingénierie dirigée par les modèles

Le travail effectué s'est imposé dans un cadre pragmatique, puisqu'il fallut obtenir un outil utilisable à court terme. Bien que le logiciel livré soit opérationnel, son évolutivité serait grandement améliorée par une refonte globale. C'est la raison pour laquelle une partie de ce travail est placée dans une démarche de rétro-ingénierie. Cette démarche doit se faire dans le cadre de l'Ingénierie Dirigée par les Modèles (IDM), dont il convient de présenter les enjeux et les concepts fondamentaux dans ce chapitre.

Dans une première partie, nous présentons l'Ingénierie Dirigée par les Modèles (IDM) afin de définir de manière précise ses concepts fondamentaux. Dans une deuxième partie, nous montrons dans quelle mesure l'approche IDM est au coeur de ce travail, notamment pour assurer le travail de refactorisation (refactoring) nécessaire *in fine* à la migration des composants logiciels patrimoniaux utilisés. Enfin, nous concluons sur les éléments présentés dans ce chapitre.

2.1 Présentation de l'ingénierie dirigée par les modèles

L'Ingénierie Dirigée par les Modèles (IDM ou MDE pour Model Drive Engineering en anglais) est une démarche du génie logiciel qui s'intéresse à la problématique du développement, de la maintenance et de l'évolution d'un logiciel dans un contexte industriel. L'IDM repose sur trois notions fondamentales : le *modèle*, le *méta-modèle*

et la *transformation*. L'IDM se fonde sur le constat que le développement actuel est centré sur le code et que celui-ci est le seul artefact de l'ingénierie qui ait fait l'objet jusqu'à maintenant de la mise en place d'outil plus ou moins *ad hoc* pour sa gestion.

Commençons tout d'abord par donner quelques définitions des concepts relatifs à l'IDM et à la rétro-ingénierie.

2.1.1 Le modèle

Dans l'optique IDM, un modèle est une représentation d'un système (*cf.* Fig 2.1). Le système étant l'entité que l'on souhaite modéliser afin de comprendre, analyser, prédire son comportement. Le but de ce modèle est de pouvoir répondre à des questions à la place du système d'étude.

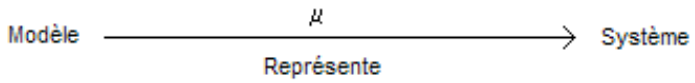

FIGURE 2.1 – Le modèle est une représentation du système modélisé

Ainsi, la relation *est une représentation de* est une des relations clés de l'IDM. Elle relie le système en cours d'étude (SUS : System Under Study) et le modèle qui le représente. Cette relation sera notée par la suite par μ et le lecteur intéressé en retrouvera des défitions voisines et complémentaires dans [Atkinson and Kuhne, 2003], [Bézivin, 2004], [Seidewitz and Technologies, 2003] et [Parastoo Mohagheghi, 2009]. Avec cette seule notion, nous sommes un peu limités car ce que nous venons de définir comme un modèle peut être alors considéré comme un système et être à son tour décrit par un autre modèle. L'utilisation de la notion de modèle n'a de sens que s'il peut être productif, c'est à dire être manipulable par la machine et ainsi améliorer notre connaissance du système. Ainsi, certains auteurs proposent comme définition du modèle :

Définition A model is a description of (part of) a system written in a well-defined language. [Kleppe et al., 2003]

que l'on peut traduire en français par :

Définition Un modèle est une description (d'une partie) d'un système écrite dans un langage bien défini. [Favre et al., 2006].

L'introduction dans cette définition de langage bien défini fait indirectement référence à la deuxième relation fondamentale de l'IDM. Il s'agit de la relation *est conforme à* (notée χ) qui va relier le modèle à son méta-modèle.

2.1.2 Le méta-modèle

Le méta-modèle est une entité qui permet de décrire le modèle. Si l'on replace la relation *est conforme à* dans le domaine de la programmation orientée objet, on reconnaît la relation qui relie une instance d'un objet à sa classe : *un objet est conforme à sa classe*.

Ainsi, l'IDM est centrée autour de la notion de modèle et surtout de méta-modèle et permet de regrouper sous une dénomination commune un ensemble de concepts que l'on retrouve dans bons nombres d'espaces techniques que ce soit dans le domaine :

- des bases de données : un enregistrement d'une base de données relationnelle est conforme au schéma de la base,
- de la programmation orientée objet : une instance d'un objet est conforme à la classe de cet objet,
- en théorie des langages et de la compilation : un code source est conforme à une grammaire,
- des technologies XML : un fichier XML est conforme à sa *DTD*.

La principale force de l'IDM repose sur cette démarche unificatrice ([Bezivin, 2005]).

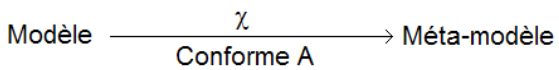

FIGURE 2.2 – Le méta-modèle est une entité qui permet de décrire un modèle.

Lorsqu'un modèle est conforme à son méta-modèle, il peut alors être modifié par un ensemble de transformations de modèle qui permettent de faire passer le modèle

d'un état à un autre.

2.1.3 La transformation

La notion de *transformation* est la troisième notion fondamentale de l'IDM et est sûrement la plus importante mais aussi celle qui ne fait pas encore totalement consensus en terme de définition [Rahim and Mansoor, 2008], [Lano and Clark, 2008], [Iacob et al., 2008]. Le lecteur intéressé trouvera une présentation détaillée des différents types de transformation dans [Favre et al., 2006]. Nous nous limitons ici à ne donner que quelques exemples de transformations dans les différents espaces technologiques :

- script de migration dans les bases de données relationnelles,
- méthodes de classe et d'instance dans les technologies objets,
- compilation, génération de code en théorie des langages,
- transformation XSLT dans l'espace technologique XML.

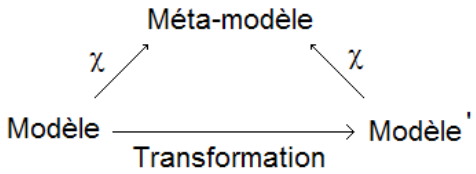

FIGURE 2.3 – Transformation d'un Modèle (en Modèle') en conformité avec son méta-modèle.

Notons que l'exemple de transformation *génération de code* prend un sens particulier dans la mesure où cette transformation fait passer un modèle vers un autre modèle qui *a priori* possède un niveau d'abstraction moins élevé et qui est donc *a priori plus* proche de la machine. Ainsi, le code binaire à partir de code assembleur peut être exécuté par la machine sur laquelle il a été compilé. On obtient alors un modèle productif pour une certaine plateforme d'exécution.

2.1.4 Application de l'IDM dans le contexte de l'étude

L'une des difficultés de ce travail réside dans l'obligation d'intégrer des composants logiciels patrimoniaux (*legacy software components*), qui reposent sur du code

informatique n'intégrant pas la notion de modèle. En fondant le développement à effectuer sur une démarche de rétro-ingénierie, il est possible de proposer un ensemble de modèles, de méta-modèles et de techniques qui permettent de mettre en place une nouvelle implémentation qui améliore sensiblement la modularité et l'évolutivité du produit final obtenu. Grâce à l'IDM, nous allons pouvoir mettre en place cette démarche de refactorisation (refactoring). Nous allons pour cela étendre les concepts de l'IDM au domaine de la rétro-ingénierie et proposer des modèles et des méta-modèles qui reposeront sur le code informatique développé pour les besoins de l'étude. Alors nous serons en mesure d'effectuer une refactorisation (refactoring) du produit logiciel obtenu afin d'en améliorer sa maintenance et son évolution.

2.2 IDM et refactorisation (*refactoring*)

L'activité de modélisation n'est pas vraiment éloignée de celle de la refactorisation (*refactoring*) puisque dans les deux cas, il s'agit de mettre en place un ensemble de méta-modèles qui permettent de décrire un modèle du système que l'on étudie. Dans la pratique, il s'agit de reprendre le code source existant afin d'en faire ressortir les modèles et méta-modèles sous-jacents. Cela peut être aussi l'occasion de faire apparaître les patrons de conception qui ont été mis en place indirectement. Au-delà de notre cas d'étude, la refactorisation est une part importante de l'IDM puisque avec le temps et l'évolution des technologies et l'apparition de nouveaux outils, de *frameworks* (cadriciels), tout système actuel devient, à un moment ou à un autre, obsolète et doit faire l'objet d'une maintenance ou d'une refonte.

A l'heure actuelle, en parlant de code patrimonial (*legacy code*), on pense surtout à des applications développées en COBOL ([McCaracken and Garbassi, 1970]. Pourtant, les applications développées aujourd'hui dans le dernier langage à la mode deviendront un jour, elles aussi désuètes et devront être maintenues ou réécrites. La réécriture sera d'autant plus facile s'il est possible, comme pour le développement dans le cadre de l'IDM de définir à partir du code source existant un modèle indépendant de la plateforme (*Platform Independen Model*, PIM), pour dans un second temps le réimplémenter sur une nouvelle plateforme d'exécution (*Plateform Specific*

Model, PSM).

Mais la rétro-ingénierie est beaucoup plus ancienne que l'IDM. Dans l'article de référence [Chikofsky and JH, 1990], les auteurs y définissent les concepts fondamentaux :

- Ingénierie directe (*Forward engineering*),
- Rétro-ingénierie (*Reverse engineering*),
- Redocumentation,
- Rétro-conception (*Design recovery*),
- Restructuration (*Restructuring*),
- Ré-ingénierie (*Reengineering*).

Nous avons retenu les principaux.

2.2.1 Les outils de rétro-ingénierie

La plupart des outils actuels de rétro-ingénierie ont en commun la volonté d'offrir des vues simplifiées du logiciel afin de faire apparaître les éléments clés du système. Ainsi, la plupart des outils consistent en des analyseurs de code qui vont tenter d'extraire du code source, qui peut être relativement important, des éléments pertinents et synthétiques ([Arnold, 1993]), [Bagci, 2009]. On retiendra par exemple comme vues particulières :

- les vues graphiques (relations entre classes, diagramme de dépendance etc. . .),
- les documentations de premier niveau (liste des fonctions *cf* JDoc, doxygen etc. . .),
- les éléments métriques (nombre de classes, nombre de fichiers, etc. . .).

2.2.2 La programmation globale

Le principal inconvénient de la rétro-ingénierie est dû au fait que souvent le logiciel étudié ne se limite pas à son code source (programmation détaillée), mais à un ensemble d'artefacts qui mettent en évidence tous les aspects du développement d'un logiciel. On trouvera une représentation Fig 2.4 telle que présentée dans [Favre, 1995]

Ainsi, un code source doit être étudié au travers de ses différentes versions. La

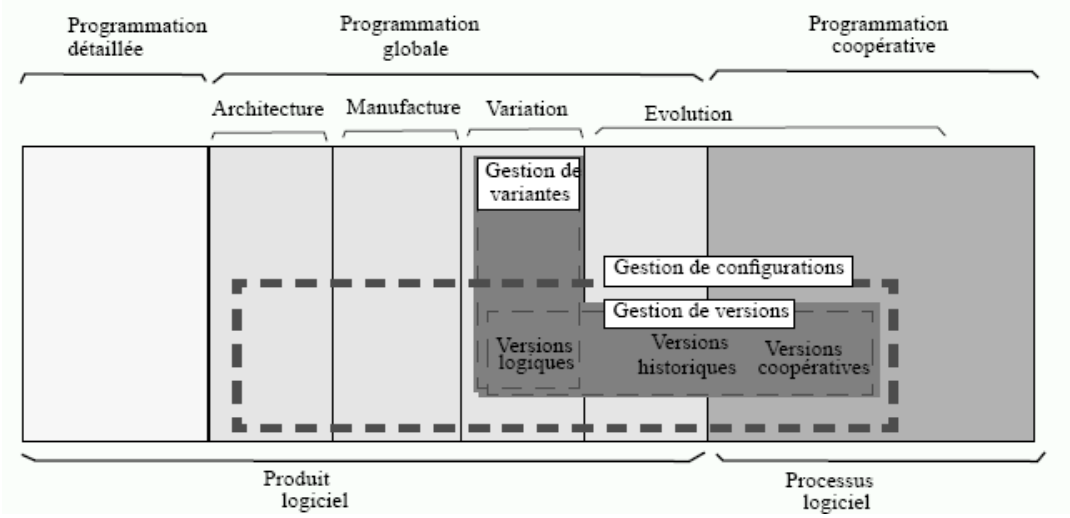

FIGURE 2.4 – La production d'un logiciel ne se limite pas au code source (programmation détaillée), mais doit être vue dans son ensemble : programmation globale et détaillée. Source de l'image : [Favre, 1995]. On en trouvera une version améliorée dans [Favre et al., 2006]

plupart du temps, le passage du code source vers l'application en production se fait au moins via un outil décrivant sa compilation (Makefile, iMake, cMake, Ant ...) quand ce n'est pas par un utilitaire de déploiement. Le code source s'accompagne aussi bien souvent d'un système de gestion de bugs. Tous ces outils sont autant d'informations à prendre en compte.

Dans la suite de ce manuscrit, nous mettrons en évidence comment certains aspects du travail réalisé s'intégraient dans une démarche comparable à celle de l'IDM et nous consacrerons une partie à la description de modèles et de méta-modèles qui pourraient servir de point de départ au refactoring du simulateur.

2.3 Conclusion sur la démarche de rétro-ingénierie adoptée

Le développement entrepris afin de fournir un outil de visualisation 3D pour l'utilisation de la SMT s'effectue avec la contrainte de l'obtention d'un outil exploitable à court terme. Cela implique l'intégration de composants logiciels patrimoniaux, et cela, au détriment de la modularité et de l'évolutivité du produit logiciel obtenu.

Grâce à la démarche de rétro-ingénierie fondée sur les concepts de l'IDM présentés précédemment, nous avons pu limiter ces inconvénients. Nous détaillons ce travail de refactorisation dans le chapitre 6.

Il convient maintenant de présenter les travaux théoriques à partir desquels nous allons modéliser informatiquement le champ électromagnétique que génère la bobine lors d'une séance de SMT. Nous détaillons également la physique du modèle envisagé.

Chapitre 3

Modélisation et calcul des effets de la SMT

Le principal but des travaux menés en modélisation, mathématique et informatique dans ce domaine a été de fournir des indications sur l'influence des champs électromagnétiques générés par la bobine sur le cortex du patient et surtout essayer de prédire le lieu où la stimulation serait la plus importante.

Comme exposé dans la section 1.3 page 30, la première loi à considérer est celle de Biot & Savart, qui permet de calculer le champ magnétique généré par la bobine. Les premiers travaux dans ce domaine ont donc eu pour but d'étudier la forme de ce champ magnétique en fonction de la forme de la bobine. C'est ainsi que de nombreux auteurs ont proposé des cartographies du champ magnétique généré par une bobine simple ou une bobine double. Ainsi Ueno *et al.* [Ueno et al., 1988] proposent de calculer le champ magnétique sur un maillage quasi-sphérique composé de 58 plans, tandis que Grandori *et al.* [Grandori and Ravazzani, 1991] effectuent une intégration numérique de la loi de Biot et Savart pour une bobine simple de forme circulaire. En utilisant les lois de Maxwell-Faraday et la loi d'Ohm $\mathbf{J} = \sigma \cdot \mathbf{E}$ dans un milieu isotrope et linéaire, Grandori *et al.* déduisent alors la densité de courant.

En utilisant le principe d'additivité des champs magnétiques, les auteurs proposent aussi des cartographies de champs pour des bobines doubles voire quadruples.

3.1 Calcul des effets de la stimulation

3.1.1 Modèle sphérique

Fondées sur les travaux de [Sarvas, 1987], les questions relatives au problème inverse (détermination des sources des champs électromagnétiques dans la tête du patient à partir de leurs mesures effectuées sur le crâne du patient) ont déjà été mises en avant par des études théoriques sur l'EEG dont la SMT se rapproche considérablement du point de vue du calcul électromagnétique. Ainsi, de nombreux auteurs ont proposé de s'intéresser comme pour l'EEG au calcul des champs électromagnétiques en assimilant la tête du patient à un modèle de sphères concentriques. Ces sphères pouvant présenter des propriétés électromagnétiques différentes, il s'agit d'étudier la diffusion des champs dans ce modèle simplifié de tête humaine.

Le principe de base du modèle sphérique présenté dans [Heller and van Hulsteyn, 1992] est fondé sur le principe de réciprocité : la densité de courant générée par un dipôle magnétique à une position r_1 dans un milieu conducteur peut se déduire du calcul de la densité de courant générée par une boucle de courant localisée à l'extérieur du milieu.

Ce principe est d'ailleurs aussi valable pour d'autres milieux qui ne sont pas forcément de forme sphérique. Cette modélisation est aussi utilisée dans [Ravazzani et al., 1996] qui s'intéressent au calcul des champs dans un médium de forme cylindrique (typiquement un membre quand il s'agit d'étudier la stimulation magnétique des nerfs périphériques).

Par contre là où le modèle sphérique est particulièrement intéressant, c'est que les auteurs de [Heller and van Hulsteyn, 1992] montrent que la valeur de $\overrightarrow{E}$ dans le milieu est indépendante du nombre de couches et des valeurs des conductivités dans celles-ci. Ils démontrent cette propriété en étudiant des conditions aux limites au niveau des interfaces des différentes couches sphériques comme par exemple la continuité des composantes radiales.

Avec de telles hypothèses, le calcul de $\overrightarrow{E}$, qui peut se déduire de celui de $\overrightarrow{B}$ via les équations de Maxwell (Maxwell-Faraday notamment), reste indépendant des variations de conductivité. Ainsi, même si cela ne justifie en rien le fait de pouvoir

négliger complètement ces variations dans le cas général, le fait que le modèle sphérique soit proche de la géométrie du crâne peut donner envie d'appliquer directement le champ sur la surface du cortex. En effet celle-ci, contrairement à la peau et au crâne, est la première surface non vraiment assimilable à une sphère dans un modèle de tête plus proche de la réalité physique. On peut alors espérer que la diffusion des champs dans ces deux premières couches soit indépendante des conductivités de ces dernières et effectuer les calculs comme si elles se comportaient comme de l'air.

Au-delà de ces simplifications, il faut prendre en compte le fait que les champs vont diffuser dans un milieu non homogène où la perméabilité magnétique change en fonction du milieu (typiquement les valeurs de perméabilité varient entre les principales classes de tissus d'un cerveau : matière blanche, matière grise, liquide céphalo-rachidien [Hédou, 1997]). Le calcul des champs dans les différents milieux devient très complexe et nécessite des outils de résolutions mathématiques et informatiques plus évolués que le modèle sphérique.

3.1.2 Méthode des éléments finis

3.1.2.1 Principe

La méthode la plus répandue est celle des éléments finis, qui consiste à définir un domaine discret (maillage) pour représenter l'ensemble du domaine sur lequel on souhaite effectuer les calculs. Une fois le maillage défini (la difficulté principale étant de réaliser un maillage qui respecte certaines conditions de régularité), il faut définir sur chacun des éléments de ce maillage une formulation locale des lois qui régissent le système macroscopique. Une fois ces équations posées pour chaque élément, celles-ci se ramènent informatiquement à la mise en place d'un grand système linéaire dans lequel les coefficients dépendent des propriétés physiques locales (ici conductivité, perméabilité). La résolution de ce système linéaire permet alors de calculer numériquement une valeur du champ recherché en chacun des éléments du maillage. Par approximation et interpolation, il est possible d'associer à n'importe quel point de l'espace une valeur du champ recherché. Les équations de Maxwell (*cf.* section 1.3 page 29) permettent alors de remonter aux autres champs électromagnétiques en fonction de celui qui a été calculé dans la formulation.

3.1.2.2 Utilisation pour la SMT

Krasteva *et al.* [Krasteva et al., 2002] étudient la diffusion du potentiel vecteur magnétique dans un milieu non homogène quelconque. En utilisant Biot & Savart, les auteurs calculent le champ potentiel magnétique généré par une bobine sur la surface englobant l'espace d'étude (qui peut présenter des zones d'inhomogénéité en terme de conductivité électrique, perméabilité magnétique). A partir de ceci, en travaillant sur les équations de Maxwell, et en proposant des hypothèses simplificatrices (quasi-statique, régime oscillant), les auteurs ramènent le système d'équations à la diffusion du champ magnétique (équation de Poisson vectorielle) qu'ils résolvent en utilisant une méthode d'éléments finis.

Dans [Miranda et al., 2003], les auteurs s'affranchissent des hypothèses simplificatrices évoquées dans le paragraphe précédent en introduisant des équations supplémentaires qui expriment les relations de continuité de champs aux interfaces entre deux milieux de conductivités différentes. Ils résolvent ensuite le système sur un maillage adaptatif (en fonction de la proximité de la bobine génératrice). Leur étude comparative entre un milieu avec des propriétés électromagnétiques constantes et non constantes tend à montrer que le champ électromagnétique dans le milieu sera plus fort au niveau des interfaces entre deux milieux.

3.1.3 Méthode d'impédance 3D

Dans [Nadeem et al., 2003] et [Persson et al., 2003], les auteurs calculent les champs électriques et magnétiques induits en utilisant une méthode de calcul d'impédance tridimensionnelle. Pour ce faire, ils utilisent les données issues du *The Visible Human Project* [Ackerman, 1998] qui fournit un volume IRM segmenté sur l'ensemble d'un être humain. Ainsi en se limitant à la tête du patient et en utilisant un maillage volumique uniforme (grille 3D calquée sur la géométrie de l'IRM), les auteurs mettent en place un réseau de résistances (cf Fig 3.1).

L'étape suivante consiste, en utilisant les lois de Kirchoff, à écrire que la somme des courants circulant entre deux noeuds du maillage est égale au courant induit localement par le champ magnétique généré par la bobine (cf Fig 3.2).

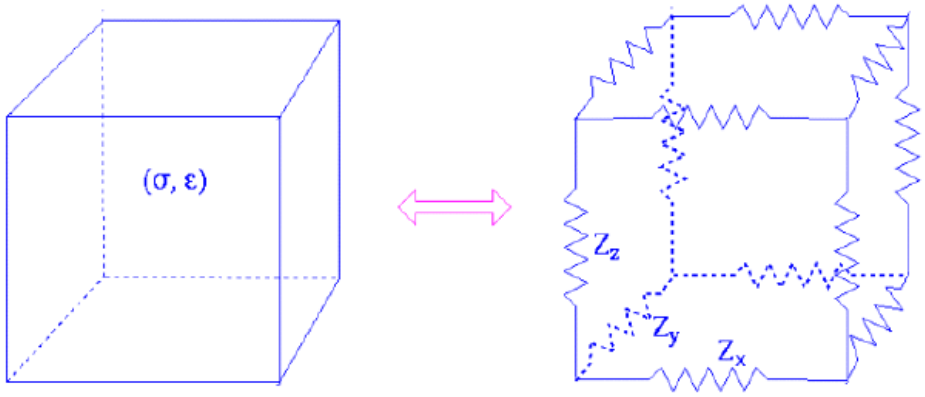

FIGURE 3.1 – Un voxel de propriétés électromagnétiques données est modélisé par un réseau de résistances. Source de l'image : [Persson et al., 2003]

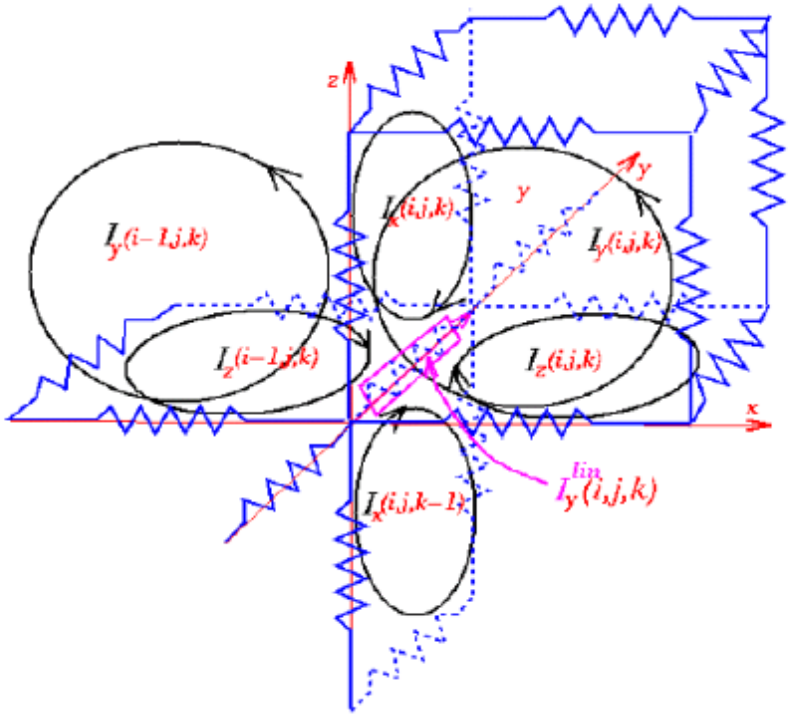

FIGURE 3.2 – Boucle de courant et loi de Kirchoff : les 4 arêtes d'une face de voxel définissent une boucle de courant sur laquelle on exprime la loi de Kirchoff. Source de l'image : [Persson et al., 2003]

Le calcul du champ magnétique est exposé dans [Nadeem et al., 2003] où les auteurs effectuent celui-ci en utilisant la loi de Biot & Savart qu'ils intègrent en

réalisant une discrétisation de bobine MC-B70 (Medtronic).

L'ensemble des équations exprimées sur l'ensemble des noeuds permet de définir un schéma numérique (cf équation 3.1) que les auteurs résolvent itérativement via une méthode de relaxation successive (*successive over-relaxation*).

$$\begin{aligned}
&[I_x(i,j,k-1) - I_x(i,j,k) - I_z(i-1,j,k) + I_z(i,j,k)]Z_y(i,j,k) \\
&+[I_x(i,j+1,k) - I_x(i,j,k) + I_y(i-1,j+1,k) - I_y(i,j+1,k)]Z_z(i,j+1,k) \\
&+[I_x(i,j,k+1) - I_x(i,j,k) + I_z(i-1,j,k+1) - I_z(i,j,k+1)]Z_x(i,j,k+1) \\
&+[I_x(i,j-1,k) - I_x(i,j,k) - I_y(i-1,j,k) + I_y(i,j,k)]Z_z(i,j,k) = -i\omega\Delta y\Delta z B.\hat{x} \\
&[I_y(i,j,k-1) - I_y(i,j,k) - I_z(i,j-1,k) + I_z(i,j,k)]Z_x(i,j,k) \\
&+[I_y(i+1,j,k) - I_y(i,j,k) + I_x(i+1,j-1,k) - I_x(i+1,j,k)]Z_z(i+1,j,k) \\
&+[I_y(i,j,k+1) - I_y(i,j,k) + I_z(i,j-1,k+1) - I_z(i,j,k+1)]Z_x(i,j,k+1) \\
&+[I_y(i-1,j,k) - I_y(i,j,k) - I_x(i,j-1,k) + I_x(i,j,k)]Z_z(i,j,k) = -i\omega\Delta z\Delta x B.\hat{y} \\
&[I_z(i,j,k) - I_z(i,j-1,k) + I_y(i,j,k-1) - I_y(i,j,k)]Z_y(i,j,k) \\
&+[I_z(i,j,k) - I_z(i+1,j,k) + I_x(i+1,j,k) - I_x(i+1,j,k-1)]Z_z(i+1,j,k) \\
&+[I_z(i,j,k) - I_z(i,j+1,k) + I_y(i,j+1,k) - I_y(i,j+1,k-1)]Z_x(i,j+1,k) \\
&+[I_z(i,j,k) - I_z(i-1,j,k) + I_x(i,j,k-1) - I_x(i,j,k)]Z_z(i,j,k) = -i\omega\Delta y\Delta x B.\hat{z}
\end{aligned} \tag{3.1}$$

Les auteurs en déduisent alors les courants induits sur chacun des éléments du maillage et peuvent ainsi remonter à la densité de courant J et au champ électromagnétique induit en utilisant la loi d'Ohm.

3.2 Aide à la conduite de séance de SMT

Outre les nombreux travaux théoriques qui permettent de faire des calculs dans l'absolu, certaines études ont été menées pour permettre de faire ces calculs et apporter une aide au placement de la bobine pendant la séance de stimulation.

Pour être capable de repérer la position de la bobine et celle de la tête du patient dans un repère commun il existe deux grandes méthodes : les appareils de fixation

et le recalage par dispositif optique. On peut leur adjoindre un système de neuro-navigation.

3.2.1 Appareil de fixation

Il s'agit d'utiliser un cadre, proche de ceux utilisés en imagerie stéréotaxique. Un cadre est un dispositif rigide fixé sur la tête du patient, et la bobine est fixée au cadre. Connaissant la position de la bobine par rapport au cadre, on en déduit la position de la bobine par rapport à la tête du patient. En connaissant les dimensions et orientations du cadre ou d'un bras pouvant être robotisé [Lancaster et al., 2004], il est possible de remonter à la matrice de transformation qui permet de passer d'un repère à un autre.

3.2.2 Recalage par dispositif optique

L'enregistrement de la position de la bobine par rapport à la tête du patient se fait via des marques, posées sur la bobine et le patient, qui sont repérées par un système optique [Noirhomme et al., 2004]. Le patient muni de ces mêmes marqueurs passe alors une IRM sur laquelle les marqueurs sont aussi enregistrés. Ceci permet ensuite de recaler la position de la bobine par rapport aux images morphologiques IRM. Comme précédemment, il est alors possible de remonter à la position de la bobine par rapport à la tête du patient.

3.2.3 Neuro-navigation

La plupart des outils issus de ces recherches proposent une visualisation des images IRM du patient [Krings et al., 2001], voire des modèles des cerveaux de ces patients (généralement une reconstruction de la surface du cortex) sur lesquelles va être superposée la bobine de la stimulation. Ainsi cela apporte une aide à la détermination de la zone du cerveau qui va être stimulée (placée par la plupart des logiciels directement sous la bobine). Il est aussi possible de trouver des logiciels de réalité augmentée présentant différentes coupes de la tête du patient, voire une reconstruction de son cortex et les effets de la stimulation magnétique représentés sur celle-ci([Noirhomme et al., 2002]).

3.3 Modélisation des effets de la stimulation sur les neurones

Cette partie s'intéresse plus particulièrement à l'effet que peut avoir la stimulation magnétique transcrânienne sur les fibres nerveuses. En effet les neurones regroupés en gaines de myéline dans la matière blanche peuvent être modélisés par des *câbles électriques*. Il est alors possible de mettre en application certaines des équations de l'électronique et notamment l'équation de câble [Roth and Basser, 1990]

Les neurones peuvent alors être vus comme une succession de circuits électriques reliés les uns aux autres. Kamitani ([Kamitani, 2001]) propose même de modéliser, outre les neurones, les synapses et autres interconnexions. En utilisant des logiciels de modélisation neuronale : Neuron ([Hines, 1993]) ou Genesis ([Wilson et al., 1989]) il est même possible de mettre en place des réseaux de neurones (au sens physique du terme) et d'étudier l'effet (et sa propagation) d'une impulsion de SMT sur le réseau ainsi créé. On notera l'importance du paramétrage de tels réseaux en terme de paramètres électromagnétiques ainsi que de leur comportement. En effet, au-delà de l'équation de câble exposée plus haut qui permet de modéliser un réseau de neurones passifs, il est possible d'utiliser des réseaux actifs. Il s'agit par exemple du modèle de Hodgkin-Huxley [Hodgkin and Huxley, 1952]. La principale différence réside dans l'introduction, au sein des équations du modèle, d'un terme exprimant qu'un potentiel interne au câble va venir s'additionner ou se soustraire au simple potentiel électrique auquel le câble est soumis.

3.4 Validation

En utilisant des techniques d'imagerie médicale et/ou de mesure de l'activité cérébrale, il est possible d'observer certains des effets de la SMT.

3.4.1 Confrontation des résultats à des techniques permettant de mesurer l'activité cérébrale

L'utilisation de la SMT conjointement à des techniques d'imagerie qui mesurent l'activité cérébrale, permet de vérifier l'influence de la stimulation sur celle-ci. Ainsi

les effets mesurables par EEG/MEG de la SMT sur une zone du cerveau contrôlant une certaine tâche motrice sont très comparables aux signaux cérébraux relevés sur un patient qui effectuerait volontairement ladite tâche.

Bestmann *et al.* proposent [Bestmann et al., 2004] d'utiliser en plus de l'EEG et de la MEG, l'IRM fonctionnelle pour étudier l'effet immédiat de la stimulation. Une analyse des possibilités et limitations de l'utilisation conjointe de toutes ces techniques est donnée dans [Sack and Linden, 2003]. C'est aussi le cas d'une étude qui porte sur l'observation des changements de l'activité cérébrale mesurée par IRM fonctionnelle tout en effectuant une stimulation via SMT ([Bohning et al., 1999]). Les auteurs y étudient s'il y a une corrélation entre la variation de l'intensité de la stimulation et l'intensité du signal mesurée par l'IRM fonctionnelle.

3.4.2 Mesure des effets de la SMT par l'IRM

L'idée soutenant cette possibilité de mesurer le champ magnétique généré par la bobine de SMT s'appuie sur le fait que la forme du champ magnétique généré par une impulsion électrique dans la bobine est la même que celle d'un champ constant généré par cette même bobine. Ainsi en modifiant l'appareil de stimulation et en créant un champ magnétique permanent en faisant passer dans la bobine un courant électrique continu (et d'intensité beaucoup moindre que celle utilisée pour la stimulation), il va être possible de le mesurer via une acquisition IRM. En effet le champ magnétique généré par la bobine va venir modifier l'imagerie de phase acquise à l'IRM et c'est cette perturbation qui permettra de remonter à la valeur du champ [Bohning et al., 1997].

3.4.3 Comparaison des effets de la stimulation

Dans [Brasil-Neto et al., 1992], les auteurs réalisent une étude comparative des effets de la SMT en fonction de différents paramètres :

- la forme de la bobine : les expérimentations sont menées avec trois bobines en forme de 8 dont l'enroulement varie (nombre de tours, pas de l'enroulement des rayons).
- l'orientation de la bobine par rapport à la tête du patient : la position de la

bobine ne change pas, différentes rotations de la bobine sont effectuées autour de l'axe normal à son plan passant au milieu du 8.

– la forme et l'intensité de l'impulsion utilisée.

Leur protocole consiste à réaliser une mesure du Potentiel Evoqué Moteur (PEM) en utilisant un électromyographe, puis à comparer l'importance de celui-ci en fonction des paramètres d'entrée décrits plus haut. Si les résultats obtenus par les auteurs sur 10 personnes confirment les conclusions de [Day et al., 1990], ils montrent aussi clairement l'importance de la forme de l'impulsion utilisée. En effet, Day *et al.* ainsi que d'autres auteurs insistent sur l'importance de l'orientation du champ électrique induit par le champ magnétique. Ainsi les neurones les plus sujets à l'excitation sont ceux qui sont alignés avec le champ. Anatomiquement parlant, les neurones qui vérifient le plus cette propriété sont les fibres horizontales, perpendiculaires au sillon central ([Marin-Padilla, 1969]). D'autre part, cette étude met en avant le fait qu'à faible intensité, les comportements en terme d'orientation d'une bobine bi-phasique (l'impulsion est une sinusoïde *cf.* Fig 1.8) et d'une bobine mono-phasique (l'impulsion est une demi sinusoïde) sont les mêmes, mais qu'il est beaucoup plus délicat d'effectuer une étude sur la meilleure orientation avec une bobine bi-phasique utilisée à haute intensité.

3.5 Discussion

Comment comparer les différentes méthodes présentées jusqu'à maintenant ? Un premier élément de réponse est la façon même dont elles ont été présentées : par ordre chronologique et augmentation du degré de complexité.

La seule utilisation d'une loi comme Biot & Savart (qui plus est, dans le vide) ne permet pas d'apprécier la diffusion du champ magnétique dans la matière. Mais cette seule formule présente l'avantage d'être relativement simple à mettre en place et donc peut servir de référence et être utilisée pour des calculs plus complexes. C'est ainsi que [Nadeem et al., 2003] utilisent le champ magnétique calculé par Biot & Savart pour déterminer le membre de droite de son système d'équations obtenu par la méthode d'impédance. Mais on peut se demander si l'approximation

faite ici n'est pas trop forte dans la mesure où la valeur du champ magnétique en tout point du maillage (correspondant à des tissus de conductivités différentes) est considérée comme étant égale à celle du vide. Quel est l'apport de l'utilisation de modélisations plus complexes (incluant tout ou partie des équations de Maxwell comme proposé dans [Miranda et al., 2003], [Krasteva et al., 2002]) en terme de précision dans les calculs ? Au-delà de la précision il s'agit aussi de la possibilité d'observer des phénomènes prenant en compte les hétérogénéités et les orientations des gaines nerveuses.

En revanche, l'introduction de ces équations est à la fois difficile et coûteuse en temps de calculs. Outre la difficulté pour les non spécialistes d'appréhender les équations, la mise en place des systèmes numériques est elle-même un problème. Ainsi pendant cette thèse, il n'a pas été possible, à partir des seuls articles évoqués plus haut, de refaire ce que les auteurs avaient implémenté pour comparer numériquement leurs résultats avec les nôtres.

Les méthodes diffèrent principalement sur les équations mises en jeu et sur les hypothèses effectuées : quasi-statique, régime harmonique, etc. Ainsi, les équations peuvent être plus ou moins simplifiées, et aboutir à la mise en place d'un système numérique plus léger. Il n'en reste pas moins que si certains auteurs préfèrent calculer les courants induits alors que d'autres préfèrent s'intéresser aux calculs directs du champ électromagnétique, les équations de Maxwell permettent dans tous les cas d'obtenir n'importe quelle grandeur électromagnétique à partir de celle qui a été calculée.

3.6 Conclusion

Dans ce chapitre, nous avons étudié les techniques de Stimulation Magnétique Transcrânienne (SMT). Nous avons recensé les différentes méthodes mathématiques et numériques qui ont été proposées dans la littérature pour calculer les effets de la SMT. Le modèle sphérique, la méthode des éléments finis, la méthode d'impédance 3D ont été détaillés et nous avons ainsi pu constater les différences et les points communs dans les différentes approches.

Nous avons également abordé dans ce chapitre la conduite d'une séance de SMT. En effet, il est important pour nos applications de sortir du cadre théorique et de se placer dans les cas concrets où la position de la bobine et celle de la tête du patient sont des facteurs essentiels. Les méthodes proposées utilisent plus particulièrement : (i) un appareil de fixation (dispositif rigide), (ii) une méthode de recalage optique grâce à des diodes (des marqueurs optiques sont placés sur la tête du patient et sur l'appareil de stimulation), (iii) la méthode de neuro-navigation complète ces deux approches en proposant une modélisation du cortex du patient (surface corticale).

En ce qui concerne la modélisation des effets de la stimulation sur les neurones, nous avons constaté dans la littérature que les neurones sont modélisés par un câble électrique passif ou actif. Puis, nous avons présenté les différentes techniques qui cherchent à mesurer les effets de la stimulation. Nous trouvons deux approches principales, soit une mesure de la SMT par EEG/MEG, soit une mesure par IRM fonctionnelle. Les effets de la stimulation sont différents suivant les formes de bobines et les protocoles utilisés.

Devant la complexité des méthodes présentées et la difficulté de leur implémentation, il a été choisi de mettre en place une méthode simple basée principalement sur la loi de Biot & Savart. Elle se rapproche ainsi de celle de [Nadeem et al., 2003] dont elle ne diffère que par la méthode d'intégration numérique. Au contraire de Nadeem *et al.* qui se limitent à la seule Bobine MC-B70, l'outil que nous proposons est totalement générique grâce à l'utilisation de contours paramétriques. Ainsi, dans le chapitre suivant, nous présentons tout d'abord les équations électromagnétiques que nous avons retenues dans notre modélisation. Puis nous présentons l'implémentation informatique de celles-ci avec la mise en place d'un moteur de calcul pour simulateur de SMT.

Chapitre 4

Le noyau de calcul du simulateur

Nous allons dans ce chapitre reprendre dans un premier temps l'équation de Biot et Savart que nous avions introduite au début de notre manuscrit. Nous avons noté que la forme des bobines pouvait jouer un rôle significatif dans le calcul du champ. Nous nous attachons donc dans une première partie à présenter les différentes formes de bobines. Ensuite nous étudierons les possibilités de modélisation de ces bobines par des équations paramétriques. Nous aborderons aussi les techniques d'intégration numériques employées ainsi que leur implémentation avant de présenter les résultats préliminaires que nous avons obtenus.

4.1 De la forme des bobines

Différents agencements de bobines ont été mis en place par les constructeurs d'appareils médicaux, afin d'obtenir différentes formes de champ. Cette forme peut se déduire de la formule de Biot & Savart (4.1) :

$$\vec{B}(\vec{r},t) = \frac{\mu_0}{4\pi} I(t) \oint_C \frac{d\vec{l} \times (\vec{r} - \vec{r'})}{|\vec{r} - \vec{r'}|^3} \tag{4.1}$$

Effectuons quelques approximations et concentrons-nous sur cette équation. D'après la littérature, les perméabilités magnétiques de la peau, du crâne, de la dure-mère et du liquide céphalo-rachidien (LCR) sont très semblables à celle du vide [Krasteva et al., 2002]. En première approximation nous pourrons donc effectuer cette première hypothèse simplificatrice qui tend à considérer que le comportement magnétique de la tête hors cortex est le même que celui du vide. D'autre part, nous

sortirons aussi l'étude de son contexte temporel pour nous focaliser sur la forme du champ magnétique. En effet, la seule influence de $I(t)$ dans l'équation précédente est la modification du module du champ magnétique. Nous nous intéressons à l'instant où ce module est maximal, au temps t_m tel que $I(t_m) = I_{max}$.

Dans ces conditions l'équation se ramène à :

$$\vec{B}(\vec{r}) = \frac{\mu_0 I_{max}}{4\pi} \oint_C \frac{d\vec{l} \times (\vec{r} - \vec{r'})}{|\vec{r} - \vec{r'}|^3}$$

Globalement la forme du champ magnétique dépend principalement de la forme de l'enroulement de la bobine C. Nous allons donc détailler les différentes formes que peuvent prendre les bobines [Jalinous, 1998].

Dans la suite nous préférerons nous intéresser au champ potentiel magnétique $\overrightarrow{A}$ qui est défini par $\overrightarrow{B} = \overrightarrow{rot}\ \overrightarrow{A}$

4.1.1 Bobine simple

On peut modéliser une bobine simple par un cercle de cuivre dans lequel circule le courant. Dès lors, la forme des lignes du champ potentiel magnétique est elle aussi circulaire. Le champ électrique induit par le champ magnétique en l'absence de potentiel électrique est donné par : $\overrightarrow{E} = -\frac{\partial \overrightarrow{A}}{\partial t}$. La forme géométrique de $\overrightarrow{E}$ est donc la même que celle de $\vec{A}$, même si les variations dans le temps sont différentes. La figure 4.1 représente le module du champ potentiel magnétique calculé dans un plan parallèle à celui de la bobine. Le champ présente des iso-contours circulaires d'autant plus intenses que leur rayon est proche de celui de la bobine.

4.1.2 Bobine en forme de 8

Si l'on place deux bobines simples l'une à côté de l'autre et que l'on y fait passer le courant dans deux sens différents, alors par le principe de superposition, le champ potentiel magnétique va être d'autant plus fort à l'intersection des deux bobines. C'est pour cette raison que cette bobine a été mise en place afin d'obtenir une meilleure focalisation des champs générés. Ainsi, en utilisant le même type de représentation que celle du paragraphe précédent, on obtient la Fig 4.2

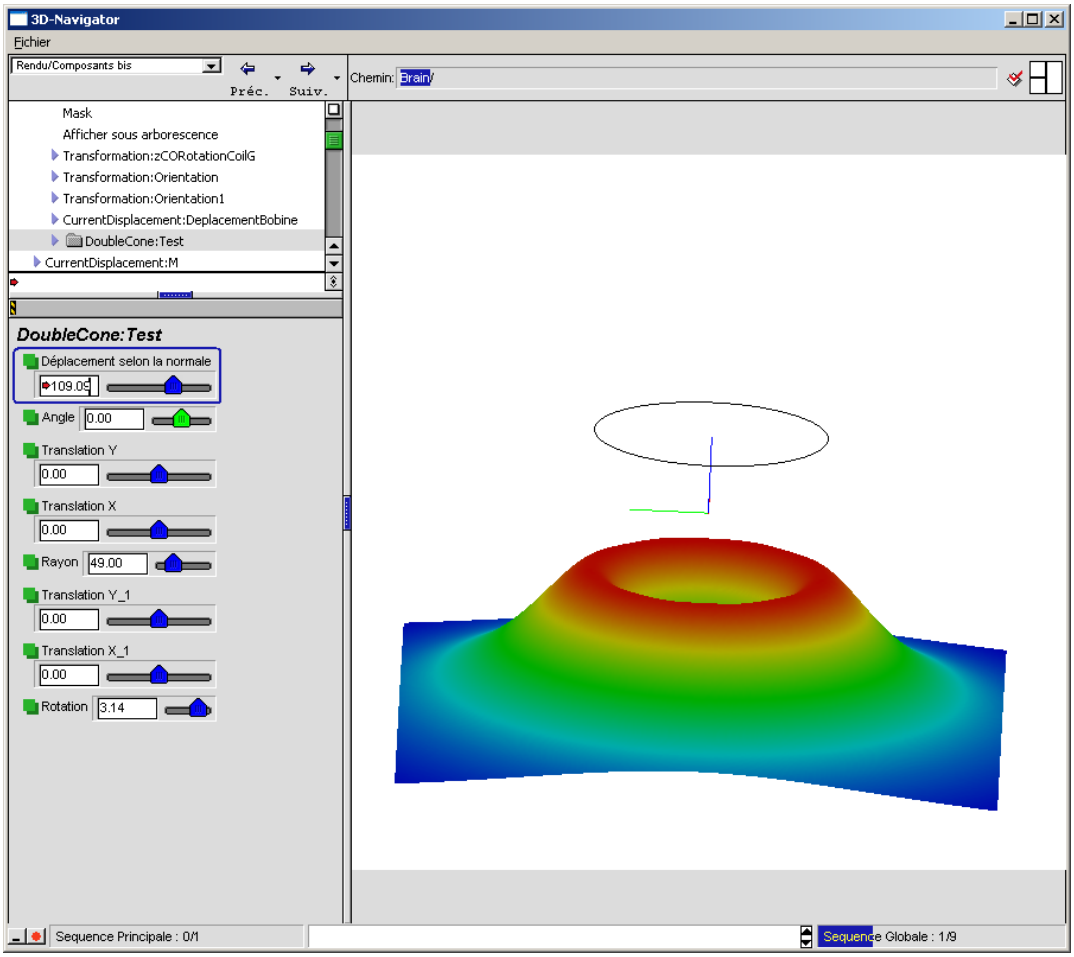

FIGURE 4.1 – Module du champ magnétique généré par une bobine simple. Le champ présente des iso-contours circulaires d'autant plus intenses que leur rayon est proche de celui de la bobine. L'intensité est représentée par la hauteur du relief ; les couleurs ne sont qu'un indice visuel supplémentaire.

Si de tels schémas permettent de mieux appréhender la forme du champ et la focalisation de la bobine en forme de huit, il ne faut pas oublier que :

- pour être complètement rigoureux, il faudrait préciser dans quel plan a été réalisé ce schéma. En effet, à cause de la décroissance rapide des champs, la focalisation de la bobine en forme de 8 disparaît très vite avec l'éloignement. Les distances physiques prennent alors toute leur importance.
- une normalisation des valeurs est effectuée par rapport aux valeurs calculées dans le plan. Il s'agit donc bien d'un maximum relatif au plan de représentation et non d'une normalisation par rapport au maximum des valeurs obtenues sur

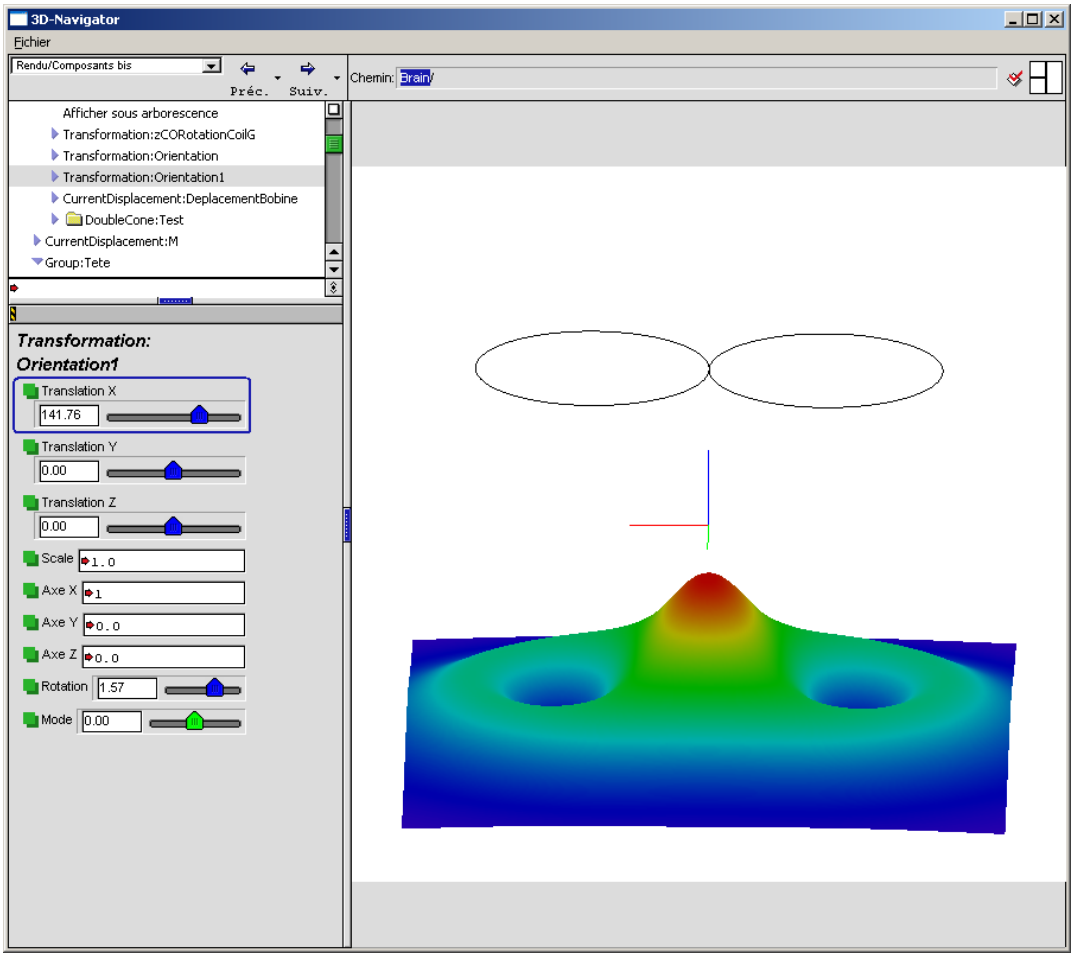

FIGURE 4.2 – Module du champ magnétique généré par une bobine double. Le champ présente une focalisation juste en dessous de l'intersection des deux ailes de la bobine. L'intensité est représentée par la hauteur du relief ; les couleurs ne sont qu'un indice visuel supplémentaire.

un espace tridimensionnel.

Le problème dans ce cas est de déterminer l'espace de calcul le plus pertinent, sachant que le champ sera toujours plus intense à proximité des bobines et que celui-ci devient même infini/indéfini au niveau de la bobine elle-même. Une approche qui pourrait être mise en place est l'utilisation de la coque de la bobine (qui contient la plupart du temps le système de refroidissement) comme référence. En effet, il est inutile de calculer le champ à l'intérieur de la coque puisque celle-ci ne sera jamais dans la tête du patient. Le champ maximal accessible se trouve donc sur la surface de la coque, et peut être utilisé quand on souhaite normaliser le champ à

l'extérieur de la bobine. Cette notion sera d'autant plus importante dans le cadre du développement d'un système voulant étudier l'influence de l'éloignement de la bobine avec la tête du patient.

Un autre point sans doute critique à ce degré de modélisation, est que les bobines physiques que l'on modélise par de simples cercles (ou par extension un ensemble de cercles pour modéliser les différents tours effectués par l'enroulement) sont en fait des spirales dont l'étalement dans le plan de la bobine est sans doute à considérer.

4.1.3 Bobine en forme de double cône

Les bobines en forme de double cône sont fortement semblables aux bobines en forme de 8, mais un angle entre les deux ailes de la bobine permet de mieux épouser la forme de la tête du patient. Pour respecter encore mieux cette forme, chaque bobine peut en outre être non planaire. Compte tenu des spécificités physiques de ces bobines, il devient difficile de faire des généralités sur la forme du champ généré. Il serait d'ailleurs sûrement plus intéressant de représenter non plus celui-ci dans un plan mais sur une surface sphérique (approximation la plus simple d'une tête).

4.1.4 Autres formes de bobines

D'autres formes un peu particulières ont notamment été proposées dans quelques cadres de recherche comme la *Hersed coil* [Roth et al., 2002] et [Zangen et al., 2005]. Il s'agit de bobines dont la forme est relativement différente de celles évoquées jusqu'à maintenant. En effet, elles ont pour but d'effectuer des stimulations dans des ordres de profondeur beaucoup plus importants que ceux permis par les autres bobines (stimulation profonde).

4.2 Modélisation par équation paramétrique

Comme nous l'avons vu en section 4.1 page 55, le calcul du champ magnétique généré par une bobine dans le vide se ramène à une intégration le long d'un contour. L'utilisation du champ potentiel magnétique permet de simplifier les écritures. Avec les mêmes notations que précédemment nous obtenons alors l'équation :

$$\vec{A}(\vec{r}) = \frac{\mu_0}{4\pi} I_{max} \oint_C \frac{d\vec{l}}{|\vec{r} - \vec{r'}|}$$

Comme pour le calcul du champ magnétique, la forme du champ potentiel magnétique est principalement dépendante du contour de la bobine. Nous avons montré dans [Luquet et al., 2005c] que ce calcul peut être fait de façon assez générique, en utilisant une description paramétrique de la bobine. En écrivant le contour comme une équation paramétrique tridimensionnelle

$$\vec{r'} = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}$$

et en décomposant l'équation précédente selon les trois composantes A_x, A_y *et* A_z on obtient :

$$A_x \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} = \frac{\mu_0}{4\pi} I_{max} \int_a^b \frac{dx(t)dt}{\sqrt{(x_0 - x(t))^2 + (y_0 - y(t))^2 + (z_0 - z(t))^2}}$$

$$A_y \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} = \frac{\mu_0}{4\pi} I_{max} \int_a^b \frac{dy(t)dt}{\sqrt{(x_0 - x(t))^2 + (y_0 - y(t))^2 + (z_0 - z(t))^2}}$$

$$A_z \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} = \frac{\mu_0}{4\pi} I_{max} \int_a^b \frac{dz(t)dt}{\sqrt{(x_0 - x(t))^2 + (y_0 - y(t))^2 + (z_0 - z(t))^2}}$$

Le calcul de chacune des trois composantes est alors relativement similaire et peut être effectué en calculant numériquement la forme intégrale en chacun des points où l'on veut connaître le champ.

4.2.1 Intégration Numérique

Quand il est impossible de déterminer analytiquement la primitive d'une équation, ou qu'il n'est pas possible de ramener le calcul d'une intégrale à une autre

dont on connaît une solution, la façon d'obtenir un résultat est de procéder à une intégration numérique. Le principe est relativement simple : il s'agit d'approximer l'intégrale par le calcul de l'aire *de la région du plan délimitée par la courbe représentative de la fonction à intégrer, l'axe des abscisses, et les droites d'équations x=a et x=b* où a et b sont bornes d'intégrations.

Pour ce faire, différentes méthodes mathématiques ont été proposées :

- la méthode des rectangles : l'aire est approximée par la somme des aires d'un ensemble de rectangles
- la méthode des trapèzes : l'aire est approximée par la somme des aires d'un ensemble de trapèzes
- la méthode de Simpson : le calcul se fait en approximant la fonction à intégrer par des polynômes de degré 2 sur un découpage de l'intervalle d'intégration
- les méthodes de Newton-Cotes et de Runge-Kutta, dont on trouvera les définitions mathématiques rigoureuses par exemple dans [Schwartz, 1993]

Ces différentes méthodes se caractérisent principalement par leur qualité d'approximation. En effet, il est possible de montrer mathématiquement que le calcul de l'intégrale peut être approché à n'importe quelle précision en jouant simplement sur le découpage du domaine à intégrer. La qualité de la méthode peut alors s'évaluer en fonction de l'erreur commise pour un découpage donné.

Pour mettre en place informatiquement ce genre de calcul, le Numerical Recipes [Press, 1992] propose l'implémentation en pseudo-code C de quelques-unes des méthodes d'intégrations sus-citées. Nous nous sommes ici appuyés sur une sur-couche du Numerical Recipes développée par Soluscience pour le Pôle de Modélisation. L'ensemble des codes sources y est regroupé sous la forme d'une bibliothèque C++ plus proche des besoins d'un informaticien que de ceux d'un mathématicien. Dans le cas présent, la méthode d'intégration retenue utilise une approximation polynomiale (polynôme de Lagrange) de la fonction à intégrer sur chacune des subdivisions de l'intervalle de définition.

4.2.2 Implémentation

Le calcul du champ potentiel magnétique se fait donc en utilisant une petite collection d'objets dont les interactions sont décrites dans le diagramme Fig 4.3.

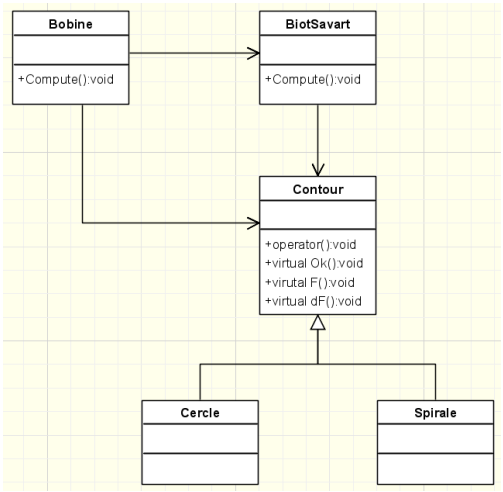

FIGURE 4.3 – Extrait du diagramme UML général : les classes permettant de faire le calcul des champs électromagnétiques

La Bobine est l'objet principal qui calcule le vecteur potentiel champ magnétique $\vec{A}$ au point x, y, z via la méthode `Compute()`. Pour faire cela, il y a d'abord conversion des coordonnées x,y,z dans le repère de la bobine. Ensuite il faut vérifier si le point obtenu n'est pas dans un voisinage trop proche de la bobine. `Bobine` délègue cette tâche au `Contour` de la bobine via la méthode virtuelle `Ok()` qui sera implémentée par chaque type de contour.

Si le point où l'on veut calculer le champ est dans le domaine de validité, le calcul est délégué à un objet de la classe `BiotSavart` qui implémente cette méthode. Quand celle-ci renvoie le résultat, il est ramené dans le repère d'origine puis retourné.

La méthode principale de la classe `BiotSavart` permet de calculer chacune des composantes du vecteur $\vec{A}$. On notera que dans cette méthode, il est possible de fixer la méthode d'approximation utilisée (rectangle, trapèze...) et la précision que l'on souhaite obtenir pour l'intégration numérique. Cette intégration est faite via la fonction `integre` (définie dans la bibliothèque du pôle de modélisation). En plus du type de méthode et de la précision, cette fonction prend en paramètres :

– un foncteur définissant la fonction à intégrer
– les bornes d'intégration

Le foncteur en question n'est autre que `Contour`. L'intérêt principal de l'implémentation proposée ici est que tout ce qui a été mis en place jusqu'à maintenant *est indépendant de la forme de la bobine*. Celle-ci sera alors simplement explicitée par les méthodes virtuelles `F` et `dF` qui définissent respectivement $x(t), y(t), z(t)$ et $x'(t), y'(t), z'(t)$: l'équation paramétrique du contour.

Le calcul du champ généré par une bobine en forme de cercle, en forme de spirale, ou de tout autre forme plus complexe, s'effectue simplement en utilisant le polymorphisme sur la classe `Contour`, et en dérivant celle-ci en autant de contours différents qui implémenteront les méthodes virtuelles `F` et `dF`. Ci-après se trouvent quelques exemples, montrant en parallèle le code et les équations mathématiques correspondantes. Le code a été pensé pour pouvoir être une transcription quasiment directe des équations mathématiques.

Equation paramétrique d'un cercle

$$\begin{aligned} x(t) &= 0 \\ y(t) &= r\cos(t) \\ z(t) &= r\sin(t) \end{aligned} \qquad (4.2)$$

$$\begin{aligned} x'(t) &= 0 \\ y'(t) &= -r\sin(t) \\ z'(t) &= r\cos(t) \end{aligned}$$

Equation paramétrique d'une spire

$$
\begin{aligned}
x(t) &= 0 \\
y(t) &= (r_{min} + pas\ t)\cos(t) \\
z(t) &= (r_{min} + pas\ t)\sin(t)
\end{aligned}
\tag{4.3}
$$

$$
\begin{aligned}
x'(t) &= 0 \\
y'(t) &= -(r_{min} + pas\ t)\sin(t) + pas\cos(t) + pas * cos(t); \\
z'(t) &= (r_{min} + pas\ t)\cos(t) + pas\sin(t)) + pas * sin(t);
\end{aligned}
$$

4.2.3 Exemple d'utilisation

Le code de définition générique des contours ayant été mis en place, nous avons réalisé les premières simulations avec des bobines simples et doubles dans un espace uniforme (une matrice 3D discrète) et calculé le champ potentiel magnétique en chacun des points de la matrice. La figure 4.4 montre que la forme des champs ainsi obtenus est bien conforme au profil de module déjà évoqué en Fig 4.1 et 4.2 page 58.

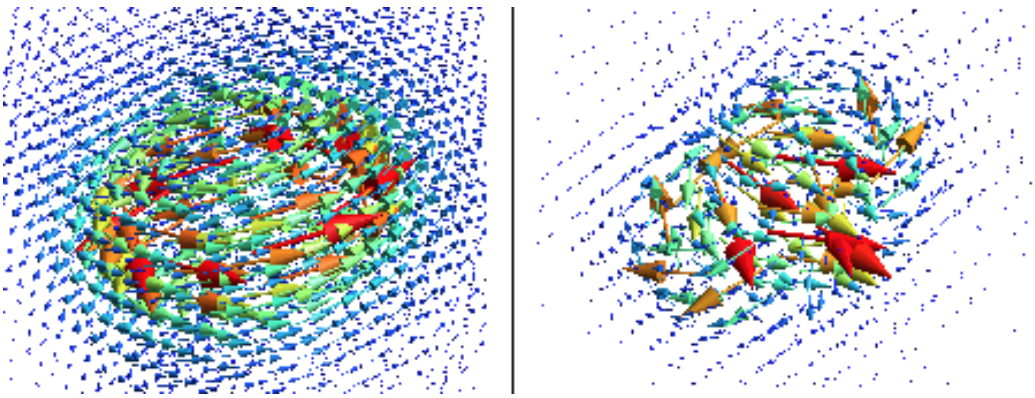

FIGURE 4.4 – A gauche, le vecteur champ potentiel magnétique généré par une bobine simple est bien constitué d'iso-contours circulaires. De même, à droite, le champ potentiel magnétique généré par une bobine double présente bien deux boucles et une focalisation à leur intersection.

Les champs vectoriels exposés sur la figure 4.4 correspondent à une capture d'écran du logiciel gmsh [Geuzaine and Remacle, 2002] que nous avons utilisé pour

visualiser les premiers champs. Ce logiciel permet la visualisation de maillages, vecteurs, champs de vecteurs, modèles géométriques, et peut servir de mailleur.

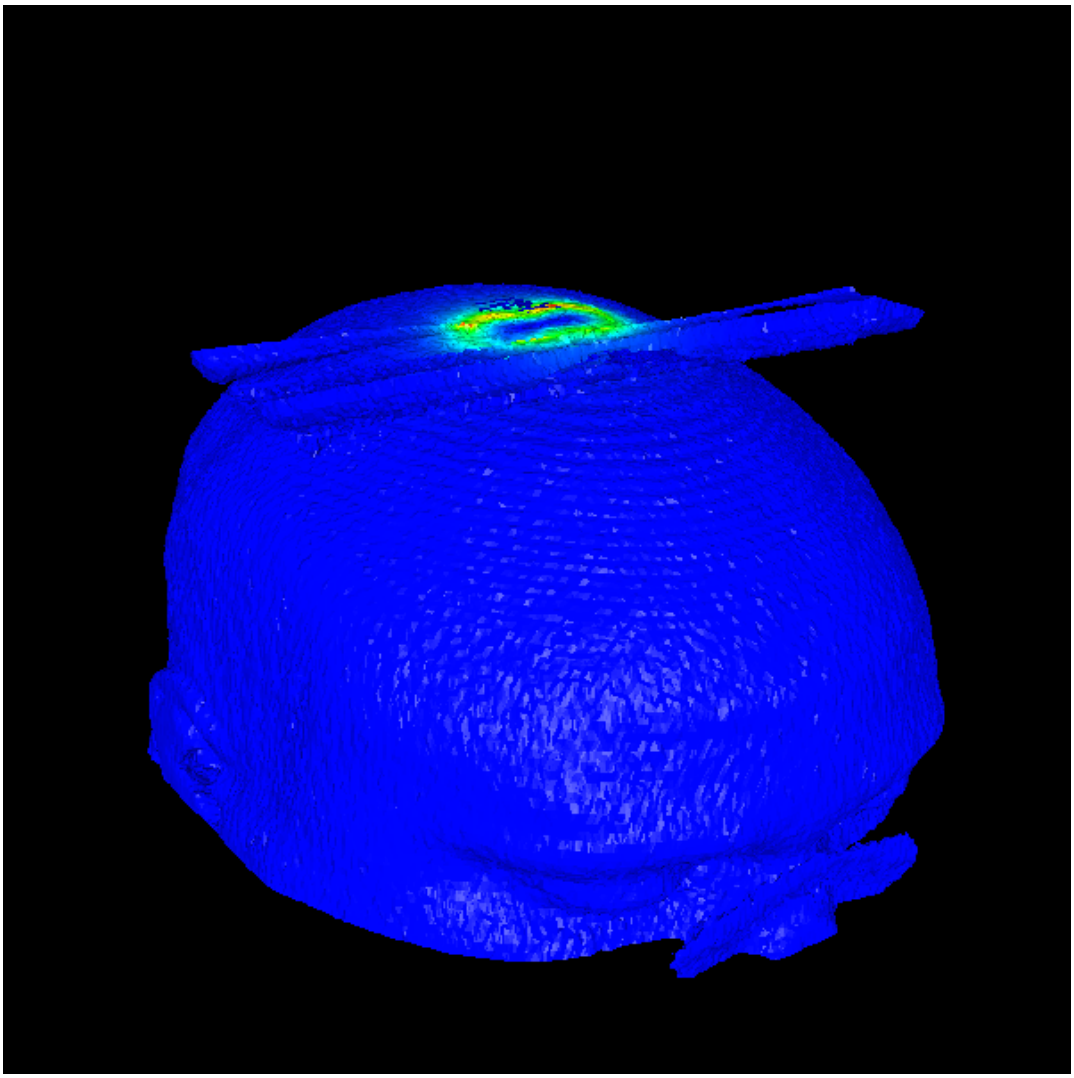

FIGURE 4.5 – Première esquisse de cartographie du champ potentiel magnétique sur un modèle de patient. La bobine utilisée est une bobine double. La barre bleue horizontale est un artefact de l'IRM (recouvrement de spectre).

Lors des premiers développements du module de calcul, le champ était calculé par un programme en ligne de commande qui sauvegardait les résultats dans un format de fichier propre à gmsh, afin d'en faire la visualisation après coup. Par la suite, pour comprendre comment le champ magnétique se répartissait sur la tête du patient, d'autres fichiers ont été générés dans d'autres formats (.mesh du logiciel mEdit de

l'INRIA [Frey, 2000] par exemple) pour avoir une visualisation 3D regroupant un maillage de la tête du patient avec le champ magnétique appliqué dessus (Fig 4.5). La maintenance d'un tel système informatique avec :

- un logiciel fait maison, en ligne de commande, pour générer champs et autres cartographies
- un autre logiciel développé au sein de l'ERIM pour générer des maillages de tête
- un logiciel de visualisation des champs
- un logiciel de visualisation des maillages et de champs

est rapidement devenu ingérable, d'autant plus qu'il était impossible de modifier le moindre paramètre de la bobine, lors de la visualisation, sans recommencer l'ensemble du processus de génération. C'est pour cela qu'il a été décidé de mettre en place un logiciel complet, qui permettrait de modéliser la bobine, générer le modèle de la tête du patient, observer les champs, et tout cela en temps réel. L'application en ligne de commande permettant de calculer les champs est ainsi devenue le noyau de calcul du simulateur. Par ailleurs, de nombreuses fonctionnalités se sont greffées par la suite afin d'améliorer la compréhension des phénomènes relatifs à la SMT.

4.3 Conclusion

Dans ce chapitre nous avons reconsidéré l'équation électromagnétique de Biot et Savart. En transformant cette équation, nous avons montré que le paramètre le plus important de cette équation était la forme du circuit de la bobine. Le logiciel que nous avons développé permet entre autre de visualiser la forme des champs générés par les bobines. Nous avons passé en revue les différents types de bobines et donc les différents types de champs magnétiques générés. Les deux principaux types de bobine sont les bobines simples et les bobines doubles (dite en en 8), la bobine en double cône est une variante de la bobine en 8 car elle dispose d'un angle entre les deux ailes de la bobine. Il existe encore d'autres types de bobines, mais elles restent peu décrites et nous ne les avons pas étudiées.

Nous avons décrit les contours des bobines par des formes paramétriques tridi-

mensionnelles en précisant les trois principales composantes. Nous avons par la suite considéré plusieurs techniques d'intégrations numériques (en partant des méthodes basiques aux techniques de Newton-Côtes et Runge-Kutta) et retenu une technique à base d'approximation polynomiale (utilisant des polynômes de Lagrange) proposée dans l'ouvrage de référence Numerical Recipies et intégrée à une biliothèque C++ développée au sein de la société (Soluscience).

Nous avons proposé quelques classes pour la modélisation des bobines et fait usage d'un foncteur (objet fonction) décrivant le contour de la bobine et de sa dérivée. Enfin nous avons mis en œuvre notre implémentation sur des exemples de bobines simples et doubles. L'étude des champs de vecteurs permet de constater que nous avons obtenu les formes attendues. Nous présentons enfin une esquisse d'une cartographie de champ potentiel magnétique obtenue après l'utilisation de quatre logiciels et un an et demi de travail. En effet, nous devions (1) générer la cartographie des champs ; (2) générer le maillage de la tête du patient ; (3) visualiser les champs et (4) visualiser les champs appliqués sur le maillage. Nous avons donc cherché à automatiser et à unifier au sein d'un même logiciel ces différentes fonctionnalités. Le chapitre suivant va présenter la conception et la réalisation d'un simulateur intégrant tous ces aspects.

Chapitre 5

Le simulateur

Nous avons vu que la faisabilité des calculs de simulation d'une Stimulation Magnétique Transcrânienne nous a conduit à utiliser plusieurs petits logiciels. Passée cette étape de prototypage, nous avons proposé une intégration de tous les aspects considérés au sein d'un seul logiciel. Pour mettre à la disposition des médecins un outil convivial, et pas seulement un prototype brut, nous avons considéré également le développement d'une interface graphique. Le développement d'interface utilisateur se trouve être au cœur de l'activité développement logiciel de Soluscience. Ainsi la mise en place du simulateur a pu profiter de l'expertise générale acquise par les différents ingénieurs de la société.

5.1 Contexte de développement

5.1.1 Cadre de développement de l'interface graphique

L'expérience acquise par Soluscience dans le développement d'IHM se matérialise par un ensemble de bibliothèques permettant de mettre facilement en place des IHM. Il s'agit principalement d'une sur-couche à WxWidget [Smart, 1995] qui est une bibliothèque graphique permettant de mettre en place des logiciels graphiques et ce avec un code compatible avec de nombreux systèmes d'exploitation et compilateurs. Au-delà de la gestion de *widget* (menu, bouton etc) permise par wx, Soluscience a mis en place tout un cadre de programmation permettant de créer des logiciels principalement dédiés à un usage scientifique (science de la vie, médecine, nucléaire etc). Ainsi l'interface décrite précédemment est commune à de nombreux logiciels

profitant des fonctionnalités développées pour d'autres applications.

Profitons de ce paragraphe pour donner quelques données techniques et préciser que le développement s'est fait sous Windows XP en utilisant l'IDE Code-Warrior. C'est actuellement le seul environnement dans lequel l'application tourne mais une branche de développement a été mise en place pour être compatible avec notamment gcc et donc avoir une chance de compiler sur des systèmes d'exploitation tel que Linux et Mac OS.

5.1.2 De l'Interface Graphique et de la ligne de commande

Il s'agit des deux grandes façons d'utiliser un logiciel. Elles ont toutes les deux leurs avantages et leurs inconvénients et sont complémentaires. Le cheminement suivi dans le cadre de cette thèse est un bon exemple de l'importance de la dualité entre ces deux approches. Tout d'abord, en terme de développement la mise en place d'une application en ligne de commande est dans un premier temps plus simple. En effet, cela permet de se concentrer sur les traitements que l'on veut mettre en place et de voir rapidement si l'on arrivera à mettre en place les algorithmes permettant d'atteindre son but (typiquement dans le cas présent : calculer le champ magnétique). Mais au-delà de la personne qui l'a programmé, un utilisateur externe qui veut découvrir le logiciel et explorer les fonctionnalités sera très rapidement décontenancé par la seule ligne de commande qui est pour le moins frustrante. De même, le développeur du module qui veut apprécier les changements engendrés par la modification d'un paramètre préférera certainement une interface graphique.

Dans notre logiciel, ce fut par exemple le cas, quand ne sachant où placer la bobine par rapport au modèle du patient, la mise en place du module de visualisation 3D a permis de recaler simplement la bobine par rapport à la tête et d'offrir une interface graphique permettant de modifier les paramètres de positionnement.

L'interface graphique est aussi une aide précieuse quand l'on veut mettre en place une suite de traitements qui nécessite d'observer l'évolution des données étape par étape et d'affiner par itérations successives les bons paramètres etc... Cela ressemble un peu à une étape de débogage où l'interface graphique permet d'apprécier facilement l'ensemble des données à chaque étape et offre la possibilité de revenir en

arrière etc.

Ainsi l'interface graphique est très efficace pour :

– vérifier l'ensemble du processus sur un cas typique d'utilisation
– vérifier que la procédure est robuste

Cependant, quand il ne reste plus qu'à appliquer ce même processus sur un grand nombre de données d'entrées, l'interface graphique devient un facteur limitant. Une approche ligne de commande (en mode expert en quelque sorte) redevient alors la méthode la plus efficace pour traiter ce grand nombre de données. En effet, une application en ligne de commande est alors, ne serait ce que via la mise en place de script, beaucoup plus facilement utilisable que la répétition des mêmes gestes dans l'interface graphique.

Dans le présent logiciel, la génération du maillage du cortex du patient à partir d'une image IRM de celui-ci est un processus entièrement automatique qui ne nécessite aucune intervention humaine qu'il serait dommageable de faire à la main pour un grand nombre de patients.

On ne s'étonnera pas à ce niveau là qu'une grande partie des programmes Unix soient de simples commandes que l'on exécute depuis un shell...

Compte tenu de ces informations, on comprendra pourquoi tout au long du développement il a été fait en sorte de bien séparer tout ce qui est interface graphique et donc intégration des outils Soluscience. D'ailleurs, eux-mêmes appliquent déjà cette séparation pour être un maximum indépendant de la couche *wxWidgets* qui est directement liée à l'interface graphique. Ainsi la partie calcul a toujours été maintenue dans un C++ le plus standard possible afin d'être compilée sur un maximum de plateformes.

Cette démarche fut particulièrement précieuse quand il fallut mettre en place une partie des algorithmes développés sous la forme de service web pour le projet Aurora. Le temps de développement fut à ce moment là radicalement diminué comparé au temps nécessaire à la DéBorlandisation des algorithmes développés par l'ERIM.

5.1.2.1 Lien avec l'IDM

On retrouvera dans le discours précédent, le terme plateforme que nous avons introduit dans le cadre de l'IDM. Si ici, il prend au premier degré le sens système d'exploitation (Windows *vs* Unix), il n'en reste pas moins que l'on peut le replacer dans le sens plus général de l'IDM et du MDA [Bezivin et al., 2008], [Watson, 2008], [Selic, 2008]. Ainsi le travail décrit plus haut qui consistait à supprimer les références à toute interface graphique peut être considéré comme une transformation de modèle pour faire passer d'une plateforme d'exécution à une autre. Les deux plateformes se trouvent formellement au même niveau d'abstraction puisqu'il s'agira au final dans les deux cas de compiler un ensemble de fichiers C++ afin d'obtenir un exécutable. Mais d'un point de vue pratique on se rendra compte que l'on gagne en terme de nombre de plateformes sur lequel compilera le code source.

Si la première version ne compilait que sur une plateforme Windows, le second code est maintenant fonctionnel sur deux plateformes. Et en réintégrant ce nouveau code à l'interface graphique dont il avait été extirpé on retrouvera le comportement initial. Il s'agit bien là d'un exemple typique de refactoring, qui, s'il ne fait apparaître de nouveau modèle structurel ou métier, permet de passer d'un PSM (*Platform Specific Model*) à un PSM un petit peu moins spécifique. Certes, cette dénomination est un peu cavalière mais il serait présomptueux d'affirmer que cette transformation a permis d'atteindre un modèle PIM (*Platform Independent Model*).

En terme d'architecture globale de logiciel, cette possibilité de séparer un pan de la bibliothèque et de le rendre exécutable en ligne de commande peut se révéler d'une importance capitale quand il faut réaliser des activités logicielles telles que :

- le déploiement des algorithmes sur des systèmes en production et leur utilisation en mode batch
- les tests automatiques de non régression
- la mise à disposition de ses algorithmes sous forme de service (Windows, web)

Mode Batch

Quand l'utilisateur a réussi à identifier via l'interface graphique l'ensemble des traitements qu'il veut réaliser sur un jeu de données, il se peut qu'il veuille appliquer ce même ensemble de transformations sur un ensemble conséquent de données. S'il doit sélectionner l'ensemble des éléments un à un dans l'interface graphique et les faire tourner un à un, il va perdre un temps considérable. L'interface graphique peut certes proposer de sélectionner l'ensemble des éléments, puis lancer le traitement sur l'ensemble des éléments. Cependant, on sera rapidement limité si l'on veut pouvoir accélérer le temps de calcul en distribuant celui-ci sur un ensemble de machines. Un tel déploiement avec un outil en ligne de commande/sans interface graphique sera largement facilité puisqu'il faudra simplement intégrer le code à une bibliothèque de parallélisation ou simplement déployer le programme sur un ensemble de machines sur lequel la charge sera répartie.

Test

On rejoint ici la notion de test que nous présenterons dans une autre partie du document (section 6.4 page 122). Il s'agit de pouvoir vérifier à tout moment que l'application réalise bien les traitements comme elle le doit. Le plus simple pour ce faire est de lancer le programme avec un jeu d'entrées connues et dont on connaît les résultats en sortie et de vérifier pour chaque version du logiciel que les résultats générés par la nouvelle version du programme sont bien ceux attendus.

A moins d'avoir beaucoup d'utilisateurs/testeurs pour faire ce travail, on préférera que cela puisse être fait automatiquement. Dans ce cas, la mise en place de ces tests sera beaucoup plus facile sur un système qui ne nécessite aucune intervention humaine. Il serait totalement possible de mettre en place un système qui va simuler les actions réalisées par un être humain (*Apple Script*, enregistrement de macro etc), mais ces outils sont beaucoup plus lourds à développer ou à mettre en place. Ils sont aussi gourmands en temps de réalisation de test. Notons au passage que ces outils sont cependant indispensables quand on développe une interface graphique et que théoriquement il faudrait vérifier (et donc écrire un test) que chaque action (pensez

uniquement en terme de clic utilisateur) ait le comportement escompté. Même s'il existe des robots pour réaliser les clics automatiquement, l'écriture de tous les scénarii est un travail considérable et on appréciera de pouvoir réaliser une grande partie des tests, celle qui concerne plus spécifiquement le cœur du système de calcul, sans faire intervenir les effets de bord liés à l'interface graphique (*cf.* séparation entre test unitaire et test fonctionnel).

Service

On peut aussi évoquer ici, l'évolution actuelle du développement logiciel dont la mouvance est la mise en place de services. On notera notamment que cette tendance se développe sous le terme SOA : Service Oriented Architecture [Erl, 2004]. Si on peut les voir comme des services type SOAP ou plus généralement WS-*, REST ou encore sous la forme très basique de CGI, il n'en reste pas moins qu'il s'agit de pouvoir prendre un fichier (flux) d'entrée (entrée standard, requête POST, fichier xml) et de produire un flux/fichier en sortie. Là encore un code sans interface graphique sera largement préférable. On appréciera ici de pouvoir faire directement appel via un contrôleur (C) simple dont la spécificité sera de traduire les entrées et de réinterpréter les sorties dans le format souhaité (négociation de contenu (V)) vers le cœur de métier du service déjà mis en place (M) [1].

5.1.3 De la gestion de versions

L'intégration de nombreuses bibliothèques informatiques au sein d'un outil commun nécessite de maîtriser chacune d'entre elles ou du moins le processus de compilation et les spécificités de chacune. L'hétérogénéité des environnements utilisés par les développeurs est aussi un problème majeur. En effet, en l'absence d'une volonté affichée de produire un code portable et que la portabilité ait été vérifiée par les auteurs, il est souvent nécessaire de modifier le code. Outre ces modifications, il est souvent nécessaire d'en faire d'autres pour l'adapter à nos besoins ou de créer des interfaces logicielles permettant de faire les passerelles entre l'outil tiers et la

1. Les notations M, V et C évoquées ici font directement référence au modèle de conception [Reenskaug, 1979], [Reenskaug, 2003], http ://en.wikipedia.org/wiki/Model-view-controller

bibliothèque agrégeante.

Ce travail peut être très délicat car la bibliothèque tierce est souvent inconnue au moment de son intégration. Mais il existe une difficulté supplémentaire souvent en plus de toutes les petites différences entre compilateurs. Une fois toutes les erreurs de compilation et d'édition de liens supprimées, il n'en reste pas moins que la bibliothèque tierce récupérée est dans une version donnée. Si celle-ci est maintenue, elle va évoluer et faire l'objet de modifications. Pour profiter de ces mises à jour, il faudra récupérer les nouvelles versions et y ré-appliquer les modifications qui avaient déjà été effectuées sur la version précédente.

Ainsi chaque changement de version (enfin surtout les versions introduisant des fonctionnalités utiles au logiciel en cours de développement) demandera un travail identique à celui mené la première fois. Afin de limiter ce genre de problème il convient de mettre en place un système de gestion de versions pour les bibliothèques tierces au même titre que la bibliothèque principale. L'utilisation appropriée de *branches* et de *tags* permet alors de considérablement améliorer ce travail d'intégration et de maintenance.

5.1.3.1 Gestion de *Vendor Branches*

La gestion des *vendor branches* (branches pour la gestion des codes tiers) est une des activités les plus délicates de la gestion de la version. En effet, si elle n'est pas pratiquée régulièrement, le risque est d'oublier les manipulations à effectuer à la prochaine itération.

Avant d'essayer de proposer une interprétation somme toute personnelle de la gestion de versions dans le cadre de l'IDM, il convient de présenter un ou deux cas d'étude de manipulation de branches. Nous nous placerons dans le cas de Subversion (SVN) un outil de gestion de code open-source bien connu et assez répandu. En nous attachant à un outil particulier, nous perdons peut-être en généralité mais cela nous permettra de mieux appréhender les concepts fondamentaux.

Le choix de SVN se justifie par le fait que la gestion de *vendor branches* avec CVS (l'ancêtre de SVN) est bien plus compliquée qu'avec SVN. En effet, l'implémentation de CVS a justement montré ses limites sur la manipulation des branches et des tags.

Nous n'aborderons pas ici les logiciels propriétaires de gestion de versions, justement parce qu'ils sont propriétaires et que nous ne les avons jamais utilisés. Notre étude ne tient pas compte non plus des gestionnaires de versions distribués autre que SVK. En effet, comme son nom l'indique SVK est intimement lié à SVN et nous avons eu l'occasion de le manipuler. Les autres (Git, Mercurial) qui se démocratisent aujourd'hui, n'étaient pas encore répandus quand nous avions choisi notre outil de gestion de versions.

Le document de référence au sujet de la gestion de *vendor branches* se trouve encore en tapant dans google ces deux mots clés : le chapitre 7 du livre en ligne sur Subversion [1]. Ce document présente une étude de cas sur un logiciel qui utilise une bibliothèque tierce de calcul mathématique. Jusque là rien de problématique. La difficulté intervient lorsqu'un bug est découvert dans cette bibliothèque ou qu'elle ne prend pas en compte tous les cas souhaités et qu'il faut la modifier pour l'adapter à ses propres besoins.

Sans difficulté, on met le code de la bibliothèque sous gestionnaire de versions et on applique les corrections souhaitées dessus. La difficulté apparaît lorsqu'une nouvelle version de la bibliothèque de calcul sort et qu'elle propose une nouvelle fonctionnalité intéressante que l'on souhaite utiliser : si on intègre directement la nouvelle bibliothèque en remplacement de celle qui avait été modifiée en interne, toutes les modifications apportées auront été perdues. Il faut donc être capable d'appliquer les modifications entre les deux versions de la bibliothèque mathématique sur notre propre version modifiée de celle-ci.

La solution proposée par le manuel est de gérer une *vendor branch* dans laquelle sera stocké l'ensemble des versions de la bibliothèque tierce (ou du moins l'ensemble des versions auxquelles on se sera intéressé indépendamment de l'éventuelle numérotation tierce). Ainsi le passage d'une version à une autre sur cette branche pourra être vu comme un unique *changeset* (ensemble de modifications atomiques) et qui pourra être intégré à un autre endroit de l'arborescence comme par exemple le *trunk* contenant la version initiale de la bibliothèque tierce sur lequel nous avons fait nos modifications.

1. http ://svnbook.red-bean.com/en/1.1/ch07s05.html

La difficulté ici, n'est pas tant d'effectuer la fusion puisque avec SVN elle est automatique, mais celle de faire passer la branche gérant le code tiers d'une version à une autre. Manuellement, regardons comment nous pouvons procéder :

- il faut remplacer les anciens fichiers modifiés par les nouvelles versions
- il faut ajouter les fichier en plus
- il faut supprimer les fichiers qui ont disparu

Dans un premier temps, nous n'essayerons pas de gérer les déplacements de fichiers. Manuellement on peut procéder ainsi (sous couvert de pouvoir réaliser en ligne de commande ou via l'interface du gestionnaire de fichiers en un temps raisonnable)

- supprimer tous les fichiers autres que les .svn
- recopier récursivement les fichiers de la nouvelle version dans l'arborescence vampirisée à l'étape précédente
- repérer les fichiers qui étaient sous gestionnaire de versions et qui n'apparaissent plus et les retirer du gestionnaire de versions (`svn del`)
- repérer les fichiers qui ne sont pas sous gestionnaire de versions et les lui ajouter (`svn add`)

Il peut même être intéressant avant de valider les modifications de faire vérifier les différences sur les fichiers modifiés (*cf* Annexe). Cette dernière opération ne peut être réalisée que s'il y a peu de modifications. Sur de très grosses variations, ces opérations manuelles deviennent rapidement non envisageables.

Heureusement, le manuel de Subversion fait référence à un script qui permet de faire les différents traitements évoqués plus haut automatiquement. Il s'agit d'un script perl `svn_load_dirs.pl`. Pour l'avoir utilisé plusieurs fois et nous être heurtés à des difficultés d'utilisation (chemin récalcitrant, utilisation outrancière des propriétés), nous en déconseillons l'utilisation.

Fort heureusement, il existe d'autres outils pour faire cela. Le premier est *piston* et notons tout de suite qu'il n'a d'intérêt que si la bibliothèque tierce est elle-même sous SVN. Il s'agit d'un script développé en Ruby et issu de la communauté Rails. Le but de piston est de permettre la gestion du répertoire de *plugins* [1] d'une application

1. greffon en français mais cette traduction est très peu utilisée

Ruby On Rails dans laquelle on souhaiterait pouvoir faire ses propres modifications plutôt que d'utiliser un `svn:externals`. La force de piston est dans le cadre de Rails de pouvoir être utilisé au moment où l'on s'apprête à faire une modification sur un plugin. Une commande permet de passer d'un svn :external vers un répertoire s'incluant dans le reste de l'application elle-même sous SVN. Une fois les fichiers importés, il est possible de faire ses propres modifications. Ensuite l'utilisation de la commande piston update récupérera les modifications nouvellement effectuées sur le SVN de la bibliothèque tierce et les appliquera dans le répertoire du *plugin* sans pour autant écraser les modifications.

En fait piston récapitule en une commande les phases de mise à jour de la *vendor branch* évoquée dans le manuel et la fusion des modifications entre deux versions de la bibliothèque tierce. Notons cependant que l'on peut aussi utiliser piston uniquement pour gérer la *vendor branch* (c'est à dire sans effectuer aucune modification sur cette branche et en utilisant une branche spéciale pour les modifications) et d'effectuer la fusion soit même.

L'outil ultime pour la gestion de *vendor branch* est SVK. Quand nous l'avons découvert, il nous a fallu du temps pour comprendre toute sa puissance. Aujourd'hui avec le recul et l'avènement de *Git* (*ie* la gestion de version distribuée), il est clair que SVK était plus en avance que les scripts évoqués précédemment. SVK est lui aussi un programme écrit en Perl et se base sur SVN pour permettre la gestion de plusieurs dépôts de fichiers : locaux ou miroirs de dépôts distants. Il permet ainsi de reproduire dans un dépôt local toutes les modifications effectuées sur celui dont il est le miroir. Cela permet ensuite de ré-appliquer ces patchs sur tout autre miroir ou tout autre dépôt local. SVK est cependant plus performant que piston quand il s'agit de faire un import à partir d'un fichier `.tgz` si la bibliothèque tierce n'est pas sous Subversion. Le fin du fin est bien entendu que d'un import sur l'autre, SVK réalisera les opérations d'ajouts/suppressions/mises à jour réalisées entre deux versions.

Ainsi, avec un miroir local de la *vendor branch* depuis le serveur centralisé sur lequel sont validées les différences entre deux imports il est possible de faire évoluer la *vendor branch*. Maintenir le miroir d'un dépôt peut être cependant gourmand en

espace disque et en temps de synchronisation. Pour être plus efficace, il suffit de synchroniser le miroir sur un état de la *vendor branch* et d'importer directement le contenu du fichier `.tgz` dessus. Les modifications relatives au passage à une nouvelle version de la bibliothèque tierce seront alors directement envoyées sur la *vendor branch*. Il n'y a même plus besoin de savoir quelle version de l'application tierce était localisée sur la *vendor branch*.

SVK en se basant sur SVN a ainsi pu devenir une sorte de pieuvre pouvant se brancher à de nombreux SVN. Pour prolonger la métaphore [1], chacune des tentacules correspond à un dépôt miroir qui vont permettre de ramener les modifications jusqu'à la tête avant de pouvoir être renvoyées dans d'autres appendices. Cependant SVK a aussi hérité de tous les inconvénients de SVN. A l'heure actuelle tant qu'à faire de la gestion de versions distribuée, nous conseillerions Git en espérant que des outils comparables à TortoiseSVN voient rapidement le jour.

5.1.3.2 La gestion de versions et l'IDM

Ce qui suit est une vision somme toute assez personnelle de la gestion de versions. Nous allons faire ressortir de la démarche relative à SVN évoquée plus haut, des notions et des concepts qui se rapprochent de ceux de l'IDM, à savoir modèle et méta-modèle et même transformation.

L'un des constats de l'IDM est que le développement logiciel est trop centré sur le code et qu'il n'est en fait qu'un artefact parmi d'autres de la production d'applications. Il n'en reste pas moins que l'on peut considérer l'ensemble du code source comme étant lui-même un système modélisable par un ensemble de fichiers sur un disque dur. Jusque là rien d'exceptionnel tant que l'on ne définit pas un méta-modèle pour décrire le modèle et mettre en place une relation de conformité qui permettra d'effectuer des transformations sur le modèle.

Cette relation de conformité peut s'énoncer simplement : *être sous gestionnaire de versions*. Pour mieux la comprendre, on peut déjà considérer la différence entre les arborescences générées respectivement par `svn export` et `svn checkout`. Le

1. On notera que le logo de github (service d'hébergement collaboratif de dépôt git) prend aussi l'apparence d'une pieuvre

premier est un répertoire comme les autres, le second est une arborescence contenant tout un ensemble de sous répertoire `.svn`. La présence de ces méta-données permet d'effectuer de nombreuses requêtes et transformations sur le modèle. Pour ne citer que les plus courantes, `svn status`, `svn diff` permettent de savoir s'il y a des modifications en cours sur la version de travail (*working copy*) du code source. Cela permet d'appréhender rapidement si le comportement de l'application a des chances d'avoir changé ou s'il s'agit d'un simple refactoring.

L'arborescence sous gestionnaire de versions peut aussi recevoir des mises à jour (`svn update` ou `svn merge`). Ces mises à jour peuvent alors être vues comme des transformations de modèle au sens de l'IDM. Chaque révision, chaque *changeset* sont alors autant de patchs applicables sur le modèle, instance de code de source. Ainsi même si, comme le fait remarquer Favre dans [Favre et al., 2006], il est dommageable qu'une transformation *correction de bug* ne soit pas généralisable, il n'en reste pas moins qu'une fois identifiée, elle est applicable sur telle ou telle branche (version 1.9 ou 2.0 par exemple) du système.

Nous reviendrons sur la gestion des bugs dans la partie sur les tests (section 6.4 page 122) et sur le tissage de la gestion de versions avec d'autres aspects de la programmation. Remarquons seulement ici que la mise en place de la gestion de versions est complètement orthogonale au reste du développement. S'il est possible théoriquement de faire du développement avec et sans gestionnaire de versions, le gain de productivité est tellement important que l'utilitaire de comparaison de fichiers est plus souvent utilisé que l'éditeur de code. Cela permet d'avoir en plus de la vue proposée par l'IDE principal, une vue listant toutes les modifications (utiles ou non) en cours. On ne peut cependant conclure ce chapitre sans parler des problèmes qui peuvent survenir en cas de mauvaise utilisation d'un outil de gestion de versions.

5.1.3.3 Utilisation dangereuse de la gestion de versions

Nous distinguerons principalement deux mauvaises pratiques qui peuvent conduire à des points de non-retour.

Branches spaghettis

La première est inhérente à la mise en place de branches et nous l'avons rencontré pendant cette thèse. Pour ne plus travailler sur un *trunk* (à l'époque nous étions sous CVS et il conviendrait plutôt de parler de *HEAD*) sans cesse cassé par les modifications faites par les différents développeurs et suite à des problèmes survenant en phase de recette, nous avons décidé de créer des branches pour certains utilisateurs. A partir de là le développement du simulateur n'était plus ralenti par les problèmes introduits suite à des manipulations faites à gauche et à droite. Nous pouvions alors nous concentrer sur les codes et problématiques spécifiques au simulateur.

Par contre un jour nous avons eu besoin de rapatrier une modification depuis le trunk pour pouvoir utiliser une fonctionnalité développée entre temps par d'autres développeurs. Une fusion des modifications de code depuis la plage de révisions concernée a permis de résoudre cette problématique. Par contre à un autre moment, un problème général avait été corrigé sur la branche du simulateur. Il fallait donc rapatrier la correction sur la branche principale, ce que nous faisions par une fusion dans l'autre sens (branche vers le *trunk*).

Par contre quand nous avons voulu refondre la branche du simulateur dans le *trunk*, nous avons dans un premier temps voulu rapatrier les modifications du *trunk* dans la branche du simulateur pour vérifier que tout continuait de fonctionner. Les modifications liées au simulateur ne seraient envoyées ensuite que si tout se passait bien. Ce fut malheureusement la première erreur. Ce rapatriement global du *trunk* impliqua le rapatriement des modifications qui avaient déjà été rapatriées une première fois. De plus, comme la branche commençait à avoir un peu d'ancienneté, quelques conflits apparurent. La résolution des conflits et la vérification de la validité de la fusion prirent un peu de temps. Le *trunk* ayant évolué entre temps, il était de nouveau nécessaire de récupérer les dernières modifications de celui-ci. Par manque d'expérience, nous avons commis une deuxième erreur en refusionnant la totalité du *trunk dans* la branche du simulateur. En plus des dernières modifications, certains des conflits déjà réglés apparaissaient de nouveau.

Ce problème aurait pu être évité en utilisant un ensemble de branches et d'éti-

quettes (*tags*). Mais sous CVS, leur utilisation est peu intuitive et même sous SVN, la multiplication de branches et l'utilisation de tags pour permettre leurs fusions réciproques devient rapidement un plat de spaghettis. C'est tout simplement inutilisable car Subversion n'a pas de *smerge* (s pour *star* et *merge* pour fusion) [1] alors que les autres systèmes comme SVK, Git etc le proposent. Ce type de fusion particulier, va maintenir en interne une liste des fusions effectuées (révision par révision) et ne ré-appliquera pas plus de deux fois la même révision sur une branche donnée.

Pour y parvenir, il faut inverser le problème et recréer une nouvelle branche à partir du *trunk* sur laquelle seront fusionnées les modifications de la branche à réintégrer. Une fois les éventuels problèmes corrigés sur cette branche temporaire, elle peut être fusionnée sur le *trunk*. Par contre, cette manipulation risque de faire perdre l'historique des modifications, les *changesets* se retrouvant regroupés en une seule et énorme révision. Si l'atomicité des révisions est importante dans le cadre du projet en cours, il conviendra d'utiliser un système tiers (la *pieuvre* SVK le fait très bien par exemple) pour rejouer les *commits* un à un. Si l'on compare l'évolution du code source à l'application de fonction mathématique on se rapproche ici de la composition de fonction $CommitNonAtomique = R_i o R_j o R_k$.

Legacy branche

Mais au-delà de ces considérations théoriques et finalement techniquement réparables, nous avions commis une erreur beaucoup plus grave. Sans faire de réintégration régulière et progressive permettant de vérifier régulièrement la compatibilité d'une branche avec le *trunk* dont elle est issue, la branche peut devenir complètement incompatible en terme d'API avec le *trunk* si celui-ci change trop. Ainsi dans notre cas, il y a eu trop de changements technologiques sur le *trunk*. Au final, le simulateur fonctionne toujours mais uniquement via le code présent sur la branche et ne profite plus des évolutions réalisées par ailleurs.

1. La version 1.5 de subversion est sortie depuis l'écriture de cette partie et propose maintenant le *merge-tracking*

5.1.4 Intégration Logicielle

Une grande partie du travail informatique réalisé au cours de cette thèse aura été de l'intégration logicielle. En effet, le simulateur reposant sur l'addition de nombreux traitements informatiques disponibles sous des formes diverses et variées et travaillant sur des données hétérogènes, il a fallu faire fonctionner tout cela ensemble afin de produire un logiciel cohérent et offrant l'ensemble des fonctionnalités désirées en piochant dans celles offertes par les différentes parties.

Le but premier du logiciel était de permettre une simulation d'une séance de stimulation magnétique transcrânienne. Mais nous préférons voir dans celui-ci le cheminement qui a conduit à la mise en place du logiciel tel qu'il est aujourd'hui. On peut ainsi voir dans les fonctionnalités qu'il propose et dans ces cas d'utilisation les questions et interrogations qui ont été posées tout au long du parcours : les éléments informatiques mis en œuvre sont autant de réponses à ces questions afin d'améliorer la compréhension de la SMT.

Par rapport aux autres méthodes présentées dans la première partie, le coeur de calcul qui a été mis en place se rapproche le plus des travaux présentés par [Nadeem et al., 2003] en terme de calcul de champ magnétique. A la différence que dans le cas présent nous nous limitons à la première étape sans calculer la diffusion du champ dans la tête du patient comme le font les auteurs (via une méthode d'impédance).

Mais le logiciel regroupe l'ensemble des fonctionnalités nécessaires à la modélisation, paramétrage de la bobine, génération d'un modèle de la tête du patient, cartographie etc.

5.2 Modélisation de la Bobine

5.2.1 Arbre de Design

Il s'agit de l'extension d'un concept né chez Soluscience dont les premières applications ont été la mise en place d'une scène 3D. Et au-delà de la scène 3D, il s'agit de la visualisation de celle-ci. Ce concept générique a été utilisé depuis dans le développement de nombreuses applications qui ne nécessitaient pas forcément de rendu 3D. Lors de ces développements, ce concept s'est étoffé de notions plus génériques,

notamment grâce au développement d'applications bioinformatiques. Dans le cadre de cette thèse il a été de nouveau utilisé pour la visualisation 3D tout en profitant des fonctionnalités développées dans les autres domaines. Pour être le plus didactique possible nous essayerons d'utiliser un maximum d'exemples pour la génération de scène simple.

5.2.1.1 Décomposition d'une scène graphique

Pour permettre de visualiser en 3D un ensemble d'objets qui interagissent les uns avec les autres, il convient de pouvoir déterminer les relations qui les lient. Ainsi une des premières notions de modélisation est la composition. L'exemple le plus simple, classiquement donné est qu'une voiture est composée de roues, d'un chassis, d'une carrosserie etc. Une roue pourra à son tour être composée de jantes, pneus, boulons etc jusqu'à descendre à un niveau de description que l'on dira atomique.

Un exemple que nous avons déjà évoqué et sur lequel nous reviendrons par la suite est qu'une bobine en forme de 8 est la composition de deux bobines simples. La modélisation retenue pour modéliser la scène et par la suite la rendre en 3 dimensions est l'utilisation d'un arbre ou chaque noeud va êre un élément de la modélisation et aura pour fils les objets qui le composent et ainsi de suite jusqu'aux éléments atomiques qui correspondront aux feuilles de l'arbre.

5.2.1.2 Parcours de l'arbre de design

L'affichage de la scène 3D va alors se réaliser en affichant chacun des noeuds de l'arbre en effectuant un parcours en profondeur. Informatiquement il suffit pour chaque noeud de définir une méthode d'affichage (qui peut ne rien faire si le noeud ne sert qu'à regrouper un ensemble de sous objets dont l'affichage définira entièrement l'objet père). En plus de la méthode d'affichage d'autres méthodes telles que des méthodes d'arrêt peuvent être mises en place afin de couper une branche de l'arbre et ainsi en interdire l'affichage et pouvoir ainsi se focaliser sur une autre partie de l'arbre.

5.2.1.3 Notion de graphe hiérarchique

En plus de la modélisation sous la forme d'un arbre qui a ainsi été mise en place (ce qui permet d'exporter facilement une scène sous la forme d'un fichier type XML), une notion de graphe hiérarchique a été rajoutée. Une fois défini, un graphe (et donc les différents objets qui le composent) peut devenir lui-même un sous arbre d'un autre objet et servir ainsi de modèle qui peut être utilisé à de nombreux endroits d'un objet englobant. La notion de modèle est elle-même importante, car la modification du modèle entraînera la modification de toutes les instances de celui-ci.

5.2.2 Exemple de modélisation : Une bobine en forme de 8

5.2.2.1 La bobine simple et ses paramètres

Comme nous l'avons déjà dit une bobine en forme de huit est la composition de deux bobines simples. Nous allons donc commencer cette partie par la représentation mise en place pour une bobine simple. Celle-ci se caractérise presque exclusivement par son rayon. Plutôt que d'introduire ici une notion de composition dans laquelle la bobine serait composée d'un rayon, le rayon va plutôt être une propriété (attribut) de la bobine. En terme de représentation dans l'arbre de design, la propriété rayon sera une feuille de l'arbre mais en terme de fonction d'affichage, seul le noeud *BobineSimple* la définira. Nous pouvons donner ici quelques indications sur la méthode d'affichage. Le rendu étant effectué en OpenGL, l'affichage consistera à dessiner un ensemble de points qui suivent le tracé de la bobine qui seront reliés entre eux.

5.2.2.2 De la bobine simple à la bobine double

Pour modéliser une bobine double, nous utiliserons deux bobines simples. Mais si cela suffit à renseigner la notion de composition, cela ne suffit pas pour décrire la géométrie de la bobine double et comment deux bobines simples vont être positionnées l'une par rapport à l'autre. Pour ce faire nous allons introduire un ensemble de transformations (translation+rotation) afin de positionner les deux bobines en face l'une de l'autre (et en introduisant une rotation supplémentaire sur l'une d'elle pour prendre en compte le fait que la circulation du courant va se faire dans l'une, en

sens horaire et dans l'autre dans le sens opposé). En terme de fonction d'affichage, celle des transformations effectue une modification de la matrice de modélisation de façon à ce que lors de l'appel des fonctions d'affichage des bobines, le tracé se fasse dans le bon repère.

Pour que les deux cercles soit jointifs, il faut que les translations soient égales aux rayons des deux bobines. Pour s'assurer que la modification d'un des paramètres rayons modifie l'autre rayon ainsi que les translations de chacune des bobines un système de fonction de rappel (*callback* en anglais) a été mis en place de sorte que la modification d'un paramètre entraîne la mise à jour des autres

Il est temps de préciser qu'en terme de graphe hiérarchique, il est possible de faire en sorte que l'un des paramètres rayons soit directement visible comme une propriété au niveau du modèle de bobine en 8 et qu'en utilisant le système de mise à jour exposé plus haut la simple modification de ce paramètre modifie simultanément les deux rayons et les distances respectives entre les deux centres de bobines

5.2.3 Exemple proche de la réalité physique

Avant de présenter un exemple de modèle un peu plus proche de la réalité physique et qui donc fait intervenir de nombreux paramètres, il convient sans doute de présenter le fonctionnement de l'interface graphique utilisée et notamment comment chaque paramètre de la bobine va être représenté dans le logiciel.

5.2.3.1 Architecture de l'interface graphique

Elle se compose de trois grandes parties qui se retrouvent dans la plupart des applications développées par Soluscience et qui correspondent à une analyse de la problématique générique des GUI pour des logiciels scientifiques :

– l'arbre de paramètres. Il s'agit de la partie supérieure gauche dans la figure 5.1. Cela permet de naviguer dans les différents objets, paramètre, groupe de paramètres gérés par l'application. Nous retrouverons ce même arbre plus tard quand nous nous intéresserons par exemple au positionnement de la bobine par rapport à la tête du patient. Dans le petit exemple présenté ici, il présente un ensemble d'objets *Transformation* (géométrique que nous ne développerons

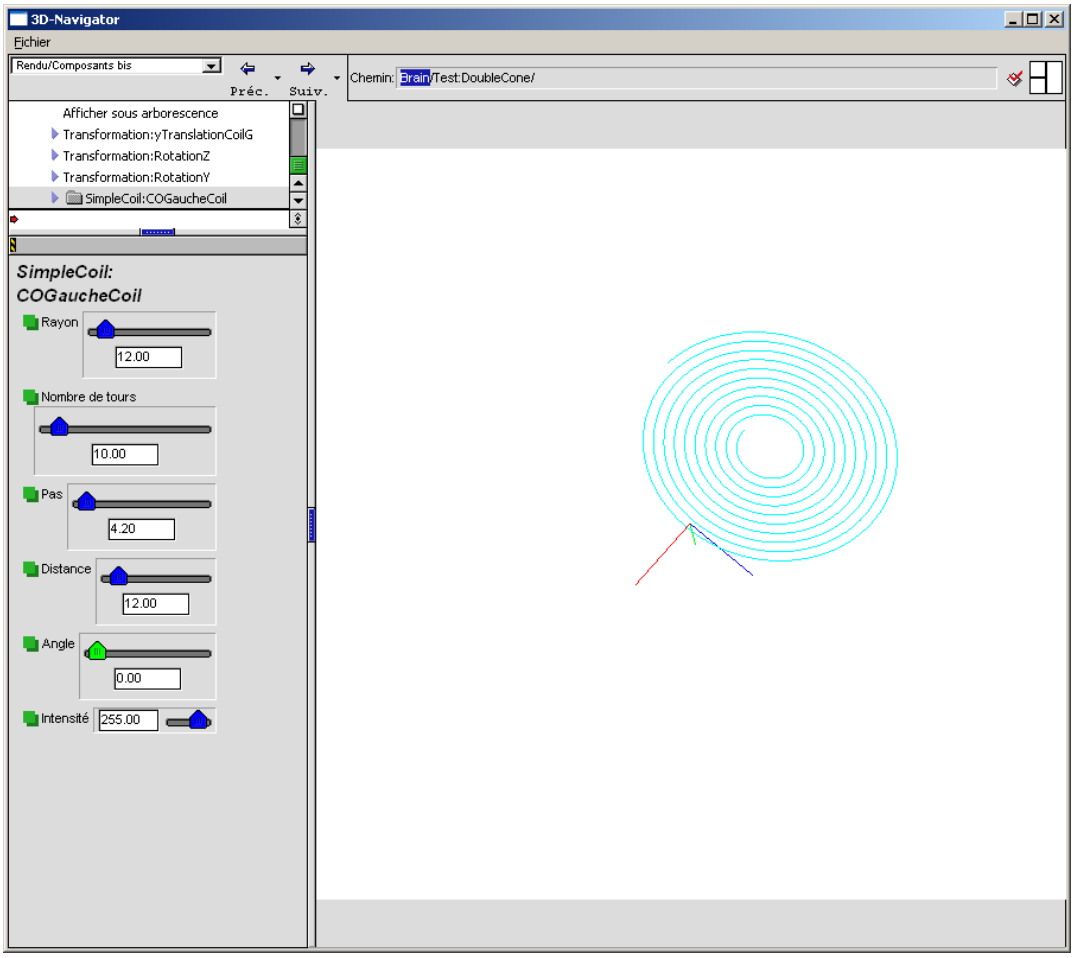

FIGURE 5.1 – Les 3 parties de l'interface graphique

pas ici) et qui permettent de placer/orienter un objet (SimpleCoil dans cet exemple) à une position donnée dans un espace 3D.

– le panneau de paramètres qui représente les paramètres de l'objet sélectionné dans l'arbre de paramètres, en l'occurrence une bobine dont l'équation paramétrique est une spirale
– spirale que nous retrouvons représentée en 3D dans la dernière partie de l'interface (Canvas OpenGL)

Ce qu'il est important de noter ici, c'est que le panneau de paramètres permet de fixer les paramètres rayons, nombre de tours et pas qui correspondent aux paramètres utilisés dans les équations paramétriques 3D écrites plus haut.

5.2.3.2 La bobine Medtronic MC-B70

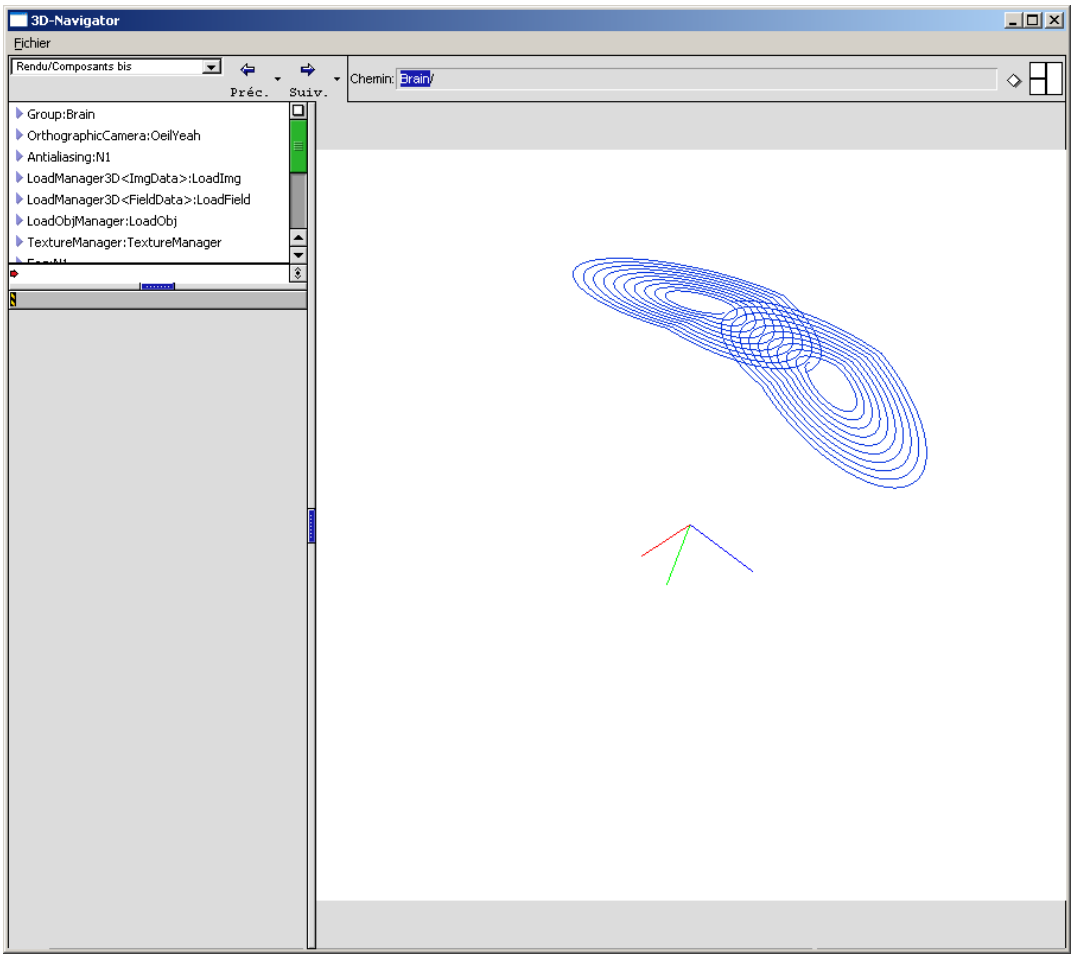

FIGURE 5.2 – Une modélisation de la bobine Medtronic MC-B70

En utilisant la modélisation géométrique réalisée par [Thielscher and Kammer, 2004] à partir d'images radiographiques effectuées sur la bobine en forme de huit de type Medtronic MC-B70, nous avons introduit une équation paramétrique permettant de la modéliser.

Cela permet de prendre en compte le fait que les enroulements dans chacune des ailes de la bobine n'étaient pas plans et que l'enroulement modélisable par deux spirales non jointives se recoupait au niveau de l'intersection des deux ailes. Comme on peut le voir sur la Fig 5.2 il reste des petites imperfections dans la modélisation notamment au niveau de la continuité des deux ailes. L'enroulement présente un

petit saut car dans le modèle choisi la bobine est modélisée par la juxtaposition de deux bobines en forme de spirale. Il est difficile d'évaluer les erreurs de calculs engendrées par ces approximations et une étude comparative n'a pu être menée faute de temps.

5.3 Modèle de cerveau et gestion des maillages

Comme exposé plus haut, des travaux précédents réalisés au sein de l'ERIM, avaient abouti à la mise en place d'un logiciel de génération de maillages représentant la surface de la tête du patient en utilisant une implémentation des marching-cubes.

5.3.1 Marching-cubes

L'algorithme des Marching-cubes a été présenté par Lorensen et Cline dans [Lorensen and Cline, 1987]. Le but de cet algorithme est de construire une isosurface dans un espace volumique donné quantifiant en chacun de ses points une mesure donnée. C'est typiquement le cas d'une matrice IRM dans laquelle chaque voxel correspond à l'expression de la quantité de protons présents dans les tissus.

La construction de l'isosurface, se fait par un parcours de l'ensemble des voxel. Par chacun d'eux, une analyse est faite suivant la valeur de la quantité mesurée en chacun des sommets du voxel par rapport à la valeur seuil de l'isosurface à reconstruire. Le nombre de combinaisons est très grand, mais les auteurs ont montré qu'à l'aide de considérations géométriques (symétrie, rotation etc), il est possible de se ramener à 15 cas pour chacun desquels ils donnent un ensemble de quelques facettes triangulaires à générer pour le voxel analysé. La concaténation des facettes générées pour chaque voxel donne l'isosurface sur l'ensemble du volume.

5.3.1.1 Détermination de la valeur seuil

Au même titre que l'implémentation de l'algorithme des marching-cubes nous avons utilisé une routine développée précédemment par l'ERIM qui effectue une analyse par histogramme afin de déterminer la valeur du niveau de gris correspondant au cortex du patient.

Ces deux outils permettent d'obtenir quasi-automatiquement un maillage du cortex du patient mettant en avant les circonvolutions. Techniquement la fonction d'affichage parcourt une à une des facettes triangulaires calculées par l'algorithme des marching-cubes.

On peut voir sur les figures 4.5 et 5.3 deux exemples de maillages générés avec les marching-cubes. Le premier a été fait sur une image IRM brute, la reconstruction nous donne donc une surface correspondant au scalp du patient. Le deuxième maillage a été obtenu en utilisant l'algorithme sur une image IRM ayant été segmentée et de laquelle on a retiré le crâne et le LCR.

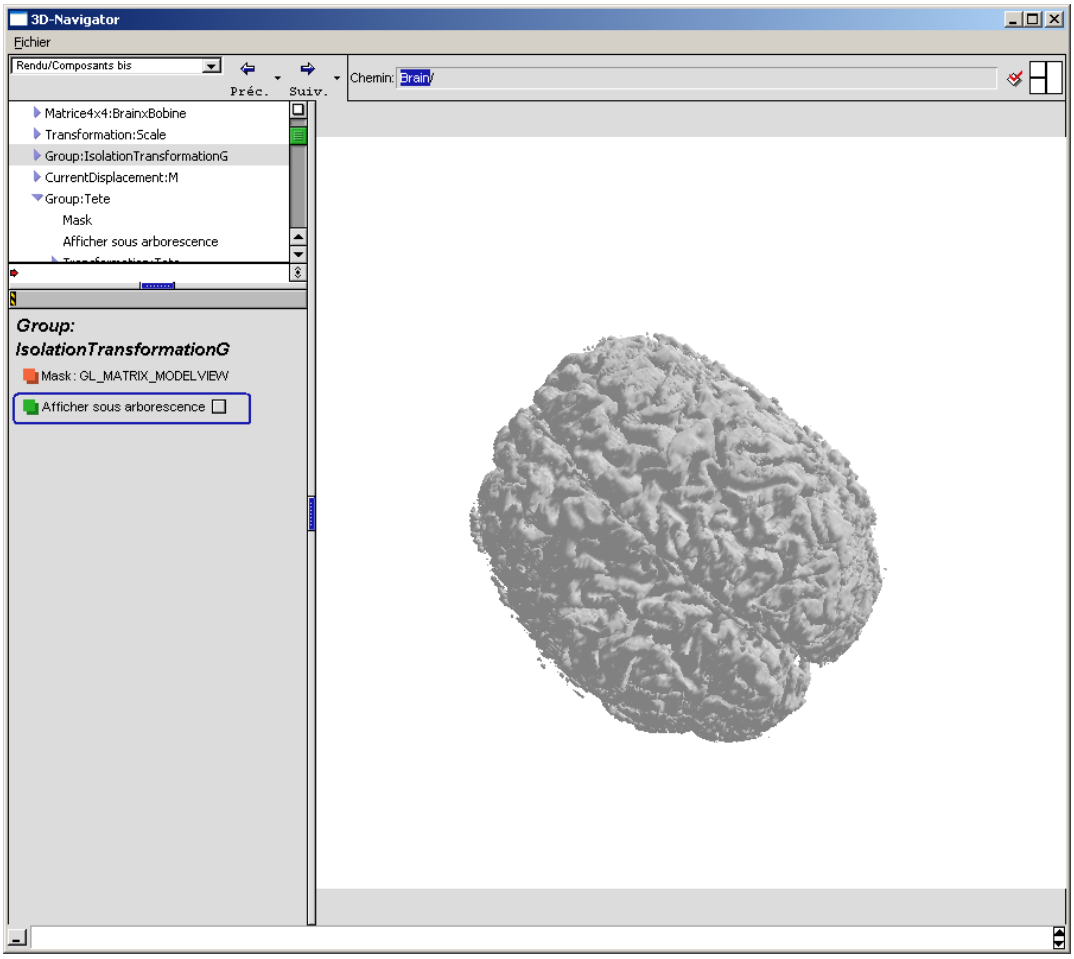

FIGURE 5.3 – Reconstruction du cortex à partir d'image IRM en utilisant les marching-cubes

Par contre la taille des maillages générés est très importante ce qui rend délicat

l'utilisation de ces derniers. C'est pour cela que nous avons essayé de mettre en place des algorithmes pour réduire leur taille

5.3.1.2 Diminution de la taille du maillage

Intégration de VTK

Comme nous l'avons vu précédemment, l'algorithme des marching-cubes génère un maillage surfacique dont le nombre de sommets et de faces est trop gros pour permettre une manipulation facile. Une des premières idées pour réduire le nombre de facettes est d'utiliser un algorithme de décimation. Il s'agit d'un algorithme qui va essayer de diminuer le nombre de facettes en jouant sur les propriétés locales d'un maillage [Schroeder et al., 1992]. Par exemple si plusieurs facettes avec des arêtes communes sont toutes dans un même plan, l'algorithme les remplacera par des triangles plus grands.

La mise en place d'un tel algorithme demande informatiquement de disposer d'un ensemble de routines de calcul permettant de faire des calculs d'angle solide, des triangulations type Voronoï/Delaunay etc... Le temps de développement (et surtout de test) de tels algorithmes est relativement lent, c'est pour cela que nous avons recherché une bibliothèque incluant ces différentes fonctionnalités. Nous nous sommes orientés sur VTK [Schroeder et al., 2000], notamment développé par les auteurs de l'article de décimation précédemment cité qui offrait une implémentation d'un algorithme de décimation.

La distribution de VTK n'offrant pas de projet pour Code Warrior (IDE utilisé par Soluscience), la première étape fut de compiler la bibliothèque VTK avec l'utilitaire *Cmake* développé par Kitware (qui développe aussi VTK). Il s'agit d'une alternative à *make*. Puis en nous basant sur un des exemples d'utilisation fournis avec la distribution, nous avons pu mettre en place un simple programme en ligne de commande permettant de lire un fichier de maillage au format VTK, de décimer celui-ci et de sauvegarder le maillage résultant au format VTK.

Les premiers essais sur des maillages fournis avec la distribution permettaient d'apprécier le fait que l'algorithme fonctionnait correctement (les exemples en tcl/tk fournis avec la distribution ne donnant pas l'impression de fonctionner). Pour pour-

suivre l'intégration de VTK dans le logiciel alors en place il a fallu écrire des méthodes pour permettre d'exporter les maillages générés par le logiciel au format VTK ainsi qu'une méthode pour charger des maillages enregistrés au format VTK dans le simulateur.

Utilisation d'un générateur de parser

L'approche retenue pour traiter un fichier (déjà utilisée pour charger en mémoire d'autres types de fichiers de maillage) est de considérer le fichier texte comme une source obéissant à une certaine grammaire et de compiler celui-ci pour obtenir la représentation mémoire correspondant à la structure utilisée dans le reste du programme. Pour définir la grammaire et en faciliter la compilation/lecture nous avons utilisé ANTLR [Parr and Quong, 1995] qui permet de définir les règles définissant le maillage et d'y associer les actions à effectuer pour charger le maillage en mémoire.

Une fois les méthodes de lecture/écriture mises en place, il a été possible d'exporter les maillages générés par les marching-cubes, de faire tourner l'exécutable en ligne de commande décrit précédemment et de recharger au sein du logiciel le résultat de la décimation.

Les résultats ainsi visualisés n'ont pas été à la hauteur de ceux attendus. L'effet de la décimation ne diminuait pas de beaucoup le nombre de facettes et visuellement, les résultats observés étaient nettement moins bons que les maillages générés directement par l'algorithme des marching-cubes (les circonvolutions du cortex apparaissant nettement moins bien).

Devant le non succès de l'utilisation de cet algorithme, l'intégration s'est ainsi limitée à un appel automatique à l'exécutable décrit précédemment par le programme principal après paramétrisation de l'algorithme (taux de décimation) via l'interface graphique. L'export et l'import du maillage au format VTK étant bien entendu rendus transparents à l'utilisateur.

Intégration de Cgal

Une autre option pour réduire la taille du maillage fut l'utilisation de l'algorithme des alpha shape [Edelsbrunner and Mücke, 1992]. Il s'agit d'un algorithme qui à

partir d'un nuage de points va reconstruire un maillage définissant au mieux la forme du nuage de points. L'idée est alors d'appliquer cet algorithme sur le maillage généré par les marching-cubes en ne considérant que les points de celui-ci. Ainsi le résultat des alpha-shape ne gardant que les facettes définissant l'extérieur du maillage de cortex permet de réduire la taille de celui-ci.

Une fois de plus l'implémentation de l'algorithme des alpha-shape fait intervenir des routines travaillant sur la topologie des maillages, la co-circularité de sommets et autre triangulation. La mise en place de l'algorithme des *alpha-shape* étant dépendante du bon fonctionnement de toutes ces sous-routines (difficiles à tester) nous a fait préférer, comme pour l'algorithme de décimation, de recourir à une bibliothèque informatique déjà existante.

Notre choix s'est orienté vers Cgal [Fabri et al., 1996] qui fut l'une des rares implémentations de cet algorithme. La licence de Cgal étant avec Copyleft et obligeant que toute utilisation de celle-ci donne le droit aux auteurs de cette bibliothèque de réclamer le code avec lequel leur bibliothèque était utilisée, la seule intégration possible fut celle retenue précédemment pour VTK. Le programme principal appellera des exécutables (qui eux répondront aux critères de Copyleft de Cgal) et travailleront sur des fichiers exportés au format Cgal.

On notera que la principale difficulté d'intégration outre l'écriture des fonctions de lecture/écriture de fichier au format Cgal fut la compilation de la bibliothèque Cgal elle-même de par sa dépendance avec la Boost Graph Library [Siek et al., 2002] qu'il fallut aussi étudier/intégrer du moins pour la partie header/template.

De manière similaire à la tentative d'utilisation de VTK, l'utilisation de l'algorithme des Alpha-Shape sur des exemples simples de maillage (limitation des marching-cubes sur un espace réduit) donnait de bons résultats et effectivement permettait de définir la forme du maillage. Mais, comme précédemment cet algorithme avait tendance à faire disparaître les circonvolutions du cortex. De plus, son utilisation sur la totalité d'un maillage généré par les marching-cubes n'arrivait pas à terme à cause d'un manque de mémoire.

5.3.2 Placement relatif du maillage et de la bobine

De la même façon que nous avons pu placer précédemment une bobine simple en face d'une autre, un ensemble de transformations ont été utilisées pour permettre de placer la bobine à la position souhaitée au-dessus de la surface du cortex. De même, d'autres transformations permettent de choisir l'orientation de la bobine par rapport au cortex. Cela permet de connaître à tout moment quelle est la transformation à appliquer au point du maillage pour les passer dans le repère de la bobine et ainsi pouvoir calculer pour chacun d'eux la valeur du champ potentiel magnétique via le module de calcul décrit précédemment.

5.3.3 Application de champ sur le modèle de surface du cortex

Les temps de calculs nécessaires à celui du champ potentiel magnétique peuvent être un petit peu longs. Aussi, nous avons essayé de rendre le logiciel le plus temps réel possible en effectuant un pré-calcul du champ magnétique et en appliquant directement ces pré-calculs sur la surface plutôt que de recalculer le champ en chacun des points.

5.4 Résultats

Nous présenterons dans ce chapitre quelques exemples de paramétrisations qui ont pu être effectuées et les résultats des calculs obtenus.

5.4.1 Calcul dans un plan

Des exemples de calculs du module du champ potentiel magnétique ont déjà été présentés dans la partie précédente afin d'illustrer la focalisation de la bobine en forme de 8 par rapport à la bobine simple. Ils sont obtenus en utilisant un maillage régulier plan près duquel on positionne la bobine (en utilisant le même principe que pour placer la bobine par rapport au maillage de surface corticale) : le champ est appliqué sur la surface sur laquelle est ensuite appliquée une petite distorsion suivant l'axe des z (Fig 4.1).

5.4.2 Importance de la distance locale entre la bobine et les circonvolutions

Le but de cette partie est d'exposer un point qui nous semble bien souvent sous estimé dans la littérature ou alors vaguement évoqué dans les discussions et restant bien dans le flou des hypothèses à étudier. Tout au long de cette thèse notre seul but a été d'essayer de construire un outil afin de nous aider à visualiser et comprendre les phénomènes et les effets (au sens large) liés à la SMT. Le temps lié au travail informatique nécessaire à la mise en place de celui-ci n'a pas forcément permis de mettre en place un modèle de calcul autre que celui utilisant les équations les plus simples. De même, les approximations utilisées sont celles qui permettaient d'obtenir des systèmes pour lesquels nous disposions d'outils informatiques pour les résoudre.

Mais la mise en place de l'infrastructure informatique et son utilisation à des fins de modélisation de placement de bobine au-dessus de la tête du patient a eu pour effet de bord la constatation de phénomènes peu exposés ou non étudiés.

5.4.2.1 Conditions d'observation

Dans le cadre de la modélisation des différentes bobines et la mise en place de bobines à deux ailes (double cône, en forme de 8), il a été développé une structure composée de deux bobines simples (cercle) collées l'une à l'autre en un point. Ce modèle présentait aussi un axe de liberté au niveau du point de jonction afin de permettre d'introduire un angle entre les deux bobines.

La fonctionnalité suivante du logiciel qui était en train d'être testée était le placement de la bobine par rapport à une reconstruction surfacique du cortex du patient :

- positionnement à une position (x,y,z)
- orientation de la bobine par rapport à la surface pour obtenir un placement tangentiel à celle-ci

C'est en effet, dans cette position que les effets de la stimulation sont rapportés par la littérature comme étant les plus importants et se situant sous l'intersection des deux ailes

Une fois le positionnement effectué, le calcul du champ sur la surface à proximité

de la bobine a pu être effectué en utilisant le module de calcul basé sur Biot & Savart.

5.4.2.2 Observation

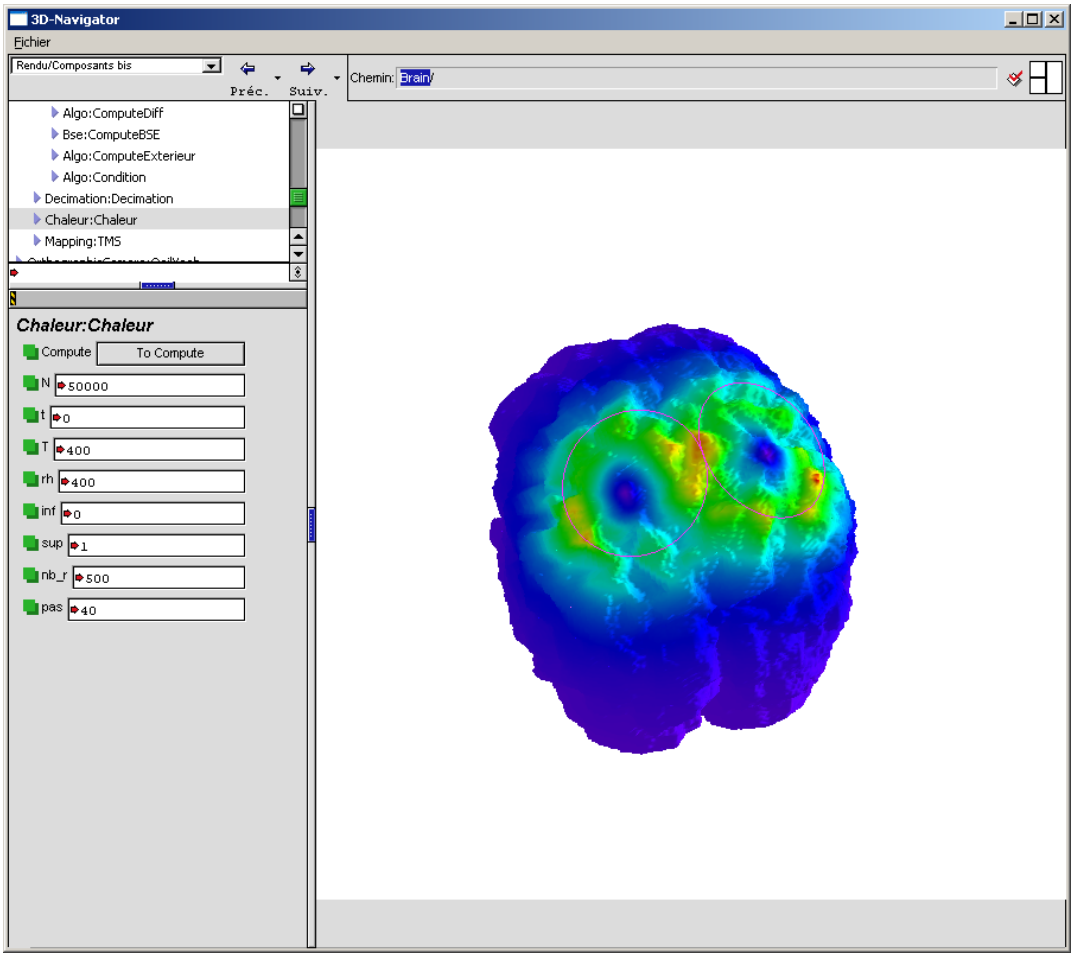

FIGURE 5.4 – Pas de focalisation dans certains cas pathologiques

Par contre le résultat ne fut pas celui escompté et la focalisation attendue sous l'intersection des deux ailes ne fut pas observée. Les valeurs maximales atteintes par le champ étaient en effet réparties sur l'ensemble du cortex correspondant aux zones où le cortex était au final le plus proche de l'enroulement de la bobine comme on peut le voir sur la figure 5.4). Par contre le fait de tenir la bobine à *l'envers* en présentant l'intersection des deux ailes au plus proche du crâne permet d'obtenir une meilleure focalisation (Fig 5.5). Nous n'avons pas eu le temps de comparer les ordres de grandeur observés pour ces positions.

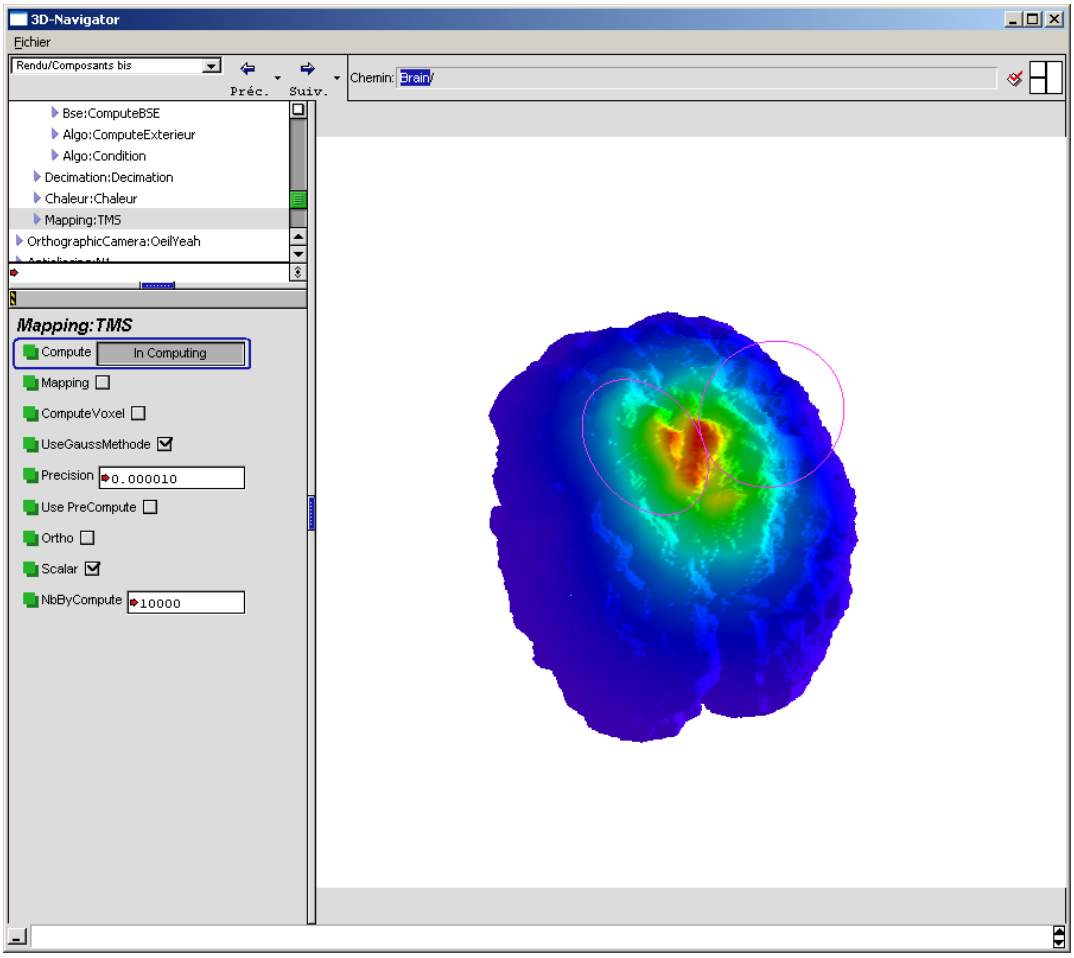

FIGURE 5.5 – Meilleure focalisation en tenant la bobine à *l'envers*

5.4.3 Diffusion

5.4.3.1 Outil de visualisation d'une matrice 3D

Largement inspiré de fonctionnalités utilisées dans différents logiciels de traitement d'image médicale, ce petit objet lui-même composé de 3 plans (cf arbre de design) peut être branché sur n'importe quelle structure implémentant une interface de matrice 3D qui associe à un triplet i,j,k une valeur. C'est typiquement le cas d'une image IRM que l'on va alors pouvoir visualiser coupe par coupe, ainsi que dans les deux plans orthogonaux au plan de coupe (Fig 5.6)

L'extension de ce concept est bien entendu la visualisation d'une image de coupe dans n'importe quel plan et plus uniquement ceux définis par les 3 directions car-

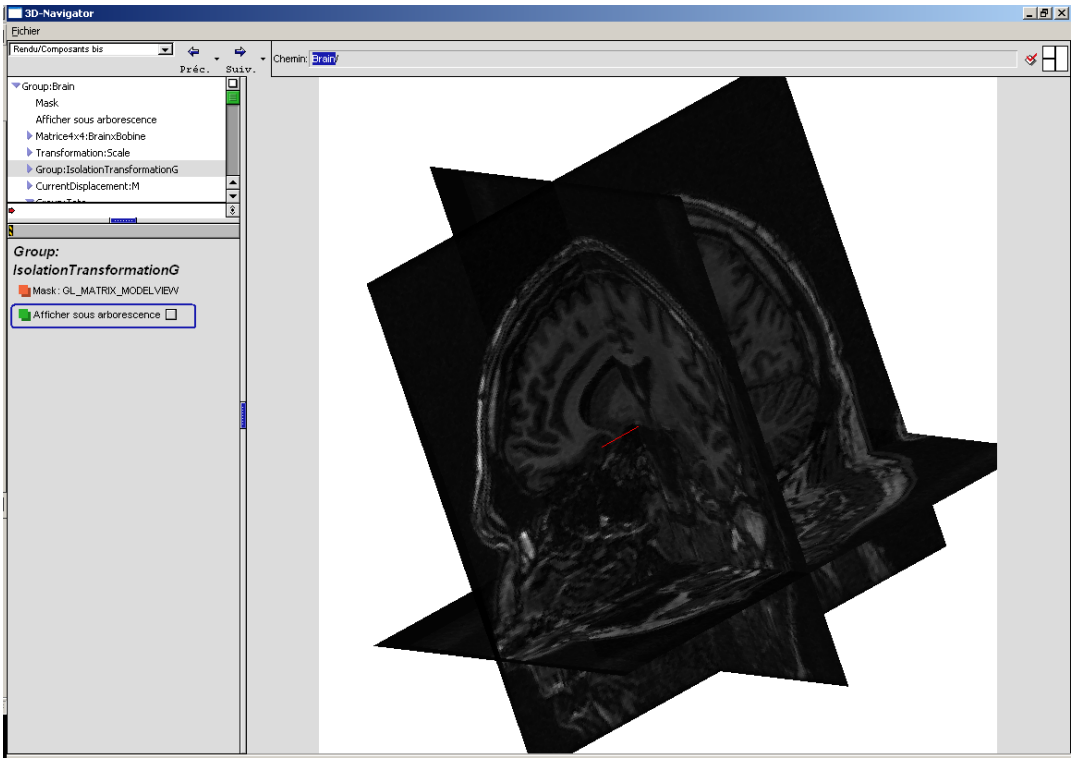

FIGURE 5.6 – Visualisation de 3 coupes orthogonales d'une image IRM en 3D

tésiennes. Cette fonctionnalité n'est cependant pas encore en place car le travail présenté dans le paragraphe précédent correspondait à une étude de faisabilité qui dans un premier temps se voulait simple.

5.4.3.2 Affichage de la diffusion

Alors que la cartographie d'un champ sur la surface corticale permettait d'avoir une idée de là où le champ était le plus fort sur la surface, l'étude de la diffusion du champ ne pourra se faire que par une étude volumique. Ainsi la visualisation de celle-ci ne pourra pas se faire par une application sur une surface, mais sera plus pertinente via l'outil de visualisation de matrice 3D exposé dans le paragraphe précédent. Le calcul de la diffusion en lui-même n'a pas encore été implémenté à ce jour.

5.4.4 Quelques métriques sur le logiciel réalisé

Nous allons dans cette partie donner quelques éléments permettant d'appréhender la complexité du logiciel obtenu. Comme nous l'avons vu dans le chapitre précédent, le nombre de classes nécessaires à la réalisation des calculs tournait autour de cinq. La mise en place de l'interface graphique a complètement fait exploser ce nombre. En effet, chaque élément de l'interface (*widget*) nécessite plusieurs classes :

- une classe indépendante de tout contexte graphique et qui décrit comment dessiner le composant. Cela permet de faire un maximum de choses en s'abstrayant totalement de la bibliothèque graphique utilisée.
- une classe purement graphique basée sur wxWidget qui permettra de le dessiner à l'écran. Grâce à l'abstraction réalisée par la classe précédente il est théoriquement possible d'utiliser une classe reposant sur une bibliothèque graphique (Qt, GTK)
- une classe décrivant les comportements clavier
- une classe décrivant les comportements souris

Mais pour notre travail de modélisation, nous avons principalement été utilisateur des composants déjà développés au sein de Soluscience. Par contre, un important travail d'intégration de ses composants a été effectué pour réaliser la liaison entre 40 fonctions, objets ou concept de visualisation 3D et le reste de l'interface graphique. Chacun de ses 40 *handlers* (nom utilisé en interne chez Soluscience) donne aussi lieu à la mise en place d'une classe.

Ces éléments nous ont permis par la suite de mettre en place les scènes tridimensionnelles décrites dans les parties précédentes (maillages, bobines etc) ainsi que les différents traitements et leurs chaînages. La mise en place de ces éléments a donné lieu à la création d'une vingtaine de classes qu'il a fallu à chaque fois doubler pour réaliser l'interfaçage entre la partie graphique et les classes purement mathématiques proposées par les différentes bibliothèques utilisées (VTK, Cgal etc)

Pour donner une idée de la complexité, on pourra se référer à la figure 5.7 montrant une capture d'écran de la documentation généré par *Doxygen*. Cette page montre les inclusions des différentes classes *handlers* du simulateur.

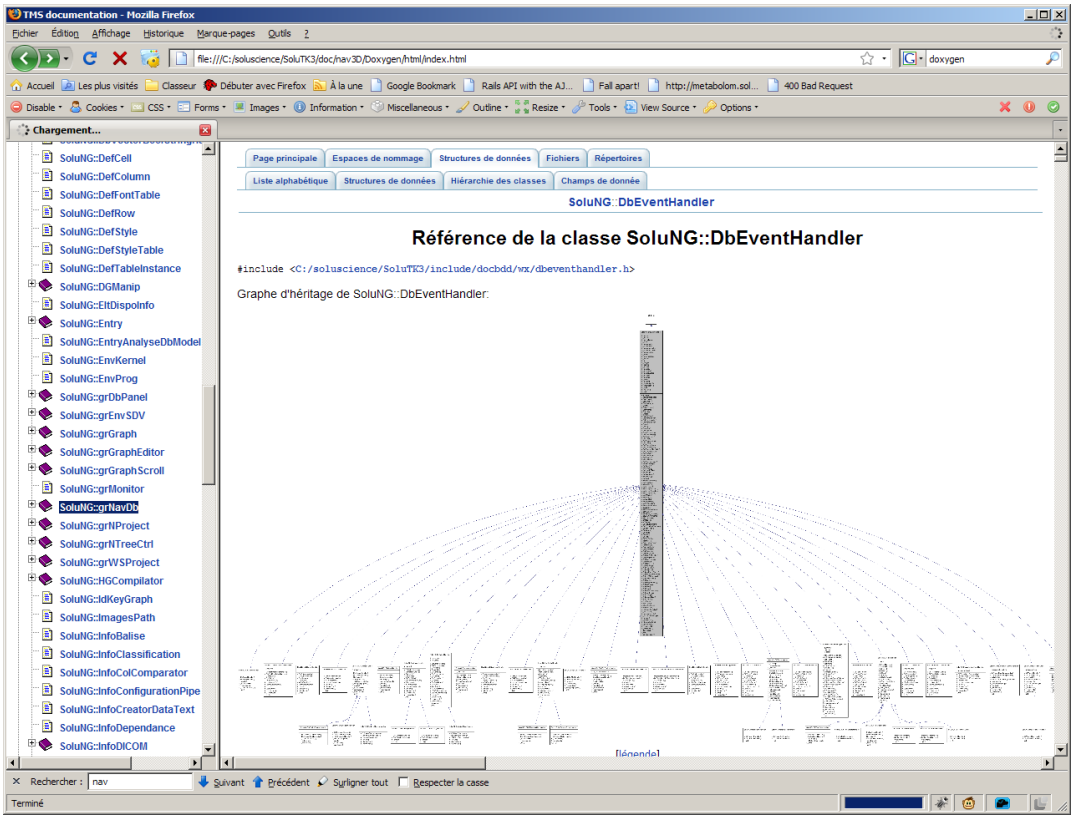

FIGURE 5.7 – Extrait du graphique des dépendances généré par Doxygen entre les différentes classes du simulateur. Ici le graphe d'héritage des différents *handlers*

5.5 Conclusion

Dans le travail d'intégration et de développement logiciel que nous avons présenté, nous nous sommes attachés, dans notre contexte, à comparer dans un premier temps les interfaces utilisateurs, sans négliger les interfaces en ligne de commande (et donc non graphiques). Au sein de notre développement, la complémentarité des approches graphiques et textuelles a été préservée, notamment pour pouvoir lancer plusieurs simulations en mode de traitement par lots (*batch*). La perspective des services Web a été également envisagée et l'approche mixte (graphique / ligne de commande) autorise la mise en place de ce type de développement. Il conviendrait alors de l'encrer dans une démarche *Model Driven Web Engineering* (MDWE) [Koch et al., 2008]. De même, les pratiques professionnelles du développement de logiciel nous ont conduit à faire usage d'un gestionnaire de versions. Cet aspect est particulièrement développé dans notre manuscrit dans une perspective d'ingénierie des modèles. C'est plus précisément la gestion des bibliothèques tierces (*vendor branches*) que nous avons détaillées car elles sont à l'origine de nombreux problèmes d'évolutivité et de portabilité.

L'intégration de logiciel a constitué une part très importante du travail de cette thèse. Notamment, nous avons regroupé au sein d'une même application l'ensemble des fonctionnalités nécessaires à la modélisation d'une séance de Stimulation Magnétique Transcrânienne, à savoir : modélisation de la bobine (simple ou double), le positionnement de la bobine, la modélisation de la tête du patient et son placement relatif par rapport à la bobine et enfin la visualisation en trois dimensions des résultats.

Les premiers résultats que nous avons pu présenter avec le logiciel complet nous permettent d'apprécier les effets de la Stimulation Magnétique Transcrânienne en fonction de la position et de l'orientation de la bobine sur une reconstruction en 3 dimensions du cortex d'un patient. Nous sommes en mesure avec ce logiciel de présenter des coupes orthogonales d'images IRM (toutes tridimensionnelles). Avec cette base logicielle, nous pensons également être en mesure de suivre les effets de stimulation se diffusant dans la tête du patient.

L'expérience acquise lors de ce développement, nous a conduit à une réflexion plus poussée sur l'importance de la plateforme retenue pour le logiciel final dans un contexte d'ingénierie des modèles. Nous allons dans le chapitre suivant étudier la relation modèle / méta-modèle pour la gestion des versions, les tests ainsi que les cadriciels de correspondance entre classes et tables relationnelles (couramment appelées ORM pour *Object Relational Mapping*).

Chapitre 6

Modèles et Méta-modèles pour la refactorisation (*refactoring)*

Le développement du progiciel de visualisation 3D pour l'utilisation de la SMT s'effectue avec la contrainte de l'obtention d'un outil exploitable à court terme. Nous avons vu que pour cela il a été nécessaire d'intégrer des composants logiciels patrimoniaux, au détriment de la modularité et de l'évolutivité du produit final obtenu. En nous fondant sur une démarche de rétro-ingénierie adossée aux concepts de l'IDM, nous proposons une analyse des inconvénients majeurs sur la base d'un travail de refactorisation. Nous présentons les modèles et les méta-modèles préalables indispensables à ce travail de refactorisation ainsi que les techniques qu'il convient de mettre en œuvre afin de le mener à bien dans ce chapitre.

Dans une première partie, nous présentons les limites de la refactorisation du code et nous exposons la démarche de refactorisation qui consiste à utiliser conjointement les techniques de génération dynamique de code et les techniques de gestion des versions. Nous détaillons plus avant cette démarche dans une deuxième partie. Dans une troisième partie, nous présentons la méta-programmation, une variante à la génération dynamique de code. Nous présentons les avantages et les inconvénients de son utilisation. Dans une quatrième partie, nous détaillons le modèle source de l'application. Nous expliquons comment il convient de gérer ces différentes versions ainsi que les limites de la technique mise en œuvre. Dans une cinquième partie, nous expliquons comment décorer les codes objets de l'application via le pattern décorateur afin de limiter les lignes de code de l'application. Dans une sixième par-

tie, nous expliquons la façon dont il convient d'effectuer la persistance des objets quand les composants logiciels le nécessitent comme cela a été le cas au cours du développement de l'application. Enfin, dans une dernière partie nous concluons sur l'utilisation d'un ORM dans le cadre de l'IDM.

6.1 Génération de code et gestion de versions

Comme nous l'avons expliqué dans le chapitre 2, nous considérons la génération de code comme une transformation, si on se réfère aux concepts de l'IDM. En programmation orientée objet, en se fondant sur ce principe, de nombreux environnements de développement intégrés (IDE) proposent de générer des classes à partir d'assistants. Ces derniers sont conçus sur la base de boîtes de dialogues qui permettent généralement de saisir les caractéristiques des objets manipulés sous forme graphique (classes, modèles, héritage, etc). À partir de cette description, il est alors possible de générer les fichiers correspondants, pourvus du code informatique qui décrit les classes du modèle considéré (*cf* Figure 6.1).

Certains IDE utilisent des schémas UML conçus de manière graphique par l'utilisateur et génèrent les classes correspondantes munies des attributs, des prototypes des méthodes et des accesseurs de base. Il est alors possible de revenir sur les déclarations et d'ajuster le code des classes en fonction des besoins. Par exemple, en *Ruby On Rails* (framework de développement d'application Web en Ruby), on utilise des scripts en langage ruby afin de générer les classes d'un modèle via des templates (gabarits). Il est également possible de développer des générateurs ad hoc en modifiant les générateurs initiaux.

6.1.1 Limitations liées à l'utilisation des générateurs de code

Il existe cependant des limitations à l'utilisation des générateurs de code. Comme pour les compilateurs que l'on retrouve en fin de chaîne, les générateurs de code se limitent encore trop souvent à un petit nombre de plateformes. Ainsi, la plupart des IDE génèrent du code en langage Java et parfois en langage C++. En effet, les

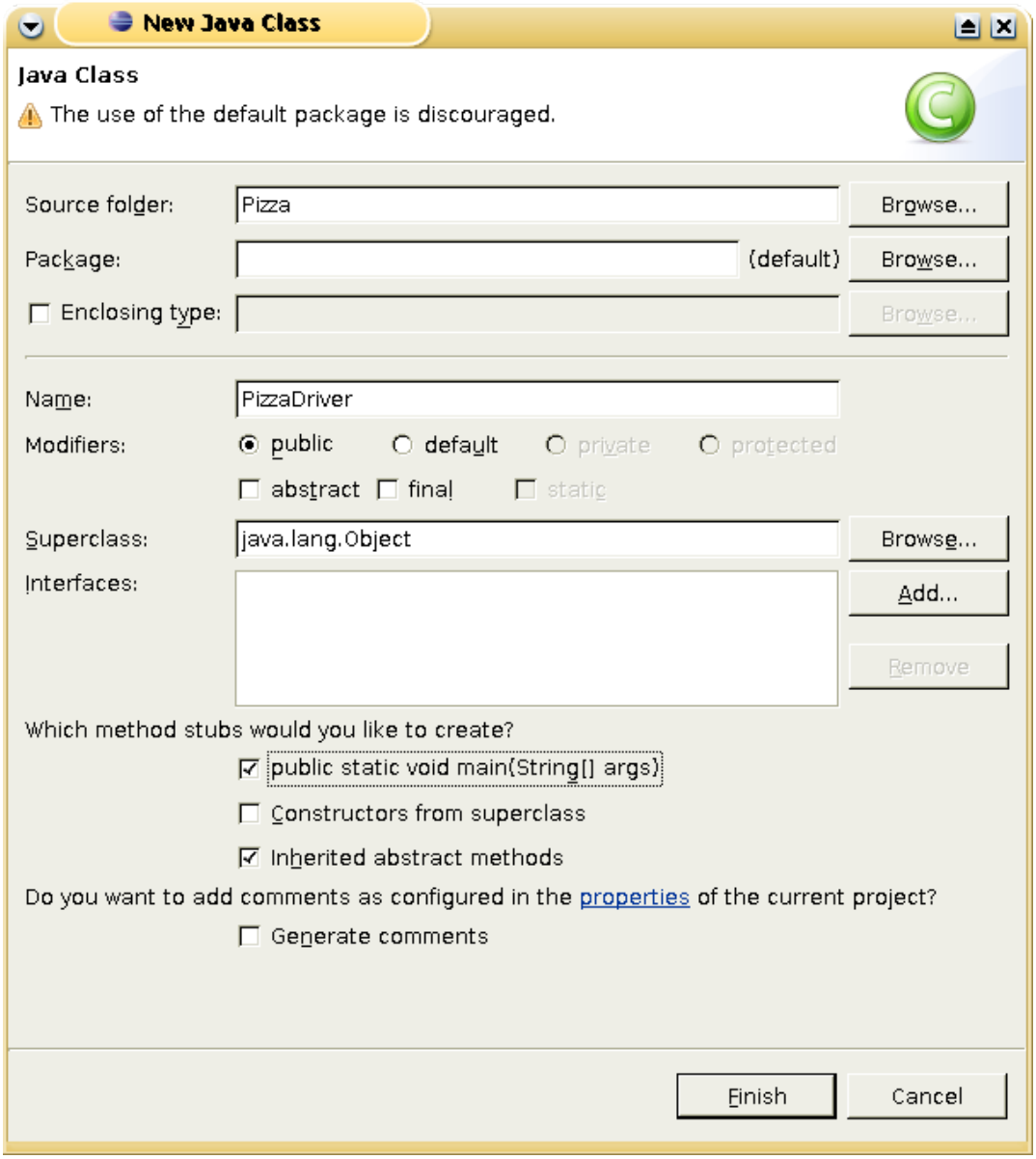

FIGURE 6.1 – L'assistant de création de classe d'Eclipse permet de définir quelques unes des caractéristiques de la classe et l'assistant génère le code correspondant

générateurs qui appartiennent souvent à des IDE se limitent au langage soutenant l'environnement de développement (exemple : JDev, GreenFoot).

La génération de code présente également des inconvénients. Dans le cas où il est nécessaire de modifier le générateur et ainsi le contenu du fichier généré. Le problème est lié au fait que la génération de code est proche du principe des opérations

informatiques dites de « copier-coller ». En conséquence, la génération de code peut propager en de nombreux composants des portions de code qui nécessiteront une refonte ultérieure. Ainsi chez Soluscience nous avons pendant longtemps utilisé un générateur de code pour générer les classes de composants d'interface graphique. Tous les objets graphiques dérivaient doublement de deux classes. Tout fonctionnait pour le mieux jusqu'au jour où nous avons découvert que dans certains cas et pour certains compilateurs, l'ordre de dérivation était important. Il a donc fallu reprendre l'ensemble des fichiers générés (environ 400) pour faire l'inversion de l'ordre de dérivation *cf.* Fig 6.2 et 6.3

```
1 class wxWColor : public grWColor, public wxAngie{
2 ...
3 };
```

FIGURE 6.2 – Déclaration du composant graphique wxWColor obtenu par le générateur

```
1 class wxWColor : public wxAngie, public grWColor{
2 ...
3 };
```

FIGURE 6.3 – Déclaration du composant graphique wxWColor telle qu'il a fallu la reprendre. Il a fallu procéder à l'inversion pour tous les autres composants graphiques

6.1.2 Limitation liée au contenu des modèles générés

Si les générateurs de code utilisés sont peu ou pas spécifiques, les descriptions effectuées en amont ne permettent pas de décrire précisément le code objet qui en résulte. La plupart du temps, le code généré constitue une matrice de départ à partir de laquelle il convient de compléter le code en se fondant sur la logique métier. Cette tâche reste à la charge du développeur.

C'est par exemple le cas d'une multitude d'environnements graphiques (MFC, VisualBasic, QT designer), avec lesquels on construit l'interface graphique de l'application via des boîtes à outils de composants (boutons, champs de saisie, table,

etc). Le travail du développeur consiste alors à compléter le code des méthodes qui réagissent aux évènements de l'utilisateur, ce qui permet de définir le comportement de l'application.

Supposons que l'on dispose d'un squelette d'application généré à partir d'une description graphique. Nous désirons modifier l'un des accesseurs afin de modéliser plus précisément la logique métier de l'application. Il convient alors de se poser la question de savoir quel est le moyen le plus efficace d'effectuer cette modification. Deux possibilités s'offrent à nous :

- utiliser l'éditeur de code afin d'ajouter de manière manuelle les attributs et les méthodes supplémentaires,
- utiliser l'éditeur graphique UML afin de décrire l'ajout puis regénérer le code.

Que ce soit en terme de rapidité et de pénibilité, l'utilisation de l'éditeur graphique apparaît comme être la méthode la plus efficiente. Remarquons à ce titre que si le générateur de code n'est pas suffisamment intelligent pour repérer les lignes de code qui ont été modifiées de manière manuelle depuis une précédente génération, celles-ci seront perdues et devront être de nouveau réécrites.

Afin de répondre de manière pragmatique à la problématique de ce travail et au regard des observations précédentes nous proposons de fonder la démarche de refactorisation envisagée sur une méthode qui allie le principe de la génération de code, de la gestion des versions et d'un IDE graphique dédié [Cook and Kent, 2008].

6.1.3 Description de la démarche de refactorisation du code

La démarche de refactorisation proposée se fonde sur l'utilisation conjointe de générateurs de code et d'un gestionnaire de versions. Pour cela, il convient de gérer tout fichier généré à l'aide d'un gestionnaire de version. Il est aussi important de prendre bien soin de faire une première révision avec uniquement les fichiers générés. Ainsi les modifications manuelles effectuées par la suite sur le fichier généré feront l'objet de *commits* indépendants. L'ensemble des révisions obtenues pourront alors être vues comme un ensemble de transformations qui pourront être ré-appliquées sur le fichier initial ou des versions dérivées (d'où l'importance de posséder une version du fichier initial dans le gestionnaire de versions).

Au moment du refactoring et juste avant de ré-générer les fichiers suite à la modification du document maître (le diagramme UML dans le cas précédemment cité), il suffira alors de s'assurer que les fichiers qui seront écrasés ne contiennent aucune modification locale (non envoyée sur le serveur de version) et de faire une ré-génération brutale des fichiers. Il suffira alors de comparer chacun des fichiers modifiés par la génération par rapport à la version contenant les derniers ajouts manuels. Un utilitaire de comparaison permettra alors de récupérer les modifications manuelles (métier) tout en conservant les nouvelles parties de code générées [1].

Le fait de tisser la génération de code avec un système de gestion de versions qui sont tous les deux des méta-modèles permet de tirer bénéfice de chacun d'eux. Il peut être tout aussi bénéfique de maintenir les différentes versions de code généré dans une branche à part entière. Cela revient alors à gérer le code généré comme n'importe quel *vendor branch* [2].

Le code courant est alors considéré comme la branche principale (*trunk*) liée au code généré. Il est alors possible d'appliquer simplement les modifications entre deux générations de code sur le trunk pour obtenir le code résultat *cf* Fig 6.4. Une autre solution peut aussi être de repartir à chaque nouvelle génération du code généré et de rejouer dessus l'ensemble des opérations de personnification *cf* Fig 6.5

L'avantage principal de cette méthodologie est qu'il n'est a priori plus nécessaire d'effectuer la fusion à la main comme proposé au paragraphe précédent. On retrouve la vision du développement comme l'application de nombreuses transformations sur un modèle *code source* via le méta-modèle offert par le gestionnaire de versions.

6.2 La méta-programmation

6.2.1 Principe de la méta-programmation

Quand la génération dynamique de code perturbe les processus de liaisons du code et conduit à des conflits entre les différents composants logiciels, il peut être intéressant d'utiliser une approche qui se fonde sur la méta-programmation. Il s'agit

1. voir annexe (6.6 page 150) sur TortoiseSVN et WinMerge
2. voir la partie 5.1.3.1 page 75 du chapitre 5

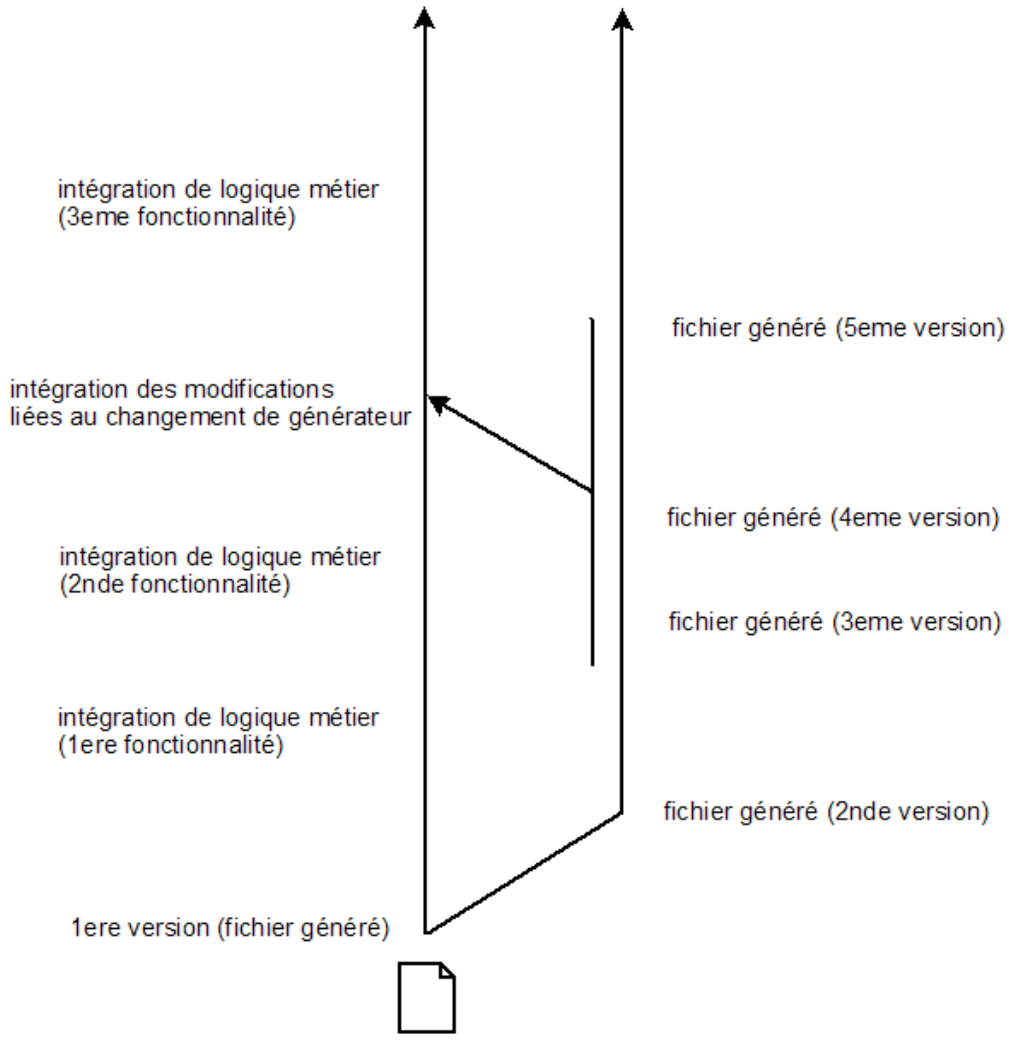

FIGURE 6.4 – Gestion de code généré sous forme de *vendor branch*

de retarder la génération effective du code des composants :

- dans le cas d'un langage compilé comme le C++, cela consiste à retarder la génération des fichiers sources à la phase de précompilation où un ensemble de macros et de templates dérouleront une simple ligne de code en une déclaration beaucoup plus longue qu'il ne restera plus qu'à compiler (*cf.* Fig 6.6).
- dans le cas d'un langage interprété comme Ruby, il s'agit de la possibilité dans une classe de générer au moment de son évaluation un ensemble de méthodes dont le squelette est identique et d'évaluer en temps réel ce code généré afin de rajouter ces méthodes à la classe (*cf.* Fig 6.7 et 6.8).

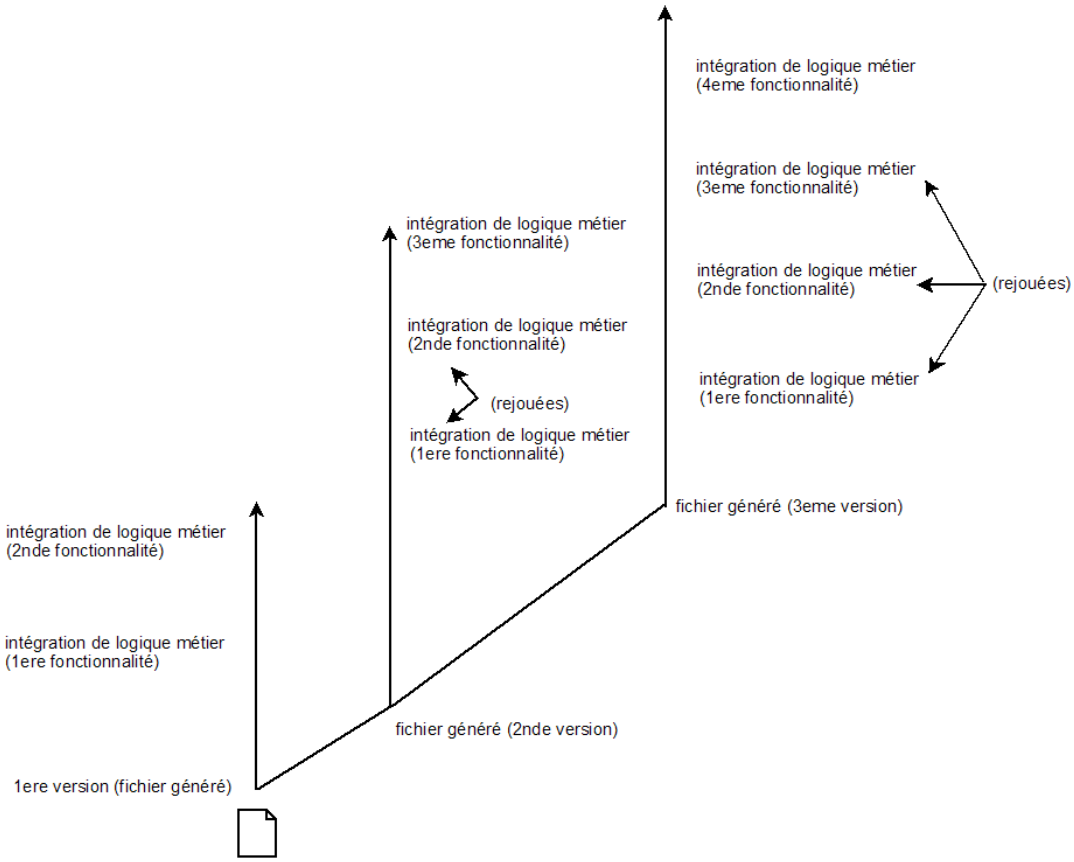

FIGURE 6.5 – Rejouer les personnalisations de code sur chaque fichier nouvellement généré

```
1 template<unsigned long N,typename T> inline Fact()
2 {
3   return (N==0) ? 1 : N*Fact<((N==0)?1:N-1),T>();
4 }
5 unsigned int x = Fact<5,unsigned int>();//5!
```

FIGURE 6.6 – Initialisation d'une variable par le calcul d'une factorielle. Le calcul n'est pas effectué à l'exécution, mais bien à la pré-compilation. Lors de cette étape, il y a déroulement (*unrolling*) récursif des instructions. Exemple de code issu de la mise en pratique du cours d'objet avancé de l'ISIMA

Si on souhaite changer le contenu de chacune des méthodes générées, il convient de le faire une seule et unique fois dans le code du *template*. La méta-programmation apporte également son lot d'avantages et d'inconvénients qui dépend de la nature du langage : compilé ou interprété

```ruby
module ArCBase
  ActiveRecord::Base.class_variables.each do |cv|
    cv = cv.gsub('@@', '')
    module_eval <<-EOF
      def #{cv}
        ActiveRecord::Base.#{cv}
      end
    EOF
  end
end
```

FIGURE 6.7 – A l'évaluation du module **ArCBase**, une méthode d'accès lui sera ajoutée pour chaque variable de classe de ActiveRecord : :Base. Le listing 6.8 donne le code équivalent après évaluation

```ruby
module ArCBase
  def connection
    ActiveRecord::Base.connection
  end
  def primary_key_prefix_type
    ActiveRecord::Base.primary_key_prefix_type
  end
  def reset_locking_column
    ActiveRecord::Base.reset_locking_column
  end
  #...
end
```

FIGURE 6.8 – Ce listing donne le code équivalent après évaluation du listing 6.7

6.2.2 Avantages et inconvénients liés à l'utilisation de la méta-programmation

Dans le cas d'un langage compilé comme le C++ le temps de compilation et plus particulièrement de précompilation va être fortement augmenté et peut devenir rédhibitoire si le compilateur ne peut pas gérer de fichiers d'entête pré-compilés. L'avantage est, qu'une fois la phase de compilation effectuée, le code sera aussi rapide que du code généré avant compilation.

Dans le cas d'un langage interprété comme Ruby, la problématique est toute

autre puisqu'en l'absence de précompilation/compilation tout va se faire à l'exécution. L'exécution peut alors en être fortement ralentie. On notera cependant que ce ralentissement n'a lieu qu'au moment de l'utilisation du code méta-programmé pour peu qu'un mécanisme d'évaluation paresseuse (*lazy evaluation*) ait été mis en place. Ce ralentissement peut aussi être limité à la seule première évaluation du code générique, dans le cas d'une machine virtuelle stockant en mémoire les classes étendues dynamiquement.

Notons aussi que les performances peuvent être améliorées par la machine virtuelle qui peut pour les fonctions méta-programmées ne plus faire une simple évaluation mais de la compilation Just In Time (JIT). Le principe est simple : les instructions les plus souvent utilisées, sont compilées en natif pour la plateforme sur laquelle est déployée l'application. En terme d'IDM, il y a pendant l'utilisation du logiciel, le passage d'une plateforme à une autre plateforme (d'un niveau d'abstraction plus proche de la machine) via le compilateur JIT.

Enfin, que le langage utilisé soit compilé ou interprété, la méta-programmation apporte un confort d'utilisation puisque le code source se limite à l'utilisation d'un ensemble réduit de directives que l'on peut assimiler à un DSL (Domain Specific Language) appartenant à une logique métier particulière ou plus générale. On peut alors envisager de spécifier l'ensemble des méta-modèles via un DSL ([Jouault and Bezivin, 2006])

Il faut cependant noter qu'il peut être très difficile de gérer un programme reposant en grande partie sur la méta-programmation. En effet, les lignes de code peut être alors être des évaluations d'évaluations et être difficilement analysables avec un *debugger* . Dans le tableau ci-dessous, nous résumons les avantages et les inconvénients liés à l'utilisation de la méta-programmation.

6.2.3 Conclusion sur l'utilisation de la méta-programmation

Il est donc difficile de faire des généralités sur le moment le plus opportun pour faire la génération de code, car il y a un compromis à trouver entre la rapidité de développement et les problèmes de performance qui peuvent survenir par la suite. Au

Type de langage utilisant la méta-programmation	Avantages	Inconvénients
Langage compilé	- une fois la phase de compilation effectuée, le code est rapide. - possibilité d'utiliser des techniques d'*unrolling* - l'utilisation d'un ensemble réduit de directives	- le temps de compilation et de précompilation va être fortement augmenté - les lignes de code sont fortement imbriquées
Langage interprété	- l'utilisation d'un ensemble réduit de directives	- l'exécution est fortement ralentie - les lignes de code sont fortement imbriquées

FIGURE 6.9 – Avantages et inconvénients liés à l'utilisation de la méta-programmation.

regard de ce qui a été énoncé précédemment, il apparaît clairement que la transformation qui consiste à traduire des directives de méta-programmation en code source est plus facile à mettre en œuvre que la transformation inverse qui vise à traduire du code informatique en directives de méta-programmation (modèle).

La transformation repose alors sur un outil logiciel de type compilateur/interpréteur qui cache physiquement le code en devenir dans des fichiers physiques. Ces fichiers, au même titre que le cache d'un serveur WEB, n'ont pas vocation à être modifiés mais sont très utiles lors de la phase de développement, notamment pour le débogage. L'opération inverse consiste en une phase de rétro-ingénierie sur un ensemble de codes (modèles). Si par la force des choses les générateurs ont évolué au fil du temps, le code est alors devenu du code patrimonial (Legacy code).

6.3 Le modèle source de l'application

Rappelons que l'objectif de l'IDM est de faire en sorte que le code source du progiciel ne soit plus le seul et unique artefact de la conception logicielle qui permette la mise en production. Pour cela, il faut que le code source de l'application soit remplacé par un modèle source (fichiers de classes métier).

6.3.1 Le modèle source de l'application

Le modèle source de l'application comprend les éléments à partir desquels on génère le modèle exécutable de l'application. Dans le cadre de la présente étude, la démarche générative adoptée précédemment est la solution la plus productive. En effet, la fin du processus de développement étant proche, il faut maintenant que le code source soit placé en production sous la forme d'un exécutable que l'on obtient à partir de ce modèle. Le fait de décrire le code source à cet instant via un modèle source permet la compilation quelque soit la plateforme informatique cible. La démarche entreprise est ainsi pragmatique et répond efficacement à la problématique de la présente étude. Un exemple de modèle source d'application est présenté sur la figure 6.10.

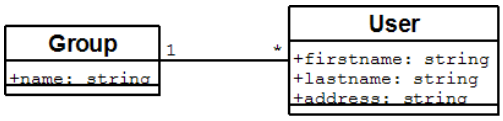

FIGURE 6.10 – Exemple de Diagramme UML source d'application (dessiné via Dia UML). Un utilisateur appartient à un groupe, un groupe est composé de plusieurs utilisateurs

Notons que, bien que le modèle source remplace le code en tant qu'élément central et point d'entrée de cette conception logicielle, nous manquons encore d'un outillage logiciel adéquate. Le code source en tant que tel s'est vu adjoindre un ensemble d'outils ad hoc au fil du temps, mais il reste à outiller la gestion des modèles et des méta-modèles de l'application. L'élément central de l'application repose maintenant sur un diagramme UML, il convient d'éviter que cela le rende dépendant de l'outil de conception logicielle qui a permis de le créer. C'est la norme XMI qui assure en principe la portabilité des modèles UML. Cependant, comment doit-on gérer les différentes versions possibles du document XMI maître ? Peut-on facilement travailler à plusieurs sur le projet et fusionner de nouveau les modifications qui ont été effectuées ?

6.3.2 Gestion des versions du modèle source de l'application - les fichiers de classes métiers

Dans le framework *Ruby on Rails*, la génération d'une classe métier utilisant l'ORM ActiveRecord se fait simplement via une simple commande en ligne *cf* 6.11. En nous fondant sur cette technique, il convient de regrouper l'ensemble des commandes permettant de générer les fichiers de classes métiers dans un seul et unique fichier (script schell). La mise sous gestionnaire de versions de celui-ci ne pose alors pas de problème. L'évolution des classes métiers, comme par exemple l'ajout d'un attribut, se fait par la simple modification du script global de génération puis une nouvelle exécution de tout ou partie de celui-ci.

Notons ici que ce fichier est lui-même re-générable par une transformation qui prendrait en entrée le fichier Dia-UML et le traduirait en directive de génération.

```
ruby script/generate model User firstname:string lastname:string \
  group_id:integer :address:string
      exists app/models/
      exists test/unit/
      exists test/fixtures/
      create app/models/user.rb
      create test/unit/user_test.rb
      create test/fixtures/users.yml
      exists db/migrate
      create db/migrate/20090326180125_create_users.rb
ruby script/generate model Group name:string
      exists app/models/
      exists test/unit/
      exists test/fixtures/
      create app/models/group.rb
      create test/unit/group_test.rb
      create test/fixtures/groups.yml
      exists db/migrate
      create db/migrate/20090326180146_create_groups.rb
```

FIGURE 6.11 – Commande de génération et fichiers générés : Les fichiers générés sont mis sous gestionnaire de versions ainsi qu'un fichier regroupant les deux lignes de commande

En utilisant cette technique, il faudra cependant être attentif à ce que :

– les générations successives des classes ne suppriment pas la logique métier qui a pu être ajoutée aux différentes classes. Mais nous avons déjà expliqué précédemment comment la gestion de versions résolvait ce type de problème de façon plus ou moins automatique ;
– le schéma de la base de données doit lui aussi être versionné. En effet, le code source de la classe n'est plus le seul artefact généré par le script. L'utilisation d'un système de correspondance objet-relationnel (en anglais *Object Relational Mapping* ou ORM) crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle. En terme IDM, le schéma d'une table qui stocke les instances d'une classe est bien le méta-modèle des différents enregistrements. Il convient donc de mettre le schéma global (ensemble de méta-modèles) sous gestionnaire de versions. Pour cela, nous pouvons exporter le schéma de la base sous la forme d'un script de génération (qui pourrait aussi bien être un fichier SQL ou XML) qui peut ainsi être mis sous un gestionnaire de versions (*cf.* Fig 6.12).

```
1  create_table "groups", :force => true do |t|
2    t.string  "name"
3    t.datetime "created_at"
4    t.datetime "updated_at"
5  end
6
7  create_table "users", :force => true do |t|
8    t.string  "firstname"
9    t.string  "lastname"
10   t.string  "address"
11   t.integer "group_id"
12   t.datetime "created_at"
13   t.datetime "updated_at"
14 end
```

FIGURE 6.12 – Script de migration global de la base de données

Cependant la simple utilisation d'un script global de création de l'ensemble des tables n'est pas suffisant. S'il est possible en phase de développement de supprimer des tables et de les re-créer avec des champs manquants, cette démarche ne peut

pas être utilisée lorsque l'application est en production. En effet, un rechargement du schéma conduit à la suppression de toutes les données de la base. C'est la raison pour laquelle nous préférons mettre en place un ensemble de scripts de migration qui décrivent les évolutions étape par étape du schéma de la base de données.

```
1 rename_column "users", "firstname", "first_name"
2 rename_column "users", "firstname", "last_name"
3 change_column "users", "address", "string", "text"
```

FIGURE 6.13 – Exemple de script de migration : les champs de table **users** qui correspondent aux attributs de la classe **User** sont renommés

En terme d'IDM, il s'agit d'une transformation de méta-modèles. Il y a aussi modification de l'ensemble des modèles associés au méta-modèle **User** puisque qu'il y a modification du type du champ adresse. Le moteur de base de données effectuera donc pour nous la modification de toutes les instances des modèles liés au méta-modèle qui vient d'être modifié.

6.3.2.1 Conclusion sur la gestion des versions des fichiers de classes métiers

Notons que si cette solution semble élégante en production, elle peut devenir une véritable gageure en conception lors du développement. Tant que le schéma global n'est pas à peu près défini, on lui préfèrera plutôt des rechargements globaux.

Le script contenant les paramètres de génération n'est plus ici le document maître or celui-ci peut cependant être très utile dans certaines circonstances. Son maintien assure par exemple en théorie l'existence d'une génération unique pour chacun des modèles permettant de saisir rapidement le schéma d'une table relationnelle. Sans cela, il faut se référer à la structure globale de la base qui peut être illisible si elle devient trop volumineuse.

Notons qu'il existe aussi un effet pervers à l'utilisation des générations s'il est possible de travailler à plusieurs sur un même méga-modèle (ensemble de modèles/méta-modèles). En effet, s'il est facile de mettre en place un ensemble de générations locales, il devient très difficile de partager ses migrations avec d'autres développeurs.

Le fait de numéroter les générations de 1 à n amène forcément des conflits si deux développeurs exécutent une génération portant le même numéro. Tout comme la gestion multi-utilisateurs d'un document XMI (ou ensemble de documents XMI) global pour une équipe, il reste ici des problèmes à résoudre.

6.3.2.2 Réduction du code source de l'application

Il est possible de réduire considérablement le code source de l'application et donc le nombre de lignes de code à maintenir. Cela peut se faire si le code source est défini par un ensemble de méthodes suffisamment génériques. Le développeur final n'a alors plus qu'à écrire du code déclaratif qui correspond à la description d'une configuration, celle-ci peut se faire alors sous forme graphique via un schéma UML. On prendra bien soin de limiter le diagramme de façon à ce qu'il soit manipulable par une transformation dont le but sera d'obtenir un modèle exécutable sur une plateforme de niveau d'abstraction inférieur.

La diminution/disparition du code ne se fera en réalité que s'il est possible, d'un projet à un autre, d'être capable d'abstraire tout ce qui peut l'être. De cette façon, un nouveau projet profite de tout ce qui a été fait sur les autres. La minimisation du nombre de lignes de code se fait si on cumule :

- la méta-programmation,
- l'héritage,
- l'héritage multiple dans les langages où il est possible comme en C++,
- l'utilisation de *Mixin* (partage de méthodes par différentes classes) dans les langages qui le proposent (ou leur émulation dans les langages à interface et supportant la délégation comme en Java), ou l'extension de prototypes dans les langages supportant ce paradigme.

Ainsi, la mise en place d'une classe se fait en la déclarant sans implémenter de méthodes qui le sont dans d'autres classes ou modules et en tissant via la dérivation ou l'inclusion de *Mixin* au sein d'une classe les différents comportements auxquels elle doit répondre.

L'écriture de méthode au sein de la classe peut alors se limiter à la plus simple expression de la logique métier. On utilise pour cela des méthodes anonymes (fonctions

```
class User < ActiveRecord
  validates_presence_of :firstname,
    :if => Proc.new { |record| record.lastname.blank? }
```

FIGURE 6.14 – Décoration des codes de l'application via le patron décorateur.

```
class User < ActiveRecord
  belongs_to :group, class_name => 'Group', : foreign_key => 'group_id'
end
```

FIGURE 6.15 – Définition d'un modèle **User** et de sa relation avec le modèle **Group**

génériques) au code très court. Elles permettent de décorer (pattern décorateur) les codes génériques qui seront quant à eux placés dans le méta-modèle parent. Cette logique est illustrée sur la figure 6.14

Notons ici que la configuration d'une classe est d'autant plus facile, si la classe est elle-même un modèle et que chacune des déclarations utilisées, est en fait l'appel à une méthode (transformation) définie par le méta-modèle. Ceci peut rendre possible par exemple, pour peu que le langage le permette, d'utiliser des arguments par défaut et de mettre en place un jeu de conventions afin de diminuer le nombre de paramètres requis lors de l'appel de la fonction.

On retrouve ici le pattern *convention over configuration* utilisé par différents *frameworks* (*cf* [Olsen, 2008]). À l'aide de *Ruby on Rails*, la simple déclaration du script 6.15 permet de définir un modèle `User` qui a une relation '0-1' avec un modèle `Group`.

Il y a deux conventions dans la déclaration `belongs_to` :

– la relation se fait avec un objet de classe `Group` (inféré à partir de `:group`)

– la clé étrangère permettant de faire la relation est `group_id`.

Il est possible d'outrepasser les conventions en passant les valeurs souhaitées pour la classe et la clé étrangère.

Depuis la console de *Ruby on Rails*, il est possible d'écrire et de résumer le listing précédent à :

```
User = Class.new ActiveRecord ::Base
User.belongs_to :group
```

```
1 u = User.new
2 u.group = Group.new
```

FIGURE 6.16 – Utilisation de la méthode **group=** qui a été ajoutée par méta-programmation à la classe **User**

La première ligne montre bien que la classe `User` est une instance de la classe `Class` et qu'elle dérive de la classe `ActiveRecord ::Base`. La seconde ligne met en avant l'appel de la méthode `belongs_to` définie dans `ActiveRecord ::Base` sur l'objet `User` permettant de lui rajouter la relation `group`.

La déclaration `belongs_to` va rajouter un ensemble de méthodes d'instance à la classe `User` avec entre autre `group` et `group=`, la définition de la classe 'Group' n'étant pas décrite ici (cf Fig 6.16).

Ce dernier exemple montre bien comment sans définir aucun code de méthode, il est pourtant possible d'ajouter des méthodes à un modèle. L'intérêt de cette approche déclarative est la possibilité d'interagir avec les modèles métiers et de définir les relations d'héritage, 1-1, 1-n dynamiquement. En quelques lignes dans un interpréteur, il est ainsi possible de tester rapidement la pertinence d'un modèle métier et quand celui-ci est validé de le sérialiser. La sérialisation est à envisager ici sous la forme de génération/mise à jour des fichiers de classe de l'application. Le code ainsi généré est une donnée comme une autre qui est enregistrée dans un fichier source pour pouvoir être ré-utilisée plus tard.

Cette utilisation agile de la ligne de commande peut ainsi se substituer à la manipulation de diagramme UML où il faudrait utiliser de nombreux clics de souris pour mettre en place des boîtes (classes) et les relier entre elles. Mais ces manipulations n'ont d'intérêt que s'il existe un mécanisme (transformation) permettant de remonter du nouveau modèle (mémoire/fichier source mis à jour après sérialisation) vers une représentation UML.

De la même façon que nous avons pu précédemment générer une partie de l'application à partir de diagrammes UML, il est fondamental de pouvoir faire l'opération inverse. En effet, la visualisation d'un diagramme général du nouveau méga-modèle (ensemble des méta-modèles) sera toujours plus agréable que la lecture fastidieuse

de tous les fichiers de modèle de l'application. Pour ce faire, la solution actuellement retenue est la génération d'un fichier *ecore*. Il s'agit d'un type de fichier XMI retraçant les relations entre les différentes classes et qui en étant ouvert par des logiciels intégrant l'*Eclipse Modelling Framework* (EMF) permet de produire des diagrammes comme celui de la Fig 6.17. Il a été généré par le logiciel TopCased. Il est intéressant de noter que ce logiciel a lui même été développé en suivant une démarche IDM. Il s'agit de la méthode TopProcess [Garcia et al., 2009], elle-même basée sur les normes OCL [OMG, 2006] et SPEM [OMG, 2008] définies par l'OMG.

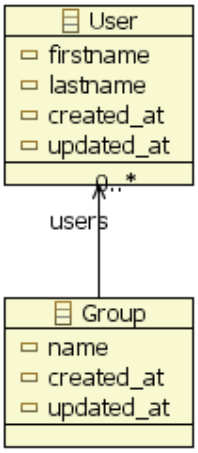

FIGURE 6.17 – Diagramme UML généré à partir d'un fichier XMI/Ecore avec le logiciel TopCased

La dernière étape qui n'est pas encore complètement finalisée est de repartir de ces fichiers UML pour régénérer l'application. Techniquement on pourra dans un premier temps s'inspirer de ce qui a déjà été mis en place pour la transformation des fichiers XML générés par le logiciel Dia en script de génération. Ainsi il pourra être envisagé de faire différentes itérations et de faire évoluer le logiciel en utilisant à chaque étape l'outil (graphique ou ligne de commande) le plus efficace pour faire évoluer le logiciel.

Il ne faut cependant pas se leurrer, comme déjà évoqué, il est fort probable qu'une grande partie de l'application doive encore pour un certain temps n'être décrite que par du code. En effet, il y a toujours des petites spécificités qui sont très difficilement généralisables. Pour les autres il sera toujours envisageable de les

décrire lors d'une première itération avec un langage de bas niveau avant de pouvoir grâce à la plateforme utilisée faire remonter l'information à un niveau d'abstraction supérieur. Ces allers et retours seront d'autant facilités si l'application est couverte par une suite de tests que nous allons maintenant développer.

6.4 Intérêt et mise en œuvre des tests unitaires

6.4.1 Principe des tests unitaires

Les tests unitaires servent à montrer que les structures et les fonctions développées réalisent bien ce que l'on attend d'elles. Dans le cadre de la refactorisation du simulateur de stimulation magnétique, il est nécessaire de mettre en place des tests pour le simulateur. Ceux-ci ne sont pas uniquement des programmes qui servent à vérifier qu'à la fin du projet le programme fonctionne correctement, ils ont surtout vocation à pouvoir être lancés sous la forme de scripts batch et tout au long de la vie du projet. Il convient alors de vérifier que l'application passe correctement tous les tests à chaque étape du développement afin de déceler les modifications de code qui conduisent à une exception ou à un résultat non attendu. Les tests forcent alors à corriger l'erreur dès qu'elle est introduite et diminuent le nombre potentiel de bugs.

Ils peuvent être lancés après une phase de développement pour faire des tests de non régression de l'application et vérifier que le comportement souhaité initialement est toujours le même dans la nouvelle version. Il est aussi possible de mettre en place un système qui vérifie en permanence les tests et les exécute en tâche de fond. Dès qu'un fichier source est modifié, les tests relatifs à ce fichier source peuvent être lancés automatiquement et donc signalent en permanence une dégradation éventuelle du code de l'application (*cf.* Fig 6.18). Quand les dépendances des tests deviennent trop importantes ou qu'une trop grande partie des ressources de la machine de développement se trouve être accaparée par les tests, un tel fonctionnement n'est plus possible. Il est alors possible de mettre en place une machine dédiée qui fait tourner les tests après chaque transaction effectuée par les développeurs. Ainsi, si l'un d'eux oublie de vérifier que tous les tests sont corrects, la machine dédiée pourra alerter les développeurs que des bugs potentiels ont été introduits.

```
Loaded suite -e
Started
.............................................................................
.......................................................F.....................
.............................................................................
.............................................................................
.............................................................................
.............................................................................
.............................................................................
.............................................................................
.................................................
Finished in 178.031 seconds.


1) Failure:
test_should_create_datum(DataControllerTest)
[./test/functional/data_controller_test.rb:31:in `test_should_create_datum'
c:/ruby/lib/ruby/gems/1.8/gems/activesupport-2.1.2/lib/active_support/testing/se
tup_and_teardown.rb:67:in `__send__'
c:/ruby/lib/ruby/gems/1.8/gems/activesupport-2.1.2/lib/active_support/testing/se
tup_and_teardown.rb:67:in `run']:
expected a redirect to <{"action"=>"edit", "id"=>"7", "controller"=>"data"}>, fo
und one to <{"action"=>"show", "id"=>"7", "controller"=>"data"}>, a difference o
f <{"action"=>"show"}>

687 tests, 1440 assertions, 1 failures, 0 errors
```

FIGURE 6.18 – Mise en place de tests sur une application : les points verts sont les tests qui sont passés, les rouges ceux qui ont échoué ou ont conduit à une erreur

6.4.2 Importance des tests unitaires lors de la démarche de refactorisation

En terme d'IDM, les tests correspondent à un méta-modèle du modèle application où la relation de conformité liant le modèle et le méta-modèle correspond à : « validité de l'application qui respecte le comportement souhaité ». C'est lors de la refactorisation que les tests se montrent les plus précieux. De nombreuses fois, pendant le développement du simulateur, la modification d'une partie de code pour le rendre objet introduisait une erreur d'exécution. La mise en place de tests permet d'éviter de générer ce type d'erreurs. Le remplacement progressif de code par les modèles prônés par l'IDM ne peut se faire que si cette évolution se fait sans détériorer l'existant. Il est donc important, avant de basculer l'architecture sous la forme d'une composition de modèles et de méta-modèles de mettre en place un système de tests unitaires. Dans la mesure où les tests peuvent être vus comme un méta-modèle simple de validation des comportements de l'application, un premier pas est

alors franchi. Les tests peuvent ensuite être maintenus pendant la mise en place et toujours utilisés avec la nouvelle architecture.

6.5 Persistance, sérialisation et Object-Relational Mapping

Une des difficultés rencontrée lors de l'élaboration des composants du progiciel est liée à la persistance des données. Au début de la mise en place du simulateur, tous les paramètres relatifs à l'orientation, au positionnement de la bobine et à la définition des différents paramètres physiques de la bobine, étaient implémentés dans le code de certains composants en C++. Aussi, après manipulation du système via le programme et l'obtention d'un positionnement intéressant de la bobine par rapport au cerveau, il est nécessaire de reporter les différents paramètres dans le code source. Sans ce report, le positionnement est perdu.

Cette méthode a pour avantage d'offrir la conservation de tous les paramètres utilisés tout au long du processus d'affinage via le système de gestion de versions mis en place au niveau du code. Elle garde en mémoire chacune des configurations pour peu que chaque configuration soit validée. Cependant cette méthode est lourde à mettre en œuvre et ne correspond pas à une utilisation permanente et routinière de l'outil.

La solution consiste à sauvegarder les paramètres dans un fichier de sauvegarde (sérialisation). Cela a été réalisé sur la branche principale (malheureusement incompatible avec la branche sur laquelle a été développé le simulateur). S'il est possible, via la part de reflexivité introduite dans le système, de sérialiser dans un fichier XML l'arbre complet des paramètres (*cf.* Fig 6.19), il n'en reste pas moins que la fluctuation relativement importante de la modélisation rend difficile la relecture des modèles une fois ceux-ci sérialisés.

Prenons en effet le cas de l'ajout d'une nouvelle transformation géométrique à un endroit quelconque de l'arbre. Il faut qu'à l'ouverture d'un fichier sérialisé ne comportant pas cet élément supplémentaire, le système de lecture soit capable de fournir des valeurs par défaut à la transformation. Si cela peut être géré pour l'ajout

d'une transformation, cela devient beaucoup plus délicat lorsque de nombreuses transformations succèdent et que l'on veut relire a posteriori ce qui a été mis en place à un moment donné. Nous proposons d'apporter une solution objet à cette problématique fondée sur l'Object Relational Mapping.

```
<?xml version="1.0" encoding="UTF-8"?>
<transformations type="array">
  <transformation>
    <axe-x type="float" nil="true"></axe-x>
    <axe-y type="float" nil="true"></axe-y>
    <axe-z type="float" nil="true"></axe-z>
    <created-at type="datetime">2009-03-26T20:17:12Z</created-at>
    <id type="integer">1</id>
    <rotation type="float" nil="true"></rotation>
    <scale type="float">3.0</scale>
    <translate-x type="float">-119.85</translate-x>
    <translate-y type="float">-119.85</translate-y>
    <translate-z type="float">-98.0</translate-z>
    <updated-at type="datetime">2009-03-26T20:17:12Z</updated-at>
  </transformation>
  <transformation>
    <axe-x type="float">0.0</axe-x>
    <axe-y type="float">0.0</axe-y>
    <axe-z type="float">1.0</axe-z>
    <created-at type="datetime">2009-03-26T20:20:00Z</created-at>
    <id type="integer">2</id>
    <rotation type="float">3.14</rotation>
    <scale type="float" nil="true"></scale>
    <translate-x type="float">91.0</translate-x>
    <translate-y type="float">41.0</translate-y>
    <translate-z type="float">87.0</translate-z>
    <updated-at type="datetime">2009-03-26T20:20:00Z</updated-at>
  </transformation>
</transformations>
```

FIGURE 6.19 – Sérialisation des paramètres des transformations géométriques

6.5.1 Object Relational Mapping

L'ORM est un système qui permet de faire correspondre le domaine Objet des composants développés au domaine relationnel des bases de données. Ainsi, nous

utilisons le patron de conception *ActiveRecord* (*cf.* 6.20)afin de fait correspondre un objet à un enregistrement dans la table d'une base de données.

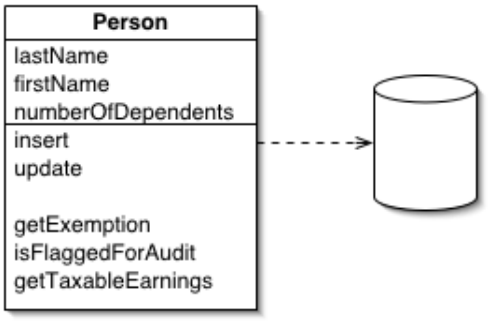

FIGURE 6.20 – Correspondance Objet-Relationnel avec ActiveRecord [Fowler, 2003], source de l'image : http ://martinfowler.com/eaaCatalog/activeRecord.html

A chaque attribut de la classe correspond une colonne de la base de données. On notera ainsi plusieurs implémentations de ce patron de conception : *ActiveRecord* pour *Ruby on Rails*, *Hibernate* pour J2EE. Il existe cependant d'autres patrons de conception de ce type offrant des interfaces légèrement différentes (*DataMapper*, *Sequel*).

Chaque objet manipulé est enregistré et sérialisé dans une base de données et peut être récupéré plus tard via un identifiant unique de type clé primaire. L'avantage de cette technique de persistance est que les modifications au niveau des attributs des différents modèles peut se faire directement via la base de données grâce à des requêtes SQL. Les modifications correspondantes sont appliquées en même temps sur chacun des enregistrements. Ainsi il n'est plus nécessaire de conserver pour chaque modèle le numéro de version du méta-modèle avec lequel il a été généré ainsi que l'ensemble des fonctions de transformation pour passer d'une version à une autre.

6.5.2 Avantages, inconvénients et limites d'un ORM dans le cas de la présente étude

Si le langage de programmation n'offre pas d'outils de sérialisation en natif, les possibilités d'introspection induite par le modèle relationnel permettent de mettre en place une exportation en XML des données ou de leur schéma. En effet, ce format

de données reste sûrement le meilleur moyen de communication, de partage et de transfert entre composants logiciels.

L'utilisation d'une base de données relationnelle comme système de persistance offre la possibilité de déléguer toute recherche sur le système à la base de données en utilisant un langage de requête de type 'SQL'. On peut alors sûrement objecter que l'espace technologique XML est doté du même genre de technologie via 'X-Query' ou 'XPath'. Mais sans un système d'indexation mis en place sur l'ensemble des fichiers, il est délicat de maintenir de bonnes performances quand les fichiers XML deviennent volumineux et très nombreux. L'indexation implicite de la base de données sur les clés primaires voire sur d'autres champs apparaît beaucoup plus facilement accessible.

Un autre avantage de cette dualité de modèle/méta-modèle est de pouvoir éventuellement faire évoluer ou remplacer l'un, pour peu qu'il continue d'utiliser l'interface proposée par l'autre. Ainsi, les systèmes permettant la saisie ou la consultation peuvent évoluer, le noyau dur restant accessible par la sur-couche ORM. Dans un système où les données sont le cœur du métier, cela est très important. On pourra objecter que dans le cadre de ce travail de doctorat, cela n'est peut être pas le cas. Effectivement, la manipulation de données volumineuses comme les images IRM ne se prête pas forcément à l'insertion dans une base de données. L'utilisation de BLOB peut être une solution mais leur utilisation a tendance à faire chuter les performances du système.

Par contre, l'utilisation conjointe d'une base de données dans laquelle sont stockées les dimensions des images, les paramètres utilisés pour la génération des images, et un système de fichiers sur lequel sont stockées les images peut être une bonne architecture.

Les inconvénients que l'on peut apporter à la généralisation d'un ORM comme système de persistance est l'utilisation qu'il aurait pu être faite d'une base de données objets. Cependant les technologies qui s'y rattachent sont inadaptées (LDAP) ou limitées (ZODB).

Il y a un autre point à considérer parmi les limitations d'un ORM. La dualité entre le monde des objets et le monde des SGBDR est un avantage au premier

abord, mais peut devenir un inconvénient quand il faut maîtriser les enjeux des deux espaces technologiques. Les problématiques transactionnelles notamment peuvent parfois prêter à confusion et être source d'incompréhension conduisant à des erreurs d'implémentation de la couche objet.

Ainsi les très grands systèmes comme Google n'utilisent plus de bases de données relationnelles standard pour la gestion des gros volumes d'information. Si pour certaines applications, la gestion des utilisateurs et de leurs préférences est faite par des bases de données relationnelles une grande partie du cœur de métier ne l'utilise pas. L'argument principal pour justifier le choix du système d'accès aux données est la prévalence du système à accéder aux données plutôt qu'à les écrire. Par exemple, un annuaire LDAP est beaucoup plus efficace qu'une base de données relationnelle avec la mise en place d'index.

Notons aussi qu'il est possible au niveau de l'ORM d'optimiser cette requête en dénormalisant le schéma de la base de données. Cela peut permettre d'obtenir des performances nettement meilleures, mais il s'en suit une dégradation de la cohérence des données. Celle-ci peut alors être maintenue par la mise en place d'un système asynchrone de vérification des données qui met à jour les index, calcule les variables compteur et les cache dans d'autres champs.

6.5.3 Conclusion sur l'utilisation d'un ORM dans le cadre de l'IDM

En terme d'IDM, un système d'ORM peut être vu comme un pont entre deux méta-modèles. Du côté relationnel, le méta-modèle est le schéma de la base de données qui va être mis en correspondance avec le méta-modèle dans le monde objet à savoir une classe. En terme de modèle, une instance de la classe sera mise en correspondance avec un enregistrement dans une ou plusieurs tables de la base de données. Ce pont permet aussi à la vision unificatrice de l'IDM de capitaliser sur les savoirs engrangés dans ces deux espaces technologiques.

Au-delà de l'utilisation du tout objet ou du tout modèle prôné par l'IDM, il semble encore difficile d'arriver à la mise en place d'un modèle ou d'un méta-modèle totalement productif. Ainsi, la mise en place à un moment ou à un autre d'un sys-

tème se fera par le choix ou la création d'une plateforme d'exécution. Par définition cette plateforme technologique sera caduque et nécessitera un remaniement dans les années qui suivront et devra probablement faire l'objet de rétro-ingénierie et de refactorisation.

L'évolution de code est délicate surtout s'il faut changer de technologie voire de langage. Mais dans le cadre d'une technologie à cheval entre deux mondes, on peut comme évoqué plus haut réussir à passer d'un code patrimonial à un modèle patrimonial. En effet, dans l'espace technologique *DbWare* le phénomène de *legacy Database* est déjà bien connu et a fait l'objet de nombreuses études [Ambler, 2003], [Gray et al., 1994]. Même si cela reste un problème épineux, il sera possible d'accélérer la refactorisation en profitant de l'expérience déjà acquise dans ce domaine.

6.6 Conclusion

Dans ce chapitre nous sommes revenus sur l'importance de la gestion des versions. Nous avons vu au chapitre précédent les concepts de *vendor branches* que nous avons mis en application ici pour la gestion de code généré. Nous montrons comment le tissage de la génération de code (une transformation au sens de l'IDM) avec la gestion de versions permet de limiter les inconvénients de la première. Leur utilisation conjointe permet d'envisager le développement de logiciel comme l'application de modifications (*patch*) automatisables. Cela met ainsi en avant un ensemble de Transformations sur le Modèle *code source de l'application*, concepts fondamentaux de l'Ingénierie Des Modèles qui apporte un cadre unificateur aux activités disparates du génie logiciel.

Dans une seconde partie nous avons proposé une alternative à la génération de code via l'utilisation de méta-programmation. Nous avons présenté une étude comparative des avantages et inconvénients de chacune de ces transformations au sens de l'IDM. Sans faire de généralités car tous les contextes de développement sont différents, il nous semble que la méta-programmation est une alternative à la génération de code plus pertinente quoique plus délicate à mettre en place. Dans un cadre pragmatique il sera probablement plus aisé de prototyper l'application

via un générateur de code puis de refactoriser le code généré en mettant en place les éléments de méta-programmation les plus pertinents. Le but de ce travail est avant tout de réussir à ramener l'application à son expression la plus simple et la plus facilement appréhendable. Nous montrons que le remplacement du code source par un modèle de plus haut niveau ne pourra se faire que s'il est possible d'inclure dans celui-ci la totalité des informations nécessaires à la description de l'application. S'il est encore difficile d'envisager l'ensemble du développement dans ce cadre, nous avons montré comment la partie métier de l'application peut faire l'objet d'une telle modélisation.

Nous présentons ensuite l'importance de la mise en place de tests pour permettre l'évolution et la maintenance de l'application. En effet, nous montrons comment les tests d'une application sont eux aussi un méta-modèle de celle-ci. L'utilisation conjointe de plusieurs méta-modèles permet alors d'envisager plus sereinement les modifications, refactorisations du modèle code source. Les tests de régression seront en effet là pour s'assurer que le fonctionnement de l'application n'est pas modifié. Il sera alors possible d'appliquer autant de modifications, de transformations sur le modèle de l'application pour abstraire et généraliser les composants développés sans pour autant introduire de régression dans le logiciel.

Nous finissons enfin ce chapitre, par la présentation des avantages apportés par l'introduction d'un autre méta-modèle au sein de l'application. Il s'agit de la mise en place d'un ORM pour la persistance des classes métiers de l'application. En établissant un pont entre le développement orienté objet de l'application et le domaine des bases de données relationnelles, on peut alors profiter des avantages du monde relationnel : persistance, sérialisation, partage d'informations avec d'autres composants logiciels. Il devient même possible d'envisager de changer complètement de plateforme en repartant du schéma de la base de données.

Conclusion

Dans ce manuscrit, nous avons présenté un travail réalisé dans un contexte thèse avec un contrat CIFRE. Après un travail initial de trois années sur la partie uniquement informatique médicale, nous n'avions pas abouti à un résultat satisfaisant en terme d'intégration logicielle. Nous avons par la suite pris du recul sur les développements informatiques et approfondi les notions unificatrices de l'ingénierie des modèles. C'est dans ce contexte que nous avons replacé notre travail de thèse qui traite de l'apport de la SMT par rapport à d'autres techniques de stimulation cérébrale plus douloureuse et/ou invasive.

Grâce à sa facilité de mise en œuvre, la SMT est un outil qui peut être utilisé en ambulatoire ce qui présente un avantage sur les autres techniques. Mais il reste encore beaucoup de chemin à parcourir avant de parvenir à comprendre finement les phénomènes impliqués lors d'une stimulation. Si à l'heure actuelle nous avons une compréhension macroscopique de ce qui se passe au moment de la stimulation, il est encore difficile de savoir quelles seront les zones du cerveau les plus sollicitées par la stimulation.

Nous avons réalisé une étude approfondie des techniques de modélisation et de calcul des effets de la stimulation magnétique transcrânienne, puis nous avons proposé une méthode de calcul qui repose sur une modélisation de la bobine par contour paramétrique avant d'implémenter un simulateur.

Le simulateur développé au cours de cette thèse est une bonne base pour pouvoir tester des hypothèses sur la forme, le positionnement d'une bobine par rapport à la tête du patient. Il permet de mieux appréhender les phénomènes. D'autre part, les préceptes de développement qui ont mené à sa réalisation devraient permettre de rajouter facilement des fonctionnalités.

Enfin, nous insistons sur les efforts faits sur la modularité du module de calcul, qui peut être facilement remplacé, notamment par un module plutôt orienté éléments finis pour prendre en compte les hétérogénéités du cerveau humain. Avec le recul pris dans notre travail de génie logiciel et en nous inspirant de l'ingénierie des modèles nous avons considéré les points suivants :

- les techniques d'appariement (*mapping*) entre base de données relationnelles et approche orientée objet (*Object Relational Mapping*),
- la gestion des tests unitaires, fonctionnels, d'intégration...,
- la génération de code,
- la gestion de versions des codes et des modèles.

Perspectives dans le cadre de la SMT et de l'imagerie médicale

D'un point de vue scientifique, des collaborations avec des spécialistes des EDP pourront contribuer à la résolution générique des équations de Maxwell. Ceci devrait permettre à terme de calculer la diffusion des différents champs dans le cerveau du patient en tenant compte des propriétés électromagnétiques de celui-ci. Il serait également opportun d'étudier la génération de maillage adapté au calcul d'éléments finis et l'association à ce maillage des propriétés électromagnétiques adéquates.

Les pistes de réflexions proposées sur ce dernier point sont :

- l'utilisation de tenseur de diffusion pour déterminer les différentes valeurs de la perméabilité magnétique et de la conductivité électrique des différents tissus. Cette étude sera à notre avis à mener en deux étapes, la première consistant uniquement à utiliser des valeurs constantes et scalaires pour un type de tissu donné avant d'introduire des valeurs anysotropes et non uniformes.
- Outre le maillage cubique inhérent à une image IRM, les maillages de la surface corticale générés par l'algorithme des *marching-cubes* pourraient être utilisés. En élargissant la génération de maillage à l'ensemble (ou un sous ensemble) des structures anatomiques cérébrales, il devrait être possible de générer un maillage 3D pour chacun des tissus.

Ces méthodes de générations de maillage pourraient être testées, il reste cependant un important travail d'ingénierie informatique pour inclure ces données dans la chaîne de traitement. Les principales difficultés seront la lecture des images DICOM puis l'intégration dans un outil commun des méthodes algorithmiques déjà développées pour d'autres projets et dans des cadres de programmation différents. A terme les algorithmes pourraient être mis à disposition sous la forme de service web.

Nous avons veillé tout au long de notre travail à mettre en place des outils génériques. Le travail de développement a été intégré avec d'autres composants issus de recherches pluridisciplinaires menées au sein de Soluscience. Notre démarche ne s'est ainsi pas limitée à la réalisation d'un outil ultra-spécifique mais à la mise en place d'un module de visualisation 3D utilisable par la société dans bien d'autres domaines scientifiques.

Avec le recul que nous avons maintenant, cette application a déjà pu être utilisée dans le cadre d'un autre projet au sein de Soluscience pour de la visualisation d'images de microscopie confoncale. Les microscopes confocaux Zeiss génèrent dans un premier temps un format propriétaire. Ce format permet de stocker les images par groupes d'expériences, par piles d'images en Z (image 3D) ou Z + t (évolution dans le temps). En plus de ces informations et au même titre que le DICOM, un ensemble de méta-données peuvent être stockées au sein du fichier (paramètres du microscope qui ont permis l'acquisition, fluorescence utilisée etc). Le microscope est aussi fourni avec un logiciel (également librement téléchargeable) qui permet de convertir les images (en perdant cependant une importante partie des méta-données) au format LSM qui se trouve être un format TIFF légèrement amélioré. Dans le cadre d'un prototypage rapide d'application et en prenant en point d'entrée les images converties au format TIFF, il a été mis en place une méthode de conversion de TIFF vers la structure de données utilisées en interne pour manipuler les matrices 3D. Grâce au travail de thèse, les images de microscopie confocale pouvaient alors être visualisées en trois dimensions via le Slicer3D. Il a même été possible de faire tourner certains des algorithmes intégrés au simulateur sur un jeu d'images standard (fourni avec les logiciels de reconstruction 3D vendus avec les appareils de microscopie confoncale). L'algorithme de reconstruction 3D par exemple a permis d'obtenir un résultat relativement proche de celui fourni par le logiciel de démonstration.

Perspectives dans le cadre de l'ingénierie des modèles

Il est difficile de s'exprimer avec certitude sur les éléments décrits au chapitre 6 qui seraient les plus pertinents pour améliorer le simulateur. Avec le recul, il ne nous est plus possible de concevoir une application qui manipule des données sans une

partie base de données et une couche d'ORM pour y accéder. Les spécificités de la visualisation 3D et la manipulation de gros volumes de données (maillage volumineux, image tridimensionnelle) peuvent néanmoins mettre à mal ce raisonnement. Le compromis que nous conseillerions actuellement serait de gérer les méta-données avec une base relationnelle et le stockage de données brutes serait laissé au gestionnaire de fichiers du système d'exploitation, permettant ainsi de tirer profit des meilleures avancées technologiques (accès disques via de la fibre optique *FiberChannel* ou encore les technologies de disques à base de mémoire Flash).

La politique de test que nous mettons actuellement en place pour le développement d'application Web s'intéresse surtout au fonctionnement côté serveur. Cette testabilité des communications entre le client et le serveur s'adapte très mal à la mise en place de tests d'interface utilisateur. Si nous devons effectuer une comparaison, les manipulations des paramètres 3D ne trouvent leur pendant qu'au niveau des pages « html » dynamiques dans lesquelles du code javascript va permettre de gérer l'évènementiel. A l'heure actuelle nous sommes loin de maîtriser totalement le processus de test de cette partie de l'application (limitation à quelques tests écrits avec un outil comme Selenium proposé par la société de Martin Fowler). La différence fondamentale est que la partie *rendu* et gestion du comportement est à la responsabilité du navigateur web et que celui-ci interprète du html et du javascript, alors que le navigateur 3D présenté ici affiche sa propre structure mémoire pour faire le rendu. Il n'y a pas de norme quant à la scène 3D à afficher ni sur la partie évènementielle qui peut être ajoutée et qui doit être écrite « en dur » dans le logiciel. Nous avions considéré l'utilisation de VRML (existant depuis la fin des années 1990) mais l'étude que nous avions menée ne nous avait pas convaincu.

Nous nous interrogeons aussi sur la possibilité d'effectuer un parallèle avec le format KML utilisé par Google dans GoogleEarth et SckechUp, pour visualiser des flux de données géographiques (picots, maillages de la surface de la terre, maillages de bâtiments, etc). En adoptant un tel format, le simulateur pourrait alors séparer la partie visualisation de la partie purement structurelle et informative. Il restera cependant à définir comment sera effectué l'appariement (mapping) bilatéral entre les données telles qu'elles sont manipulées par le simulateur et la partie visualisation

à proprement parler (conversion des données en éléments d'interface graphique) - cf. génération de xhtml ; modification des données après une modification depuis la zone d'affichage - cf. requête « http » et envoi du formulaire.

Plus généralement, si nous devions encadrer de futurs développements du simulateur, nous envisagerions une gestion de versions améliorée (*starmerge*) qui serait la solution au problème de fusion de branches. Il serait possible de travailler sur une branche personnelle pour ne pas être obligé de recompiler systématiquement l'ensemble de l'application suite aux modifications effectuées au jour le jour par le reste de l'équipe de développement. Par contre, nous pensons qu'il est important de s'assurer régulièrement des ré-intégrations de la branche principale dans la branche de fonctionnalité (*feature-branche*) dans laquelle une nouvelle fonctionnalité est développée. L'idéal serait de réussir à mettre en place une procédure d'intégration continue qui réalisera cette incorporation automatiquement et vérifiera la compilation de l'application avec la dernière version de la bibliothèque sur laquelle reposera le simulateur. Au-delà de la compilation, s'il y a une suite de tests en place, ce sera aussi l'occasion de les lancer pour vérifier que tout se passe convenablement.

A l'avenir, les autres éléments qui feront *a priori* partie de notre démarche de conception logicielle, sont le recours systématique à la mise sous gestionnaire de versions (quitte à ce que cela soit uniquement via un simple dépôt local) de tout élément de code, fichier (et donc modèle), partageable avec le reste des développeurs. Outre le fait de conserver les différentes versions, le plus important ici est d'utiliser le système de gestion de versions pour mettre en avant toute modification (et de fournir rapidement le contenu de cette modification) au niveau du modèle manipulé, afin de pouvoir contre-valider la modification/évolution du modèle/méta-modèle qui amène à la génération de la cible. De même, nous pensons qu'il faut considérer que tout élément qui peut se retrouver dans un cache doit être mis systématiquement sous gestionnaire de versions.

Bibliographie

[Ackerman, 1998] Ackerman, M. (1998). The Visible Human Project. *Proceedings of the IEEE*, 86(3) :504–511.

[Ambler, 2003] Ambler, S. (2003). *Agile Database Techniques : Effective Strategies for the Agile Software Developer*. John Wiley & Sons, Inc. New York, NY, USA.

[Arnold, 1993] Arnold, R. (1993). *Software reengineering*. IEEE Computer Society Press Los Alamitos, Calif.

[Arsonval, 1896] Arsonval, A. (1896). Dispositif pour la mesure des courants alternatifs de toutes fréquences. *C R Soc Biol (Paris)*, 3 :450–457.

[Atkinson and Kuhne, 2003] Atkinson, C. and Kuhne, T. (2003). Model-driven development : a metamodeling foundation. *Software, IEEE*, 20(5) :36–41.

[Bagci, 2009] Bagci, E. (2009). Reverse engineering applications for recovery of broken or worn parts and re-manufacturing : Three case studies. *Advances in Engineering Software*, 40(6) :407–418.

[Barker et al., 1985] Barker, A., Jalinous, R., and Freeston, I. (1985). Non-invasive magnetic stimulation of human motor cortex. *Lancet*, 1(8437) :1106–7.

[Barra and Boire, 2000] Barra, V. and Boire, J.-Y. (2000). Tissue segmentation on mr images of the brain by possibilistic clustering on a 3d wavelet representation. *Journal of Magnetic Resonance Imaging*, 11 :267–278.

[Basser, 1993] Basser, P. (1993). Cable equation for a myelinated axon derived from its microstructure. *Medical and Biological Engineering and Computing*, 31(1) :87–92.

[Basser et al., 1992] Basser, P., Wijesinghe, R., and Roth, B. (1992). The activating function for magnetic stimulation derived from a three-dimensional volume conductor model. *IEEE Trans Biomed Eng*, 39(11) :1207–10.

[Beck, 1999] Beck, K. (1999). *Extreme programming explained : embrace change*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

[Beck, 2003] Beck, K. (2003). *Test-driven Development : By Example*. Addison-Wesley Professional.

[Bestmann et al., 2004] Bestmann, S., Baudewig, J., Siebner, H., Rothwell, J., and Frahm, J. (2004). Functional MRI of the immediate impact of transcranial magnetic stimulation on cortical and subcortical motor circuits. *European Journal of Neuroscience*, 19(7) :1950–1962.

[Bézivin, 2004] Bézivin, J. (2004). In Search of a Basic Principle for Model Driven Engineering. *Novatica Journal, Special Issue, March-April*, 2 :21–24.

[Bezivin, 2005] Bezivin, J. (2005). On the unification power of models. *Software and Systems Modeling*, 4(2) :171–188.

[Bezivin et al., 2005] Bezivin, J., Bruneliere, H., Jouault, F., and Kurtev, I. (2005). Model Engineering Support for Tool Interoperability. *Proceedings of 4th Workshop in Software Model Engineering at 8th Int. Conf. on Model Driven Engineering Languages and Systems*.

[Bezivin et al., 2008] Bezivin, J., Vallecino-Moreno, A., Garci a Molina, J., and Rosi, G. (2008). MDA at the Age of Seven : Past, Present and Future. *UPGRADE, The European Journal for the Informatics Professional*, IX(2) :4–5.

[Bohning et al., 1997] Bohning, D., Pecheny, A., Epstein, C., Speer, A., Vincent, D., Dannels, W., and George, M. (1997). Mapping transcranial magnetic stimulation (tms) fields in vivo with mri. *Neuroreport*, 28(11) :2535–8.

[Bohning et al., 1999] Bohning, D., Shastri, A., McConnell, K., Nahas, Z., Lorberbaum, J., Roberts, D., Teneback, C., Vincent, D., and George, M. (1999). A combined TMS/fMRI study of intensity-dependent TMS over motor cortex. *Biological Psychiatry*, 45(4) :385–394.

[Branston and Tofts, 1991] Branston, N. and Tofts, P. (1991). Analysis of the distribution of currents induced by a changing magnetic field in a volume conductor. *Phys. Med. Biol.*, 36 :161–168.

[Brasil-Neto et al., 1992] Brasil-Neto, J., Cohen, L., Panizza, M., Nilsson, J., Roth, B., and Hallett, M. (1992). Optimal focal transcranial magnetic activation of the human motor cortex : effects of coil orientation, shape of the induced current pulse, and stimulus intensity. *J Clin Neurophysiol*, 9(1) :132–6.

[Bronzino, 1995] Bronzino, J. (1995). The Biomedical Engineering Handbook. *CRC Press, Inc, 2000 Corporate Blvd, NW, Boca Raton, FL 33431, USA, 1995. 2896.*

[Brun and Pierantonio, 2008] Brun, C. and Pierantonio, A. (2008). Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional*, IX(2) :29–33.

[Chikofsky and JH, 1990] Chikofsky, E. and JH, I. (1990). Reverse engineering and design recovery : a taxonomy. *Software, IEEE*, 7(1) :13–17.

[Cook and Kent, 2008] Cook, S. and Kent, S. (2008). The Domain-Specific IDE. *UPGRADE, The European Journal for the Informatics Professional*, IX(2) :17–21.

[Davey et al., 1997] Davey, N., Puri, B., Lewis, H., Lewis, S., and Ellaway, P. (1997). Effects of antipsychotic medication on electromyographic responses to transcranial magnetic stimulation of the motor cortex in schizophrenia. *J Neurol Neurosurg Psychiatry*, 63(4) :468–73.

[Day et al., 1990] Day, B., Dressler, D., Hess, C., Maertens de Noordhout, A., Marsden, C., Mills, K., et al. (1990). Direction of current in magnetic stimulating coils used for percutaneous activation of brain, spinal cord and peripheral nerve. *J Physiol*, 430 :617.

[Demeyer et al., 2003] Demeyer, S., Ducasse, S., and Nierstrasz, O. (2003). *Object-Oriented Reengineering Patterns.* Morgan Kaufmann.

[DeRemer and Kron, 1975] DeRemer, F. and Kron, H. (1975). Programming-in-the large versus programming-in-the-small. *Proceedings of the international conference on Reliable software table of contents*, pages 114–121.

[Edelsbrunner and Mücke, 1992] Edelsbrunner, H. and Mücke, E. (1992). *Three-dimensional alpha shapes*. ACM Press New York, NY, USA.

[Epstein, 1996] Epstein, C. (1996). Optimum stimulus parameters for lateralized suppression of speech with magnetic brain stimulation. *Neurology*, 47(6) :1590–1593.

[Erl, 2004] Erl, T. (2004). *Service-Oriented Architecture : A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR Upper Saddle River, NJ, USA.

[Fabri et al., 1996] Fabri, A., Giezeman, G., Kettner, L., Schirra, S., and Schonherr, S. (1996). The CGAL kernel : A basis for geometric computation. *Proc. 1st ACM Workshop on Appl. Comput. Geom*, 1148 :191–202.

[Favre, 1995] Favre, J. (1995). *Maintenance et Ré-ingénierie globale des logiciels*. PhD thesis, Thèse de doctorat, Université de Grenoble, 1996.

[Favre et al., 2006] Favre, J., Estublier, J., and Blay-Fornarino, M. (2006). *l'ingéniérie dirigée par les modèles au delà du MDA*. Lavoisier.

[Fowler, 2003] Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

[Frey, 2000] Frey, P. (2000). Medit : outil de visualisation interactif.

[Garcia et al., 2009] Garcia, A., Combemale, B., Crégut, X., Guyot, J., and Libert, B. (2009). TOP PROCESS. *Revue de l'Electricité et de l'Electronique (REE)*.

[George et al., 1999] George, M., Lisanby, S., and Sackeim, H. (1999). Transcranial magnetic stimulation : Applications in neuropsychiatry. *Arch. Gen. Psychiatry*, 56(4) :300–311.

[George et al., 2003] George, M., Nahas, Z., Kozol, F., Li, X., Yamanaka, K., Mishory, A., and Bohning, D. (2003). Mechanisms and the current state of transcranial magnetic stimulation. *CNS Spectr*, 8(7) :496–514.

[George and Wassermann, 1994] George, M. and Wassermann, E. (1994). Rapid-rate transcranial magnetic stimulation and ECT. *Convuls Ther*, 10(4) :251–4.

[George et al., 1995] George, M., Wassermann, E., Williams, W., Callahan, A., Ketter, T., Basser, P., Hallett, M., and Post, R. (1995). Daily repetitive transcranial magnetic stimulation (rTMS) improves mood in depression. *Neuroreport*, 6(14) :1853–6.

[Geuzaine and Remacle, 2002] Geuzaine, C. and Remacle, J. (2002). Gmsh : a three-dimensional finite element mesh generator with built-in pre-and post-processing facilities. *available from internet :< URL : http ://www. geuz. org/gmsh.*

[Grandori and Ravazzani, 1991] Grandori, F. and Ravazzani, P. (1991). Magnetic stimulation of the motor cortex–theoretical considerations. *IEEE Trans Biomed Eng*, 38(2) :180–91.

[Gray et al., 1994] Gray, W., Wikramanayake, G., and Fiddian, N. (1994). Assisting legacy database migration. *Legacy Information System-Barriers to Business Process Re-engineering (Digest no. 1994/246), IEE Coilloquium on*, page 5.

[Hallett, 2000] Hallett, M. (2000). Transcranial magnetic stimulation and the human brain. *Nature*, 406 :147–50.

[Hédou, 1997] Hédou, V. (1997). Méthodes numériques pour la modélisation électro-anatomique du cerveau. Mathématique et applications, Thèse de l'Université de Rennes I -.

[Heller and van Hulsteyn, 1992] Heller, L. and van Hulsteyn, D. (1992). Brain stimulation using electromagnetic sources : theoretical aspects. *Biophysical Journal*, 63(1) :129–138.

[Hines, 1993] Hines, M. (1993). NEURON - a program for simulation of nerve equations. *Neural Systems : Analysis and Modeling*, pages 127–136.

[Hodgkin and Huxley, 1952] Hodgkin, A. and Huxley, A. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J Physiol*, 117(4) :500–544.

[Hoffman et al., 2000] Hoffman, R., Boutros, N., Hu, S., Berman, R., Krystal, J., and Charney, D. (2000). Transcranial magnetic stimulation and auditory hallucinations in schizophrenia. *The Lancet*, 355(9209) :1073–1075.

[Iacob et al., 2008] Iacob, M., Steen, M., and Heerink, L. (2008). Reusable Model Transformation Patterns. In *Workshop on Models and Model-driven Methods for Enterprise Computing (3M4EC))*, page 1.

[Ilmoniemi et al., 1985] Ilmoniemi, R., Hamalainen, M., and Knuutila, J. (1985). The forward and inverse problems in the spherical model. *Biomagnetism : Applications and Theory*, pages 278–282.

[Ilmoniemi et al., 1997] Ilmoniemi, R. J., Virtanen, J., Ruohonen, J., Karhu, J., Aronen, H. J., Näätänen, R., and Katila, T. (1997). Neuronal responses to magnetic stimulation reveal cortical reactivity and connectivity. *NeuroReport*, 8 :3537–3540.

[Jalinous, 1998] Jalinous, R. (1998). Guide to magnetic stimulation. *The Magstim Company Limited, U.K.*

[Jouault and Bezivin, 2006] Jouault, F. and Bezivin, J. (2006). KM3 : a DSL for Metamodel Specification. *Lecture Notes in Computer Science*, 4037 :171–185.

[Kamitani, 2001] Kamitani, Y. (2001). Psychobiophysics of transcranial magnetic stimulation. Master's thesis, California Institute of Technology.

[Kleppe et al., 2003] Kleppe, A., Bast, W., and Warmer, J. (2003). *MDA Explained : The Model Driven Architecture : Practice and Promise*. Addison-Wesley Professional.

[Koch et al., 2008] Koch, N., Meliá-Beigbeder, S., Moreno-Vergara, N., Pelechano-Ferragud, V., Sánchez-Figueroa, F., and Vara-Mesa, J. (2008). Model-Driven Web Engineering. *UPGRADE, The European Journal for the Informatics Professional*, IX(2) :40–45.

[Krasteva et al., 2002] Krasteva, V., Papazov, S., and Daskalov, I. (2002). Magnetic stimulation for non-homogeneous biological structures. *BioMedical Engineering OnLine*, 1(1) :3.

[Krings et al., 2001] Krings, T., Chiappa, K., Foltys, H., Reinges, M., Cosgrove, G., and Thron, A. (2001). Introducing navigated transcranial magnetic stimulation as a refined brain mapping methodology. *Neurosurg Rev*, 24(4) :171–9.

[Lancaster et al., 2004] Lancaster, J., Narayana, S., Wenzel, D., Luckemeyer, J., Roby, J., and Fox, P. (2004). Evaluation of an image-guided, robotically positioned transcranial magnetic stimulation system. *Hum Brain Mapp*, 22(4) :329–40.

[Lano and Clark, 2008] Lano, K. and Clark, D. (2008). Model Transformation Specification and Verification. In *Quality Software, 2008. QSIC'08. The Eighth International Conference on*, pages 45–54.

[Lefaucheur et al., 2001] Lefaucheur, J., Drouot, X., and Nguyen, J. (2001). Interventional neurophysiology for pain control : duration of pain relief following repetitive transcranial magnetic stimulation of the motor cortex. *Neurophysiol Clin.*, 31(4) :247–52.

[Lorensen and Cline, 1987] Lorensen, W. and Cline, H. (1987). Marching cubes : A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4) :163–169.

[Luquet et al., 2004] Luquet, S., Barra, V., Haug, C., and Lemaire, J. (2004). Calcul du champ magnétique dans le cadre de la stimulation magnétique transcrânienne : utilisation d'un modèle paramétrique de bobine. *Journée Sciences, Technologies et Imagerie pour la Médecine*.

[Luquet et al., 2005a] Luquet, S., Barra, V., Haug, C., and Lemaire, J. (2005a). Simulation of transcranial magnetic stimulation : Brain mapping of magnetic field potential. *IBMISPS, International Brain Mapping and Interaoperative Sugical Symposium, Los Angles 2005*.

[Luquet et al., 2005b] Luquet, S., Barra, V., and Lemaire, J. (2005b). Transcranial magnetic stimulation : Magnetic field computation using a parametrical coil model. *OICMS, 1st Open International Conference on Modeling & Simulation Clermont-Ferrand 2005*, pages 451–455.

[Luquet et al., 2005c] Luquet, S., Barra, V., and Lemaire, J. (2005c). Transcranial Magnetic Stimulation : Magnetic Field Computation in empty free space. *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, pages 4365–4368.

[Mally and Stone, 1999] Mally, J. and Stone, T. (1999). Therapeutic and "dose-dependent" effect of repetitive microelectroshock induced by transcranial magnetic stimulation in parkinson's disease. *J Neurosci Res*, 57(6) :935–40.

[Marieb et al., 1993] Marieb, E., Lachaîne, R., and Moussakova, L. (1993). *Anatomie et physiologie humaines*. De Boeck Université.

[Marin-Padilla, 1969] Marin-Padilla, M. (1969). Origin of the pericellular baskets of the pyramidal cells of the human motor cortex : a Golgi study. *Brain Res*, 14(3) :633–46.

[McCann et al., 1998] McCann, U., Kimbrell, T., Morgan, C., Anderson, T., Geraci, M., Benson, B., Wassermann, E., Willis, M., and Post, R. (1998). Repetitive Transcranial Magnetic Stimulation for Posttraumatic Stress Disorder.

[McCaracken and Garbassi, 1970] McCaracken, D. and Garbassi, U. (1970). *A Guide to COBOL Programming*. John Wiley & Sons, Inc. New York, NY, USA.

[McConnell et al., 2001] McConnell, K., Nahas, Z., Shastri, A., Lorberbaum, J., Kozel, F., Bohning, D., and George, M. (2001). The transcranial magnetic stimulation motor threshold depends on the distance from coil to underlying cortex : a replication in healthy adults comparing two methods of assessing the distance to cortex. *Biol Psychiatry*, 49(5) :454–9.

[Mendelzon and Sametinger, 1995] Mendelzon, A. and Sametinger, J. (1995). Reverse Engineering by Visualizing and Querying. *Software - Concepts and Tools*, 16(4) :170–182.

[Miranda et al., 2003] Miranda, P., Hallett, M., and Basser, P. (2003). The electric field induced in the brain by magnetic stimulation : a 3-D finite-element analysis of the effect of tissue heterogeneity and anisotropy. *Biomedical Engineering, IEEE Transactions on*, 50(9) :1074–1085.

[Nadeem et al., 2003] Nadeem, M., Thorlin, T., Gandhi, O., and Persson, M. (2003). Computation of electric and magnetic stimulation in human head using the 3-d impedance method. *IEEE Transactions on Biomedical Engineering*, 50(7) :900–907.

[Nguyen et al., 2003] Nguyen, J.-P., Lefaucheur, J., and Keravel, Y. (2003). Motor cortex stimulation. *Electrical Stimulation in Pain Relief - Pain Research and Clinical Managemnet Series*, 15(13) :197–209.

[Noirhomme et al., 2004] Noirhomme, Q., Ferrant, M., Vandermeeren, Y., Olivier, E., Macq, B., and Cuisenaire, O. (2004). Registration and real-time visualization of transcranial magnetic stimulation with 3-d mr images. *IEEE Trans Biomed Eng*, 51(11) :1994–2005.

[Noirhomme et al., 2002] Noirhomme, Q., Romero, E., Cuisenaire, O., Ferrant, M., Vandermeeren, Y., Olivier, E., and Macq, B. (2002). Registration of transcranial magnetic stimulation, a visualization tool for brain functions. In *Proceedings of DSP*, pages 311–314.

[Olsen, 2008] Olsen, R. (2008). *Design Patterns in Ruby*. Addison-Wesley.

[OMG, 2006] OMG (2006). *Object Constraint Language (OCL) 2.0 Specification*. OMG.

[OMG, 2008] OMG (2008). *Software & Systems Process Engineering Meta-Model (SPEM) 2.0*. OMG.

[Parastoo Mohagheghi, 2009] Parastoo Mohagheghi, Vegard Dehlena, T. N. (2009). Definitions and approaches to model quality in model-based software development - A review of literature. In *Information and Software Technology*, pages Article in Press, Corrected Proof.

[Parr and Quong, 1995] Parr, T. and Quong, R. (1995). ANTLR : A Predicated-LL(k) Parser Generator. *Software - Practice and Experience*, 25(7) :789–810.

[Pascual-Leone, 1991] Pascual-Leone, A. (1991). Induction of speech arrest and counting errors with rapid-rate transcranial magnetic stimulation. *Neurology*, 41(5) :697–702.

[Paus et al., 1997] Paus, T., Jech, R., Thompson, C., Comeau, R., Peters, T., and Evans, A. (1997). Transcranial magnetic stimulation during positron emission tomography : a new method for studying connectivity of the human cerebral cortex. *J Neurosci.*, 17(9) :3178–3184.

[Persson et al., 2003] Persson, M., NADEEM, M., and THORLIN, T. (2003). COMPARING ECT AND TMS USING THE 3D IMPEDANCE METHOD. *Department of Electromagnetic, Institute of Clinical Neuroscience, Gothenburg University, S-413*, 45.

[Pinzger et al., 2005] Pinzger, M., Gall, H., Fischer, M., and Lanza, M. (2005). Visualizing multiple evolution metrics. *Proceedings of the 2005 ACM symposium on Software visualization*, pages 67–75.

[Press, 1992] Press, W. (1992). *Numerical recipes in C : the art of scientific computing*. Cambridge University Press.

[Puri, 1996] Puri, B. (1996). An investigation of motor function in schizophrenia using transcranial magnetic stimulation of the motor cortex. *The British Journal of Psychiatry*, 169(6) :690–695.

[Rahim and Mansoor, 2008] Rahim, L. and Mansoor, S. (2008). Proposed Design Notation for Model Transformation. In *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, pages 589–598.

[Ravazzani et al., 1996] Ravazzani, P., Ruohonen, J., Grandori, F., and Tognola, G. (1996). Magnetic stimulation of the nervous system : induced electric field in unbounded, semi-infinite, spherical, and cylindrical media. *Ann Biomed Eng*, 24(5) :606–16.

[Reenskaug, 1979] Reenskaug, T. (1979). THING-MODEL-VIEW-EDITOR : an Example from a Planning System. *Xerox PARC Technical Note (May 1979)*.

[Reenskaug, 2003] Reenskaug, T. (2003). The Model-View-Controller (MVC) Its Past and Present. *Java Zone*.

[Rollnik et al., 2002] Rollnik, J., Wüstefeld, S., Daüper, J., Karst, M., Fink, M., Kossev, A., and Dengler, R. (2002). Repetitive Transcranial Magnetic Stimulation for the Treatment of Chronic Pain – A Pilot Study. *European Neurology*, 48(1) :6–10.

[Roth and Basser, 1990] Roth, B. and Basser, P. (1990). A model of the stimulation of a nerve fiber by electromagneticinduction. *Biomedical Engineering, IEEE Transactions on*, 37(6) :588–597.

[Roth et al., 2002] Roth, Y., Zangen, A., and Hallett, M. (2002). A coil design for transcranial magnetic stimulation of deep brain regions. *J Clin Neurophysiol*, 19(4) :361–70.

[Ruohonen, 1998] Ruohonen, J. (1998). Transcranial magnetic stimulation : Modelling and new techniques. Doctor of technology, Helsinki University of Technology.

[Ruohonen et al., 1996] Ruohonen, J., Panizza, M., Nilsson, J., Ravazzani, P., Grandori, F., and Tognola, G. (1996). Transverse-field activation mechanism in magnetic stimulation of peripheral nerves. *Electroencephalogr Clin Neurophysiol*, 101(2) :167–74.

[Sack and Linden, 2003] Sack, A. and Linden, D. (2003). Combining transcranial magnetic stimulation and functional imaging in cognitive brain research : possibilities and limitations. *Brain Research Reviews*, 43 :41–56.

[Sarvas, 1987] Sarvas, J. (1987). Basic mathematical and electromagnetic concepts of the biomagnetic inverse problem. *Physics in Medicine and Biology*, 32(1) :11–22.

[Schroeder et al., 2000] Schroeder, W., Avila, L., and Hoffman, W. (2000). Visualizing with VTK : a tutorial. *Computer Graphics and Applications, IEEE*, 20(5) :20–27.

[Schroeder et al., 1992] Schroeder, W., Zarge, J., and Lorensen, W. (1992). Decimation of triangle meshes. *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70.

[Schwartz, 1993] Schwartz, L. (1993). *Calcul intégral - Analyse Tome 3*. Hermann.

[Seidewitz and Technologies, 2003] Seidewitz, E. and Technologies, I. (2003). What models mean. *Software, IEEE*, 20(5) :26–32.

[Selic, 2008] Selic, B. (2008). MDA Manifestations. *UPGRADE, The European Journal for the Informatics Professional*, IX(2) :12–16.

[Shimamoto et al., 2001] Shimamoto, H., Takasaki, K., Shigemori, M., Imaizumi, T., Ayabe, M., and Shoji, H. (2001). Therapeutic effect and mechanism of repetitive transcranial magnetic stimulation in Parkinson's disease. *Journal of Neurology*, 248 :48–51.

[Siebner et al., 1999] Siebner, H., Tormos, J., Ceballos-Baumann, A., Auer, C., Catala, M., Conrad, B., and Pascual-Leone, A. (1999). Low-frequency repetitive transcranial magnetic stimulation of the motor cortex in writer's cramp. *Neurology*, 52(3) :529–37.

[Siek et al., 2002] Siek, J., Lee, L., and Lumsdaine, A. (2002). *The Boost Graph Library : User Guide and Reference Manual*. Addison-Wesley.

[Smart, 1995] Smart, J. (1995). User Manual for wxWindows 1.65 : a portable C++ GUI toolkit. *Artificial Intelligence Applications Institute, University of Edinburgh, Aug.*

[Stewart et al., 2001] Stewart, L., Walsh, V., Frith, U., and Rothwell, J. (2001). TMS produces two dissociable types of speech disruption. *Neuroimage*, 13(3) :472–478.

[Szuba et al., 2001] Szuba, M., O'Reardon, J., Rai, A., Snyder-Kastenberg, J., Amsterdam, J., Gettes, D., Wassermann, E., and Evans, D. (2001). Acute mood and thyroid stimulating hormone effects of transcranial magnetic stimulation in major depression. *Biological Psychiatry*, 50(1) :22–27.

[Theodore et al., 2002] Theodore, W., Hunter, K., Chen, R., Vega-Bermudez, F., Boroojerdi, B., Reeves-Tyer, P., Werhahn, K., Kelley, K., and Cohen, L. (2002). Transcranial magnetic stimulation for the treatment of seizuresA controlled study. *Neurology*, 59(4) :560–562.

[Thielscher and Kammer, 2002] Thielscher, A. and Kammer, T. (2002). Linking physics with physiology in tms : A sphere field model to determine the cortical stimulation site in tms. *Neuroimage*, 17(3) :1117–1130.

[Thielscher and Kammer, 2004] Thielscher, A. and Kammer, T. (2004). Electric field properties of two commercial figure-8 coils in tms : calculation of focality and efficiency. *Clinical Neuophysiology*, 115(7) :1697–708.

[Tofts, 1990] Tofts, P. (1990). The distribution of induced currents in magnetic stimulation of the nervous system. *Phys. Med. Biol.*, 35 :1119–1128.

[Ueno et al., 1988] Ueno, S., Tashiro, T., and Harada, K. (1988). Localized stimulation of neural tissues in the brain by means of a paired configuration of time-varying magnetic fields. *Journal of Applied Physics*, 64(10) :5862–5864.

[Wassermann et al., 1996] Wassermann, E., Grafman, J., Berry, C., Hollnagel, C., Wild, K., Clark, K., and Hallett, M. (1996). Use and safety of a new repetitive transcranial magnetic stimulator. *Electroencephalogr Clin Neurophysiol.*, 101(5) :412–7.

[Wassermann et al., 1992] Wassermann, E., McShane, L., Hallett, M., and Cohen, L. (1992). Noninvasive mapping of muscle representations in human motor cortex. *Electroencephalogr Clin Neurophysiol*, 85(1) :1–8.

[Watson, 2008] Watson, A. (2008). A Brief History of MDA. *UPGRADE, The European Journal for the Informatics Professional*, IX(2) :7–11.

[Wedegaertner et al., 1997] Wedegaertner, F., Garvey, M., Cohen, L., Hallett, M., and Wassermann, E. (1997). Low frequency repetitive transcranial magnetic stimulation can reduce action myoclonus. *Neurology*, 48 :119–126.

[Wilson et al., 1989] Wilson, M., Bhalla, U., Uhley, J., and Bower, J. (1989). GENESIS : a system for simulating neural networks. *Advances in neural information processing systems 1 table of contents*, pages 485–492.

[Zangen et al., 2005] Zangen, A., Roth, Y., Voller, B., and Hallett, M. (2005). Transcranial magnetic stimulation of deep brain regions : evidence for efficacy of the H-coil. *Clin Neurophysiol*, 116(4) :775–9.

[Ziemann, 1997] Ziemann, U. (1997). Decreased motor inhibition in Tourette's disorder : evidence from transcranial magnetic stimulation.

[Ziemann et al., 1998] Ziemann, U., Steinhoff, B., Tergau, F., and Paulus, W. (1998). Transcranial magnetic stimulation : its current role in epilepsy research. *Epilepsy Res.*, 30(1) :11–30.

Annexe

TortoiseSVN et WinMerge

Cette annexe sera principalement constituée de captures d'écran pour essayer de reproduire le niveau d'intégration de TortoiseSVN dans l'explorateur de Windows ainsi que l'interaction entre TortoiseSVN et WinMerge.

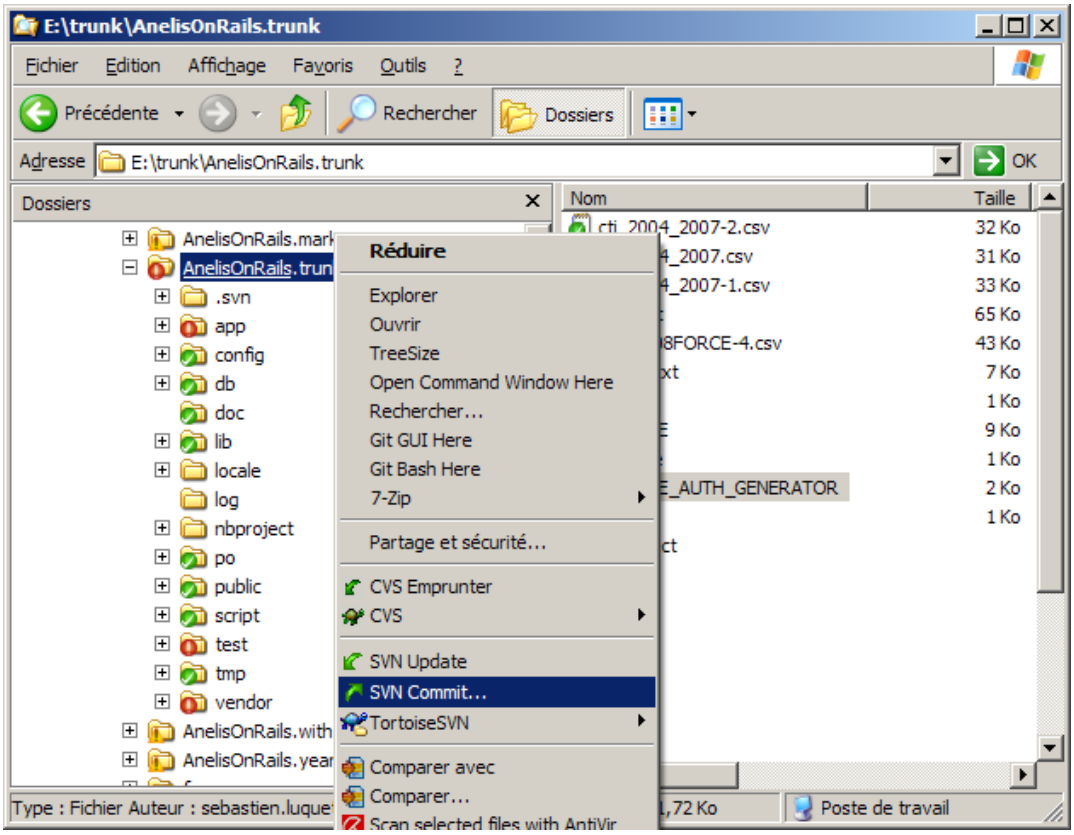

FIGURE I – Quelque part à la racine de l'arborescence d'un projet

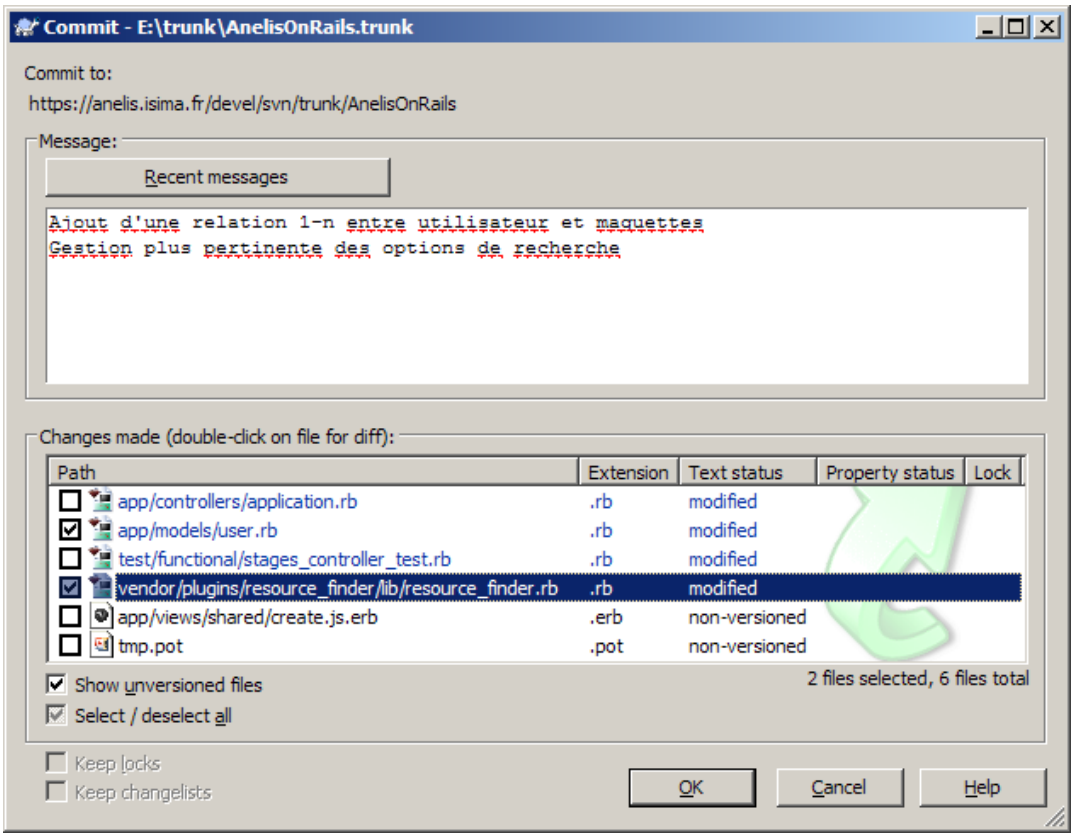

FIGURE II – L'équivalent de la commande status. La boîte fait apparaître les fichiers modifiés. Il est possible de sélectionner les fichiers que l'on souhaite valider. Un double-clic sur un fichier ouvre la comparaison de la version courante avec la version initiale du fichier

Le seul inconvénient de ces deux outils est qu'ils n'existent que sous Windows et que nous n'avons jamais rencontré une association de Explorateur de Fichier + Client SVN (status) + Diff/Merge aussi performante sous d'autre système d'exploitation. Il existe des plugins Subversion pour Eclipse ou Netbeans, mais là aussi nous n'avons jamais retrouvé un niveau d'intégration et une rapidité d'utilisation comparable.

Seule la ligne de commande atteint le même niveau de performance pour d'autres utilisations de SVN. Notons cependant que certains travaux visent à étendre Eclipse par un module de comparaison de modèle [Brun and Pierantonio, 2008].

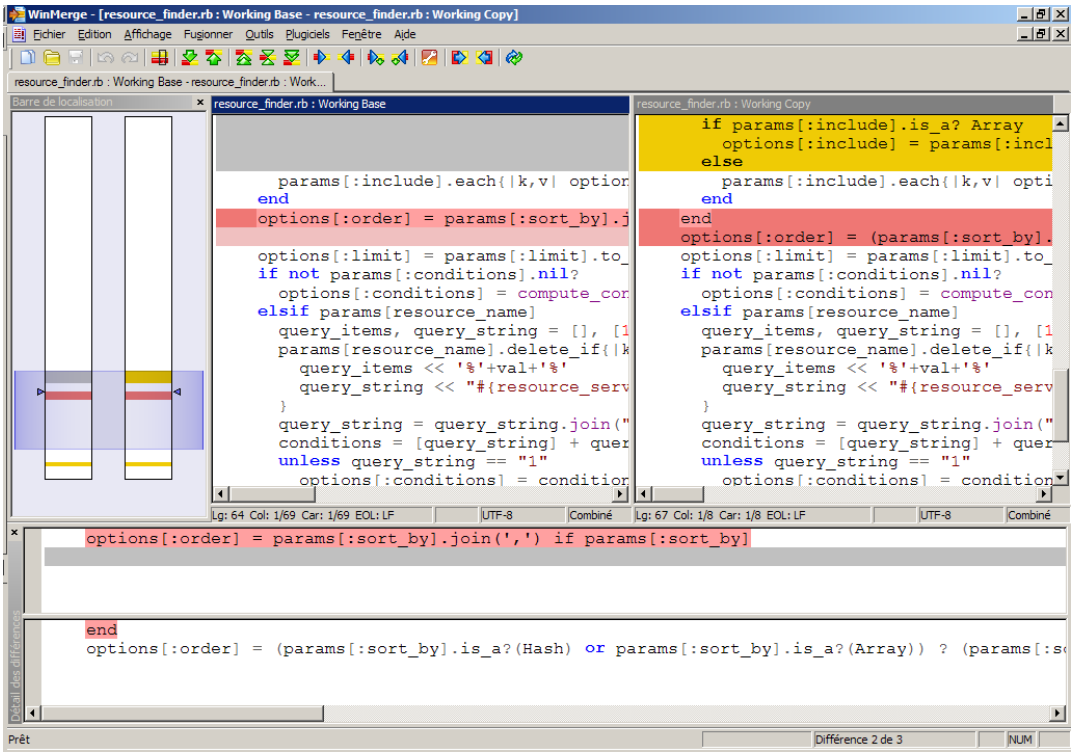

FIGURE III – Comparaison d'un fichier avant sa soumission au serveur. On peut naviguer entre chaque modification en pressant simplement les touches Alt+Up/Down. Cela permet de vérifier quelles étaient les modifications utiles et celles qui avaient été faites uniquement pour des tests en local.

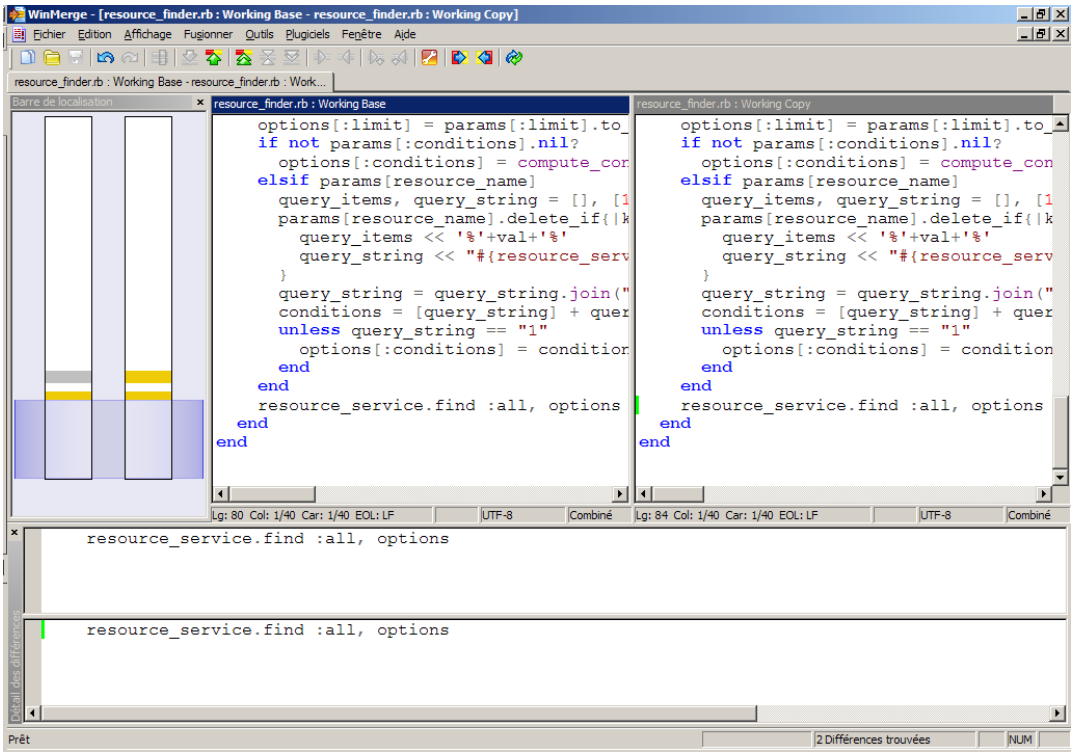

FIGURE IV – La dernière modification était en trop. On rapatrie le code original depuis la version non modifiée en pressant simplement Alt+Right

Résumé

La Stimulation Magnétique Transcrânienne (SMT) est une technique de stimulation neuronale offrant de nombreuses applications médicales. Cependant son utilisation reste empirique. L'objectif de cette thèse était de mettre en place un logiciel permettant de mieux comprendre les effets de la stimulation et d'aider à la réalisation de séances de SMT. Suite au développement de ce logiciel, il est apparu que celui-ci était devenu patrimonial. L'objectif secondaire de ce travail fut donc d'analyser l'obsolescence du logiciel et d'essayer d'apporter des solutions pour limiter ce phénomène via l'utilisation de l'Ingénierie Dirigée par les Modèles (IDM).

Après avoir présenté les phénomènes régissant l'activité cérébrale nous présentons les différentes techniques de stimulation neuronale et les avantages qu'offre la SMT. Nous présentons également les bases nécessaires à la compréhension des phénomènes électromagnétiques, puis une introduction aux concepts fondamentaux de l'IDM. Dans une troisième partie, l'accent est mis sur les différentes modélisations et méthodes de calcul des effets de la Stimulation Magnétique Transcrânienne avant de présenter la solution qui a été retenue et implantée dans le simulateur.

Les deux dernières parties du manuscrit se focalisent sur le simulateur dans sa globalité (visualisation 3D) puis à la manière dont il pourrait être refactorisé pour faciliter sa maintenance et l'inclusion des évolutions possibles que nous présentons en conclusion.

Mots-clefs : Stimulation Magnétique Transcrânienne, IDM, Logiciel patrimonial, Refactoring

Abstract

Transcranial Magnetic Stimulation (TMS) is a new technique for brain stimulation. TMS has many medical applications but TMS uses are still empirical. With the development of a simulator, this thesis tries to explain some of the phenomena implicated in TMS. But the simulator became legacy software and we tried to explain why and how we could refactor this application using Model Driven Engineering (MDE).

The first part of this document deals with cerebral activity description and techniques for brain stimulation. Secondly we will expose MDE concepts. Then, we will expose the differents kinds of TMS modelling used in the literature and our method to compute TMS effect into patient head.

Finally, we will describe the integration of this computationnal module into a 3D visualization software application and how we could refactor and extend this legacy application in the future.

Keywords : Transcranial Magnetic Stimulation, MDE, Legacy software, Refactoring