Introduction de processus de conception pour la modélisation interactive de modèles physiques particulaires 3D complexes dans l'environnement MIMESIS

Ali Allaoui

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE
Spécialité **Ingénierie de l'interaction, de la cognition, de l'apprentissage et de la création**

Arrêté ministériel : 7 août 2006

Présentée et soutenue publiquement par

ALLAOUI Ali

le **21 octobre 2010**

INTRODUCTION DE PROCESSUS DE CONCEPTION POUR LA MODELISATION INTERACTIVE DE MODELES PHYSIQUES PARTICULAIRES 3D COMPLEXES DANS L'ENVIRONNEMENT MIMESIS

Thèse dirigée par **Cadoz Claude** et codirigée par **Luciani Annie**

JURY

Nigay Laurence	Présidente	Université de Grenoble
Faudot Dominique	Rapporteuse	Université de Dijon
Luga Hervé	Rapporteur	Université de Toulouse I
Bollaert Gilles	Examinateur	Ecole Européenne Supérieure de l'Image
Claude Cadoz	Examinateur	Ministère de la culture
Annie Luciani	Examinatrice	Ministère de la culture

Thèse préparée au sein du **Laboratoire ICA** dans **l'Ecole Doctorale EDISCE**

Remerciements

Je remercie tout d'abord Annie Luciani pour m'avoir fait confiance pendant ces quatre années et pour m'avoir insufflé l'esprit de la modélisation physique pour l'animation. Je remercie Claude Cadoz, Jean-Loup Florens et Annie Luciani pour avoir fondé l'ACROE et construit une pensée forte autour de l'instrumentalité et des nouvelles technologies. Le formalisme CORDIS-ANIMA, pierre angulaire de cette pensée, leur est dû et est un élément essentiel de ce travail de thèse. Je remercie tous les membres actuels et passés du groupe ICA-ACROE, et au premier rang l'équipe Image, aujourd'hui composée de Kévin Sillam, Maria Christou, Saman Kalantari et Annie Luciani. Je remercie également Matthieu Evrard, parti vers d'autres horizons en janvier 2010 et qui m'a accueilli dans cette équipe, ainsi que Chi Min Hsieh, premier docteur en Art, Science Technologie, et aujourd'hui professeur à Taiwan. J'ai beaucoup partagé avec eux dans les premières années, je leur dois beaucoup. Je remercie mes colocataires pour m'avoir supporté, surtout pendant cette 4ème et dernière année, mon moral leur doit beaucoup. Je remercie ma famille, mon père, ma sœur et ma mère, disparue il y a maintenant treize ans. Ce modeste ouvrage lui est dédié.

Statut du mémoire de thèse au regard de la propriété intellectuelle

Ce mémoire de thèse a été rédigé pour servir de support à l'obtention du grade de docteur. Bien que ne portant que le nom de son auteur, il est le fruit d'un travail d'équipe et a été rédigé en collaboration avec le directeur de thèse. Il ne constitue pas une publication scientifique normalisée, pour laquelle tous les contributeurs au travail dont il est rendu compte sont clairement identifiés.

Nous rappelons donc ici qu'un mémoire de thèse ne devrait pas faire l'objet de citations dans le cadre de publications normalisées, notamment de publications scientifiques.

Toute référence au travail rapporté dans ce mémoire devrait s'appuyer sur les différentes publications (actes de conférences, articles de journaux, livres) qui s'y rattachent. Les publications ou usages ultérieurs qui pourraient se baser sur ou reprendre tout ou partie de ce travail, se devront de respecter les participations de chacun, présentes et passées.

Table des matières

Table des matières 5
Table des figures 9
Table des équations 12
CHAPITRE I. UN LANGAGE POUR L'ART DU MOUVEMENT 17
1. Image et mouvement 18
2. Image animée et ordinateur 20
2.1. Une brève histoire de l'informatique graphique 20
2.2. Modèles descriptifs et modèles générateurs 20
2.3. Diversité des modèles physiques 21
3. CORDIS-ANIMA 24
3.1. Pourquoi et comment concevoir du mouvement 24
3.2. De Newton aux réseaux masses-interactions 25
3.3. CORDIS-ANIMA, un formalisme pour les réseaux masses-interactions 27
3.4. Simulation et algorithmes 29
3.5. Un formalisme tourné vers le temps réel et la multisensorialité 30
4. MIMESIS 32
4.1. Historique : du langage à l'environnement interactif 32
4.2. Processus de modélisation avec CORDIS-ANIMA 33
4.3. MIMESIS 4 34
5. Conclusion 36
CHAPITRE II. RESEAUX MASSES-INTERACTIONS ET PHENOMENES COMPLEXES 39
1. Non-linéarités et contrôle paramétrique en ligne dans un réseau masses-interactions 40
1.1. Introduction 40
1.2. Historique 41
1.3. Contrôle paramétrique en ligne dans CORDIS-ANIMA 45
2. Processus de contrôle de modèles physiques : une étude comparative 48
2.1. Contrôle de paramètres et contrôle de la simulation 48
2.2. Modélisation de mouvements de vers et de serpents 49
2.3. Créatures virtuelles et modèles comportementaux 50
2.4. Réseaux de neurones et contrôleurs 51
2.5. Contrôleurs et modèles physiques thermiques 51
2.6. Conclusion 52
3. Les briques pour le contrôle paramétrique dans CORDIS-ANIMA 53
3.1. Introduction 53

3.2. Réseaux CORDIS-ANIMA unidimensionnels 54
3.3. CORDIS-ANIMA 1D comme langage de conception des processus de contrôle. 55
3.4. Les modules de contrôle paramétrique 55
3.5. Les modules de mesures 56
3.6. Contrôlabilité des paramètres 58
4. Pratique du contrôle paramétrique avec CORDIS-ANIMA, deux exemples de modélisation 60
4.1. Modélisation du phénomène de striction 60
4.2. Modélisation de sauts à pieds joints en série 65
5. Bilan et Perspectives 71
CHAPITRE III. OUTILS POUR LA CREATION DE RESEAUX CORDIS-ANIMA COMPLEXES 73
1. Des outils pour créer et paramétrer un réseau CORDIS-ANIMA 74
1.1. Introduction 74
1.2. Nommer les modules 74
1.3. Connecter les modules. 77
1.4. Structures répétitives et outils associés 79
1.5. Duplication de sous-parties de réseau 82
1.6. Manipulations des paramètres physiques 83
1.7. Synthèse 84
2. Les piliers de MIMESIS V 86
2.1. Analyse des tâches 86
2.2. Un langage de script : PNSL (Physical Network Scripting Language) 89
2.3. Un système de labellisation : créer un espace de noms cohérent 89
2.4. Filtres et opérateurs ensemblistes 91
2.5. Des espaces graphiques pour la sélection, la manipulation et la visualisation 92
2.6. Des outils de manipulation du réseau 95
2.7. Des outils de connexion évolués : des labels pour les points de communication 97
2.8. Contrôle de paramètres dans MIMESIS V 99
2.9. La collaboration entre modalité graphique et PNSL : une nécessité problématique 100
3. Cadre d'étude pour la collaboration graphique textuelle 102
3.1. Les cadres d'étude pour la multimodalité 102
3.2. Métaphores d'interaction avec l'ordinateur 104
3.3. Evaluation d'une interface 106
3.4. Conclusion 107

4. Conditions et enjeux de la collaboration entre modalité graphique et modalité script 109
4.1. Champ de la modalité graphique dans l'interface MIMESIS 109
4.2. Travailler sur le processus comme si on travaillait sur l'état 110
4.3. Un objectif simple : la collaboration horizontale. 111
4.4. Un objectif complexe : la collaboration verticale 111
4.5. Enjeux liés à la représentation graphique du résultat d'un script 113
4.6. Conclusion 114
5. Outils pour la collaboration entre modalité graphique et modalité script 115
5.1. Introduction 115
5.2. L'interaction instrumentale 116
5.3. Fonction 117
5.4. Encapsulation 119
5.5. Conclusion 122
6. Organisation et usage des outils dans MIMESIS V 124
6.1. Organisation de MIMESIS en 5 temps 124
6.2. Fonctions, capsules et séparation des phases 125
6.3. Outils auxiliaires 125
6.4. Pédagogie 126
6.5. Conclusion 127
7. Scenarii 128
7.1. Introduction 128
7.2. Agglomérat et oscillateur 128
7.3. Conception d'une structure complexe de grande taille 137
7.4. Modèle à deux couches : topologie agglomérat sur topologie graphe de Heawood 145
7.5. Conclusion 148
8. Conclusion 149
CHAPITRE IV. CONDITIONS INITIALES ET SPATIALITE DANS LA MODELISATION PHYSIQUE : UN ENJEU POUR LES MODELES COMPLEXES 151
1. Enjeux liés aux conditions initiales 152
1.1. Enoncé du problème 152
1.2. Relation forme-mouvement 154
1.3. Ciblage de l'objet d'étude 155
2. Introduction de structures spatiales élémentaires au sein de MIMESIS. 159
2.1. Introduction 159

2.2. Objets spatiaux : quelle séparation entre topologie spatiale et plongement géométrique ? 160
2.3. Les objets spatiaux élémentaires dans MIMESIS 160
2.4. Les opérateurs spatiaux : définir et manipuler des objets spatiaux..... 161
2.5. Enjeux du lien entre objets spatiaux et conditions initiales 162
2.6. Espace graphique pour la manipulation des conditions initiales..... 163
2.7. Conclusion..... 165
3. Importation de structures spatiales dans MIMESIS : positions initiales et interactions physiques 166
3.1. Introduction 166
3.2. Etat de l'Art 167
3.3. Méthode d'importation de données géométriques dans MIMESIS V 171
3.4. Contour optique et contour mécanique..... 172
3.5. Chapelet et tapis de masses..... 172
3.6. Sphère circonscrite 173
3.7. Sphérisation 174
3.8. Conclusion..... 175
4. Obstacles fixes : Interactions anisotropes pour le codage de la forme 176
4.1. Principe et champ d'application 176
4.2. Description de l'existant dans MIMESIS 4 177
4.3. Evolutions 178
4.4. Conclusion..... 179
5. Expérimentations 180
5.1. Sphérisation 180
5.2. Rebond sur un sol "lisse" : comparaison de trois méthodes de modélisation.... 181
5.3. Scénario de collaboration horizontale 184
6. Conclusion 185
CHAPITRE V. CONCLUSION ET PERSPECTIVES..... 189
Glossaire..... 195
Bibliographie..... 197
Annexe 1 : Les modules de MIMESIS 4..... 206
Annexe 2 : Le saut et ses différentes étapes..... 210
Annexe 3 : La goutte..... 211
Annexe 4 : KPovModeler, un exemple de réification du script..... 212

Table des figures

Figure 1 : Processus de création pour l'Art du Mouvement 25
Figure 2 : Un réseau masses-interactions 26
Figure 3 : Points M et L 27
Figure 4 : Un réseau CORDIS-ANIMA élémentaire 28
Figure 5 : Aperçu de l'interface MIMESIS 4 34
Figure 6 : Modèle d'une interaction conçue avec le formalisme LCM 41
Figure 7 : Deux fonctions non linéaires, potentielles et dissipatives, et leurs linéarisations par morceaux 42
Figure 8 : Manipulation d'une interaction LLM générique 43
Figure 9 : Caractéristique Force/Vitesse de l'interaction de frottement d'archet pour différentes valeurs de pression 44
Figure 10 : Interaction de plasticité 44
Figure 11 : Schéma fonctionnel de l'interaction de butée dans MIMESIS 44
Figure 12 : Le module frottement d'archet couplé au TGR (traduit de [Flo02]) 45
Figure 13 : Mode d'avancement du péristaltique 46
Figure 14 : Module piston 46
Figure 15 : Contrôle des paramètres via un système comportemental 50
Figure 16 : Contrôle des paramètres par un réseau de neurones 51
Figure 17 : Contrôle des paramètres par un réseau thermique 52
Figure 18 : Schéma général du contrôle paramétrique dans CORDIS ANIMA 54
Figure 19 : Interaction de butée 1D 54
Figure 20 : Exemple élémentaire de connexion d'un <MCP> 56
Figure 21 : Les 3 modules de mesure 57
Figure 22 : Principe de fonctionnement d'une module de mesure 57
Figure 23 : Une famille d'automates d'état définissant des LLM 59
Figure 24 : Effet du changement de signe d'une raideur sur l'interaction <ATR3> 59
Figure 25 : Illustration visuelle du phénomène de striction 60
Figure 26 : Modèle physique de striction, 4 étapes du phénomène 61
Figure 27 : Interaction de cohésion linéarisée 61
Figure 28 : Paramètres critiques de l'interaction <COH3> 62
Figure 29 : Contrôle du paramètre K_1 de <COH3> 63
Figure 30 : Contrôle conjoint des paramètres K_2 et S de <COH3> 63
Figure 31 : Le modèle de striction et son contrôleur 64
Figure 32 : Modèle de saut passif 66
Figure 33 : Modèle de saut et son modèle de contrôle 67
Figure 34 : Phase de contraction 67
Figure 35 : Phase d'extension 68
Figure 36 : Phase de saut (h est la hauteur du saut) 68

Figure 37 : Phase d'atterrissage et de contraction 68
Figure 38 : Relations entre variables phénoménologiques du saut et paramètres du contrôleur 69
Figure 39 : Butée visqueuse en phase d'approche 69
Figure 40 : Image de la bande freinante 70
Figure 41 : Automate d'état décrivant l'interaction de bande freinante 70
Figure 42 : Topologie du modèle "boules suspendues" 76
Figure 43 : Un graphe des labels et les noms engendrés 76
Figure 44 : Une représentation ensembliste du graphe des labels 76
Figure 45 : Interface graphique d'ANIMA 78
Figure 46 : Topologies de base générables avec ANIMA-OFF 78
Figure 47 : Quelques atomes volumiques 79
Figure 48 : Un mur, fracturé, s'écroule 80
Figure 49 : Une topologie pour un modèle de drapeau 80
Figure 50 : Fusion de deux modules <MAT> 81
Figure 51 : Les quatre étapes de la création et connexion d'un <LIA> 93
Figure 52 Les sous-réseaux A et B avant fusion 97
Figure 53 Le sous-réseau C résultant de la fusion 97
Figure 54 : Représentation graphique d'un <MCP> dans l'espace topologique 100
Figure 55 : Schéma pour une combinaison de modalité avec un point de contact. Exemple avec l'aspect Temps. 104
Figure 56 : Schéma général de la métaphore collaborative (d'après [Hut88]) 105
Figure 57 : Distance sémantique et distance référentielle (traduit de [HHN85]) 106
Figure 58 : Le tableau des fonctions 119
Figure 59 : Représentation fusion 120
Figure 60 : Représentation câblée 120
Figure 61 : Deux capsules et leurs connexions 121
Figure 62 : Représentation graphique d'un agglomérat de 300 masses 129
Figure 63 : Zoom sur l'agglomérat 129
Figure 64 : Agglomérat en interaction avec un pendule 130
Figure 65 : Zoom sur l'agglomérat en interaction avec le pendule 130
Figure 66 : La capsule et sa LPC sur l'espace topologique 133
Figure 67 : Capsule des interactions entre l'agglomérat et le pendule 133
Figure 68 : Connexion des capsules 134
Figure 69 : Un macro-MAS (en haut) et un macro-COH3 (en bas) sur l'espace topologique 135
Figure 70 : Agglomérat en interaction avec un pendule, version macro-modules 136
Figure 71 : Une maille et l'arbre de ses labels 138
Figure 72 : Deux mailles avant fusion 138
Figure 73 : Deux mailles fusionnées 138

Figure 74 : Deux lignes avant fusion 139
Figure 75 : Deux lignes fusionnées 139
Figure 76 : Trois mailles du drapeau avant fusion 140
Figure 77 : Connexion drapeau mât "directe" 142
Figure 78 : Connexion drapeau mât "croisée" 142
Figure 79 : Drapeau pendant l'opération de fusion 143
Figure 80 : Topologie cylindrique après fusion 143
Figure 81 : Vue n°1 de la topologie tore 144
Figure 82 : Vue n°2 de la topologie tore 144
Figure 83 : Graphe de Heawood 145
Figure 84 : Grille et échelle sur l'espace 3D 164
Figure 85 : Un rectangle et son axe médian en pointillés 167
Figure 86 : Différentes collections de sphères approchant un modèle 3D de lampe (tiré de [Hub96]) 168
Figure 87 : Un raffinement de l'axe médian par adaptation de l'échantillonnage (repris de [BO02]) 168
Figure 88 : Compositions des interactions <BUT> et <BUL> (tiré de [Cas10]) 169
Figure 89 : Interface de mapping 170
Figure 90 : Processus de modélisation et importation de données géométriques 171
Figure 91 : Une sphérisation d'un rectangle représentant un immeuble 172
Figure 92 : Un contour 2D et un pavage associé 173
Figure 93 : Transformation d'une scène géométrique via la méthode des cercles circonscrits 174
Figure 94 : Un contour (en vert), discrétisé (en rouge) et sphérisé (centres des sphères en bleu) 181
Figure 95 : Rebond sur un sol lisse : résultats avec trois méthodes de modélisation. Viscosité forte de l'interaction entre la bille et le sol (0.01) 182
Figure 96 : Rebond sur un sol lisse : résultats avec trois méthodes de modélisation. Viscosité faible de l'interaction entre la bille et le sol (0.001) 183

Table des équations

Équation 1 : Formule du module d'une force visco-élastique 29
Équation 2 : Schéma de discrétisation de CORDIS-ANIMA 29
Équation 3 : Algorithme des modules <MAT> 30
Équation 4 : Algorithme des modules <REF> 30
Équation 5 : Paramètres algorithmiques et paramètres physiques 30
Équation 6 : Fonction potentielle linéaire par morceaux 42
Équation 7 : Fonction dissipative linéaire par morceaux 42
Équation 8 : Expression de K_2 en fonction des autres paramètres de <COH3> et de S_0. 65
Équation 9 : Définition d'une surface isopotentielle 177
Équation 10 : Fonction distance privilégiant l'axe X 177

Chapitre I. Un langage pour l'Art du mouvement

1. Image et mouvement

Les images sont au cœur de nombreuses activités, en particulier artistiques, qui l'utilisent pour leur puissance d'évocation mais aussi pour la questionner. En effet la nature de l'image est problématique, sa durée de vie, son support, ses composants sont autant de questions que l'on peut légitimement se poser à son propos. Nous prenons le parti pris de définir une image comme toute activité cognitive reliée à la perception visuelle. Nous englobons ainsi dans le concept d'image à la fois l'imagination et la perception. Annie Luciani, dans son cours sur l'art du mouvement, distingue trois attributs principaux dans une image :

1. Les attributs morphologiques qualifient l'organisation de l'espace, la forme des objets.
2. Les attributs visuels sont propres à la nature optique de l'image. Ce sont les propriétés d'interaction des objets avec la lumière qui sont concernés : transparence, brillance, couleur, etc...
3. Les attributs cinétiques sont les attributs propres à l'origine mécanique des phénomènes présents dans une image. Des attributs comme la lourdeur ou la viscosité peuvent ainsi être identifiés dans une image.

Ce dernier attribut met en avant une qualité essentielle de l'image : sa dynamique. L'image est la partie visible du monde, ce monde est animé, en perpétuel mouvement. Quelle que soit la nature du mouvement, son échelle, il est toujours présent dans l'image. L'image est issue d'une dynamique et elle exprime cette origine. Il faut savoir retrouver et interpréter cette expression pour comprendre l'image. L'attribut morphologique découle en partie du mouvement. Pour Paul Klee, la forme naît du mouvement. Il est l'un des premiers à donner une place centrale au mouvement dans les arts plastiques, qui auparavant étaient des arts essentiellement statiques. Le mouvement n'avait de place que dans des arts spécialisés comme le théâtre d'ombres. On identifie deux approches antagonistes dans la création artistique visuelle : faire naître la forme à partir du mouvement et animer la forme. Choisir la première approche, c'est prendre le parti pris du mouvement, le prendre comme objet d'intérêt, objet artistique en soi, pour soi. Les autres composantes de l'image, morphologique et visuelle, ne sont alors qu'au service de la révélation du mouvement.

Le mouvement intervient dans de nombreux arts, mais mis à part la musique, il n'y tient pas une place centrale. Il n'existerait donc pas d'art visuel dédié au mouvement. On pourrait objecter que l'objet principal de la danse est le mouvement, cependant son medium n'est pas l'image mais d'abord le corps. De son côté, le cinéma d'animation a

plus souvent été un art du récit qu'un véritable Art du Mouvement. L'art cinétique est certainement ce qui se rapproche le plus de l'art du mouvement, ce courant artistique est en effet fondé sur l'esthétique du mouvement, et s'exprime principalement à travers des sculptures mobiles et des illusions d'optique. L'art cinétique reste cependant un simple courant dans l'art contemporain.

Les technologies ont été, et sont encore, un facteur de changement important dans l'art. Le groupe Image de l'ACROE-ICA, sous l'impulsion d'Annie Luciani, fonde ses travaux sur l'idée que l'ordinateur, dans le domaine du mouvement visuel, pourrait notamment changer sa vision cinématique en faveur d'une vision dynamique, et constituer ainsi un outil de choix pour des artistes désireux d'expérimenter et de travailler le mouvement comme objet d'art en soi.

2.Image animée et ordinateur

2.1. Une brève histoire de l'informatique graphique

L'informatique graphique fait ses débuts dans les années 60 avec le *sketchpad* de Sutherlands [Sut63]. Ce travail éminemment précurseur permet à un utilisateur de dessiner et manipuler des formes simples sur un écran à affichage vectoriel, il préfigure également la naissance d'un domaine : l'interaction humain-machine. Les années 70 voient la mise en avant des caractères morphologiques avec la mise au point de modèles géométriques 2D et 3D, c'est le début de la synthèse d'image en trois dimensions. Les propriétés optiques de l'image commencent également à être maitrisés, un domaine à part entière de l'informatique graphique lui est consacré, le rendu. On relève notamment l'apport d'Henri Gouraud [Gou71] avec la simulation efficace des ombres, le fameux "Gouraud Shading", qui rend pour la première fois crédibles des rendus de scènes tridimensionnelles. Avec la puissance croissante des machines, le rendu se développe avec l'avènement du lancer de rayon. Parallèlement, la modélisation géométrique se diversifie avec par exemple les surfaces implicites [Bli82]. Dans cette évolution rapide de ce nouveau champ de recherche, la dynamique reste le parent pauvre de l'informatique graphique. Jusqu'à la fin des années 1980, le domaine reste cantonné à l'interpolation entre images clés et aux fonctions d'évolutions prédéfinies. On pense notamment aux travaux pionniers de Peter Foldes et Nestor Burtnyk qui développent dans les années 1970 l'un des premiers systèmes d'animation par ordinateur [BW76]. Nous développons dans la suite comment l'animation par ordinateur s'est peu à peu diversifiée, notamment en changeant complètement de paradigme avec la notion de modèle générateur.

2.2. Modèles descriptifs et modèles générateurs

Au début des années 80, Luciani [LC84] propose le concept de *modèles générateurs* de mouvement, en dualité avec les *modèles descriptifs*. Ces derniers s'intéressent au phénomène, qu'ils décrivent à travers des algorithmes à temps explicite. A l'inverse, les modèles générateurs s'intéressent à la cause du phénomène. Celle-ci est décrite par des algorithmes à temps implicite, le modèle est donc *simulé* pour produire le phénomène. Reynolds [Rey82, Rey87] introduit très tôt le concept d'entités capables de réagir en fonction de données provenant de l'environnement suivant un comportement décrit par

l'utilisateur dans un langage procédural. Notons que la modélisation procédurale en informatique graphique n'a pas commencé avec la dynamique mais avec la géométrie. Les travaux de Mandelbrot sur les fractales [Man79] ont ainsi impulsé une nouvelle branche de la modélisation géométrique, les L-Systems [Smi84] en constituent un exemple remarquable. Mais revenons aux systèmes générateurs de mouvements. Les systèmes de particules de Reeves [Ree83] sont souvent cités comme exemple pionnier, mais ils ne sont pas initialement des systèmes générateurs, ce sont des modèles descriptifs donnant les trajectoires, et d'autres propriétés à temps explicite, des particules. Reeves mentionne cependant la possibilité de définir la dynamique des particules à l'aide d'équations différentielles. À la même époque, Luciani et Cadoz introduisent le modèle physique particulaire pour l'animation avec le formalisme CORDIS-ANIMA [LC84] sur lequel nous reviendrons plus longuement. Ce type de modèle permet à l'époque d'obtenir des mouvements de surface déformable et des marionnettes manipulées en temps réel. Les modèles physiques constituent un pan important des modèles générateurs, ils se développeront notablement dans la deuxième moitié des années 1980 avec les travaux de Miller [Mil88], de Terzopoulos [TPF89] et de Tonnesen [Ton91] entre autres. Les automates cellulaires logiques sont un autre type de modèle générateur, dans lequel des règles logiques déterminent l'évolution du système. L'exemple le plus célèbre est le jeu de la vie de Conway. Parallèlement, les modèles descriptifs évoluent également avec l'apparition de la capture de mouvement ("motion capture") et de la cinématique inverse, deux techniques très utilisées pour l'animation d'humanoïdes [CCP82, GM85].

Le développement des modèles générateurs, dont font partie les modèles physiques, a engendré un intérêt croissant pour les phénomènes émergents. Ces modèles sont apparus comme un moyen économique, en termes de calcul mais aussi au regard de la taille et de la complexité des algorithmes, pour produire des phénomènes riches et complexes.

2.3. Diversité des modèles physiques

Au cours de l'histoire des modèles physiques pour l'animation, différents types de modèles ont vu le jour, chacun inspiré par des champs différents des sciences physiques, et plus exactement de la mécanique. La mécanique du solide est l'un des premiers modèles transposé à l'informatique graphique. En effet, on peut considérer ce domaine comme assez proche de la géométrie, car les éléments de base sont des volumes infiniment rigides. En leur associant un moment d'inertie et un centre de gravité, et en leur appliquant des forces et des couples, ces volumes peuvent être mis en mouvement [AG85]. Les forces proviennent majoritairement des collisions entres solides. La détection de collisions constitue ainsi la principale difficulté de ces modèles, à chaque pas de simulation il faut déterminer toutes les interpénétrations entre les volumes de la scène. Ce problème purement géométrique a fait l'objet d'une littérature abondante [TKH+04].

Les modèles masses-ressorts ont connu un grand succès pour la modélisation d'objets déformables à topologie fixe, comme les tissus [Pro95], les solides "mous" [TPF89, BC00]. Dans ces modèles, la préséance de la géométrie est toujours très forte. En effet, le modèle masse-ressort se déduit du modèle géométrique. Dans le cas d'un modèle

surfacique, on fait globalement correspondre les masses aux sommets et les ressorts aux arêtes du modèle polygonal. Dans le cas d'un modèle volumique, on opère une tétrahédrisation ou une autre transformation du volume en volumes élémentaires. Les systèmes masses-ressorts ont aussi été utilisés pour l'animation de créatures virtuelles [Mil88, TT94]. Notons que Miller ne parle pas de systèmes masses-ressorts mais simplement de modèles basés sur la physique newtonienne. Aujourd'hui, les systèmes masses-ressorts sont directement associés à la modélisation d'objets faiblement déformables.

La mécanique des corps déformables a inspiré toute une série de méthodes. Terzopoulos et al. modélisent ainsi des objets élastiques en se basant sur la loi de Hook [TPB87], la méthode est ensuite élargie pour modéliser des comportements plus complexes comme la plasticité et les fractures [TF88]. La mécanique des milieux continus a quant à elle inspirés de nombreux autres travaux qui traitent par exemple de la résolution de l'équation de Navier-Stokes pour la synthèse d'images de fluides. Ils se distinguent par le type de système de représentation choisi. La représentation lagrangienne met en œuvre des particules [FM95], la représentation eulérienne discrétise l'espace et définit des champs de vitesse et de densité en chaque cellule de la grille [Sta99]. Héritée de l'astrophysique [GM77], la méthode des *smoothed particles hydrodynamics (SPH)* est une méthode hybridant représentations lagrangienne et eulériennes. Introduite en informatique graphique dans [DC96], elle a connu un succès important pour la modélisation des liquides [MCG03] et est aujourd'hui intégrée dans le moteur physique temps réel de Nvidia, PhysX [Phy].

Ces trois types de modèles sont propres à des classes de phénomènes dynamiques bien circonscrites. Ce ne sont donc pas des modèles génériques. La combinaison de ces modèles est un champ de recherche ouvert, exploré notamment par la bibliothèque SOFA [ACF+07]. Malgré cette possible combinaison, ces systèmes de modélisation ne peuvent être considérés comme de véritables langages de modélisation, les modèles développés sont très particuliers à un effet et sont construits sur la base d'une géométrie existante (mis à part les SPH qui sont une approche dite "meshless", sans maillage).

Greenspan, à contre-courant de la méthode des éléments finis, remet en cause la représentation continue des phénomènes physiques. Son raisonnement se base sur le fait que les modèles numériques sont nécessairement discrets, de même que les mesures qui les alimentent. Il avance de plus la difficulté d'introduction de non-linéarités dans les formalismes continus, ainsi que le fait que les phénomènes physiques émergent de l'interaction entre particules [Gre73]. Il démontre la pertinence de son approche en modélisant un grand nombre de phénomènes différents dans son ouvrage "Particle Modeling" [Gre97]. Ces modèles seront une source d'inspiration en synthèse d'image pour Miller et Pearce [MP89] ainsi que Tonnesen [Ton91], qui modélisent ainsi principalement des fluides visqueux. Parallèlement, Cadoz et al. introduisent en 1984 les *réseaux* masses-interactions [CLF84]. Ces réseaux, aussi nommés plus tard réseaux newtoniens par Luciani [Luc04], proposent une réécriture du modèle newtonien sous la forme de modules interconnectés de masses et d'interactions. Dans la suite de cet exposé, nous les appellerons indifféremment modèles masses-interactions ou réseaux masses-interactions. Avec cette réécriture, les auteurs définissent un langage de modélisation et de simulation, CORDIS-ANIMA [CLF84, LJF+91, CLF93].

Le choix de la physique, et plus particulièrement de la physique newtonienne sous la forme de réseau, est issu d'une réflexion profonde sur les besoins du praticien de l'art du mouvement. Nous détaillons dans la prochaine partie les origines de ce choix et les fondements de CORDIS-ANIMA.

3. CORDIS-ANIMA

3.1. Pourquoi et comment concevoir du mouvement

Dans leurs travaux sur le mouvement visuel, Annie Luciani et l'équipe Image du groupe ACROE-ICA se positionnent par rapport à une problématique bien précise : la recherche du mouvement juste avec l'outil informatique dans une démarche artistique. L'outil informatique s'intègre donc dans un processus de création centrée sur le mouvement. Ce processus cyclique, proposé par Annie Luciani et résumé en figure 1, montre cinq étapes. La première, certainement la plus difficile, consiste à passer de l'observation d'un phénomène dynamique à son abstraction. Cet exercice de notre imagination dynamique nous oblige à formuler ce qui est constitutif du phénomène que l'on désire modéliser, quelles sont ses caractéristiques morpho-dynamiques qui nous permettent de l'identifier mais aussi de provoquer une émotion. On est dans une optique différente du physicien, qui veut connaître tout avec l'approximation la plus fine. Nous avons l'objectif d'aboutir à un modèle *juste*, qui représente bien les phénomènes choisis pour notre perception, notre première exigence est la *crédibilité*. La deuxième étape est la fabrication du modèle. Elle nécessite l'existence d'un système de représentation pour son élaboration, un modeleur. Ce modeleur doit être basé sur un système générateur car la complexité des phénomènes que l'on souhaite modéliser est inatteignable à l'aide de modèles descriptifs. Les opérateurs de la physique et plus précisément de la mécanique, science du mouvement par excellence, apparaissent comme le choix le plus naturel, le plus à même de décrire les relations en jeu dans les phénomènes dynamiques. La troisième étape consiste à simuler ce modèle, permettant ainsi d'enchaîner sur les étapes 4 et 5, étapes de validation par rapport à l'abstraction et par rapport à l'observation. L'étape de simulation doit être complètement maitrisée par le modélisateur, sous peine de fausser complètement les étapes de validation. Cette maitrise signifie une identité complète entre le système de modélisation et le système de simulation. Le langage de modélisation doit nécessairement être un langage de simulation, les éléments du langage sont donc aussi des algorithmes, connus par le modélisateur. Le calcul effectué en aval de la modélisation est indissociable du modèle : les discrétisations temporelles et spatiales, le pas de simulation, le codage des nombres, sont autant de paramètres, usuellement réservés au numéricien, qui font partie intégrante du modèle.

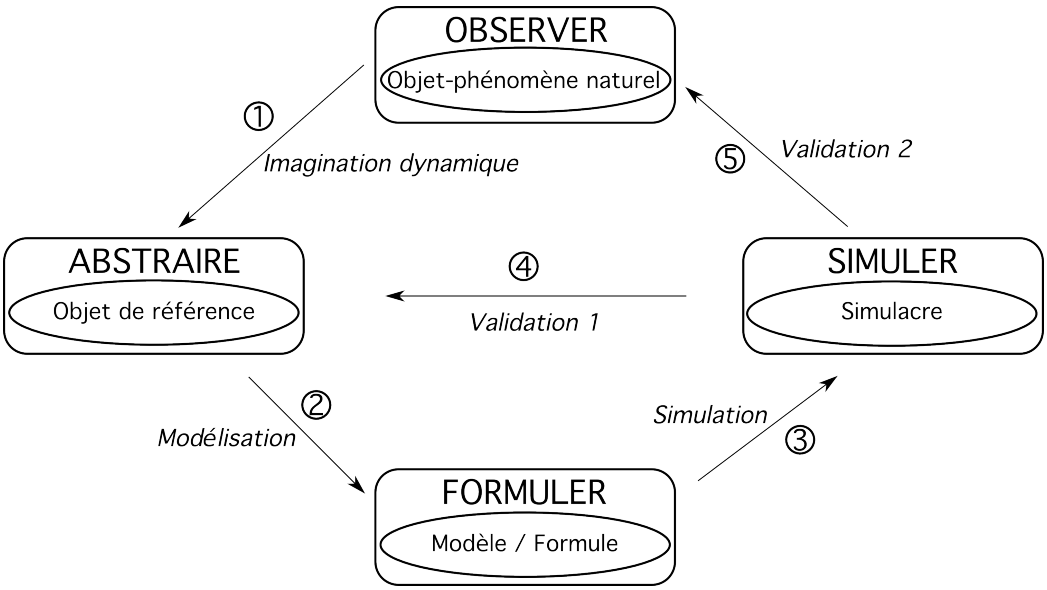


Figure 1 : Processus de création pour l'Art du Mouvement

Quels principes physiques pour constituer la colonne vertébrale de ce langage ? Quel mode d'organisation du langage pour favoriser puissance d'expression et maniabilité ? Nous tentons d'aborder ces points en montrant comment CORDIS-ANIMA s'inspire des principes newtoniens et propose un langage de modélisation et de simulation adapté à notre objectif.

3.2. De Newton aux réseaux masses-interactions

Faire appel à la physique pour produire du mouvement suppose 1) de disposer d'un langage de modélisation, permettant la conception de modèles divers 2) de disposer de moyens de calculer le modèle physique. Nous montrons ici comment les concepteurs de CORDIS-ANIMA ont défini le principe de réseaux masses-interactions en se basant sur les lois du mouvement de Newton.

Les trois principes de la dynamique de Newton constituent la base de la mécanique classique. Le concept de force a permis d'exprimer mathématiquement les lois du mouvement. En effet, en introduisant cette cause virtuelle du mouvement, Newton a inventé une abstraction à même de rendre compte des mouvements observés. La force, variable vectorielle, permet ainsi de quantifier l'influence que les corps exercent les uns sur les autres. Revenons en au trois lois de Newton. Nous laissons de côté la première, qui postule l'existence d'un repère galiléen, et intéressons nous tout d'abord à la troisième : tout corps A exerçant une force sur un corps B subit une force d'intensité égale, de même direction mais de sens opposé, exercée par le corps B. Ce principe, également connu sous le nom de principe d'action-réaction, met en avant le concept fondamental d'interaction. Toute force exercée sur un corps provient d'une interaction entre ce corps et un autre, il n'y a pas de force exercée *ex nihilo* ! En vis-à-vis, le second principe vise l'élément matériel et définit comment il est influencé par les différentes

forces qui s'exercent sur lui, c'est le principe d'inertie : l'accélération subie par ce corps dans un référentiel galiléen est proportionnelle à la résultante des forces qu'il subit, et inversement proportionnelle à sa masse m. Autrement dit, plus un objet est massif, moins il est influençable.

Les principes de Newton ont la caractéristique intéressante d'être opératoires, différemment de la mécanique de Lagrange, parfaitement équivalente mais analytique. L'aspect opératoire des principes de Newton se manifeste de la manière suivante : le deuxième principe, permet de calculer l'accélération d'un corps connaissant les forces qui lui sont appliquées. On déduit ainsi la position au temps $t+\delta t$ connaissant la position et la vitesse au temps t. Le caractère générateur du modèle physique apparaît clairement ici, l'équation résumant le deuxième principe étant une équation à temps implicite. En vis à vis, le troisième principe donne une contrainte forte pour le calcul des forces. Nous avons donc des composants matériels, qui opèrent le second principe, et des composants d'interaction, qui respectent le troisième principe. Ces composants sont connectés entre eux pour former un réseau dans lequel circulent des variables extensives, les positions et vitesses des masses, et des variables intensives, les forces. Les variables extensives sont localisées et diffusantes, alors que les variables intensives sont circulantes et fusionnantes. Cette définition, associée aux deuxième et troisième principes, donne les règles de connexion du réseau. Une interaction est connectée à exactement deux éléments matériels, un élément matériel est connecté à un nombre quelconque d'interactions. Ainsi les forces émises par des interactions fusionnent sur un élément matériel, alors que les éléments matériels diffusent leurs positions aux interactions qui leur sont connectés. La première communication permet à l'élément matériel d'opérer le deuxième principe, la deuxième permet aux interactions de calculer une force en fonction des positions et vitesses des éléments matériels connectés. On constate que dans ce modèle de communication théorique, la causalité est atemporelle, c'est à dire que les forces à l'instant t déterminent l'accélération à l'instant t et que les positions et vitesses à l'instant t déterminent les forces à l'instant t. Nous décrivons donc un modèle idéal dans lequel communications et calculs sont instantanés. Cette causalité atemporelle est forcément remise en cause lorsqu'un tel modèle est mis en œuvre sur un calculateur réel.

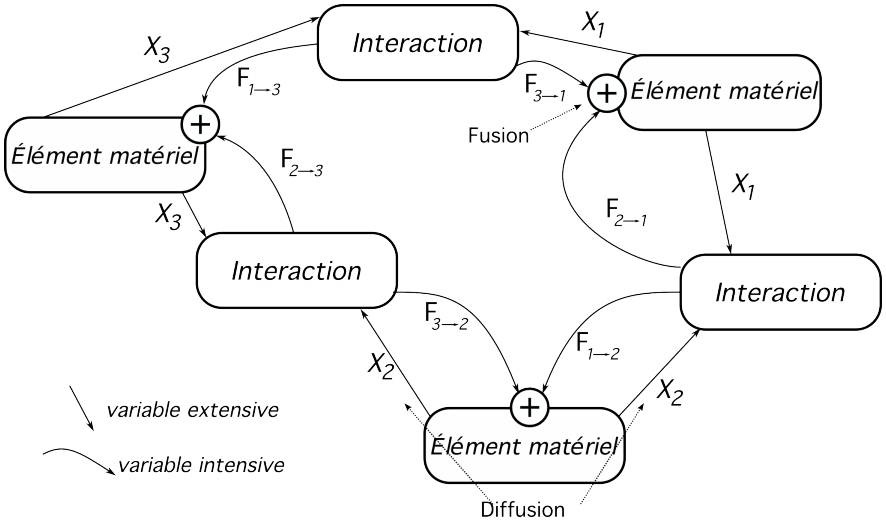


Figure 2 : Un réseau masses-interactions

La figure 2 illustre un réseau masses-interactions élémentaire avec les circulations des différentes variables. Ce réseau exhibe une topologie particulière, tous les éléments matériels sont en interaction les uns avec les autres, le formalisme réseau permet d'autres topologies et c'est d'ailleurs là une de ses forces.

Nous avons présenté les principes élémentaires des réseaux masses-interactions. Nous montrons à présent comment CORDIS-ANIMA constitue un formalisme dédié à la modélisation et à la simulation de réseaux masses-interactions sur des ordinateurs, c'est à dire sur des machines discrètes.

3.3. CORDIS-ANIMA, un formalisme pour les réseaux masses-interactions

CORDIS-ANIMA est basé sur la notion de module. Proche de la notion de composant développé dans le paragraphe précédent, le module est davantage ancré dans le monde des calculateurs. Un module est ainsi constitué de trois éléments :

- un ensemble de points de communication,
- un état,
- un algorithme (et ses paramètres).

De manière générale, l'algorithme calcule l'état suivant en fonction de ses paramètres, des variables qu'il reçoit via les points de communication, et de la mémoire des états précédents. L'état est décrit par un ensemble de variables extensives et intensives, c'est-à-dire des vecteurs positions et forces. Les points de communication permettent aux modules d'échanger des variables. Ils sont de deux types : les points M, qui envoient des positions et récupèrent des forces, et les points L, qui envoient des forces et récupèrent des positions. Les points de communication sont dimensionnés. En effet CORDIS-ANIMA permet de modéliser des objets dans des espaces de dimension 1,2 ou 3. Deux classes de modules installent CORDIS-ANIMA comme un formalisme pour les réseaux masses-interactions :

1) les modules <MAT> élémentaires présentent un point M et représentent le principe d'inertie.

2) les modules <LIA> élémentaires présentent deux points L et représentent des interactions respectant le 3ème principe. Les forces émises par un module <LIA> sont des forces axiales, c'est-à-dire que la direction des vecteurs force est définie par la droite passant par les deux points matériels connectés.

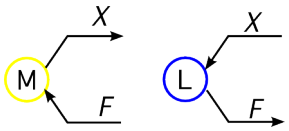


Figure 3 : Points M et L

Les règles de connexion de CORDIS-ANIMA se déduisent ainsi des règles de connexion d'un réseau masses-interactions : un point M est connectable à un nombre quelconque de points L, un point L est connectable à un et un seul point M. Mais ces règles sont aussi la conséquence de la nature des variables échangées. Une position est diffusante, donc un point M diffuse une position à différents modules en se connectant à plusieurs points L. Une force est fusionnante, donc un point M fusionne les forces qu'il reçoit des différents points L. En revanche, comme les positions ne sont pas fusionnantes, un point L ne se connecte qu'à un unique point M.

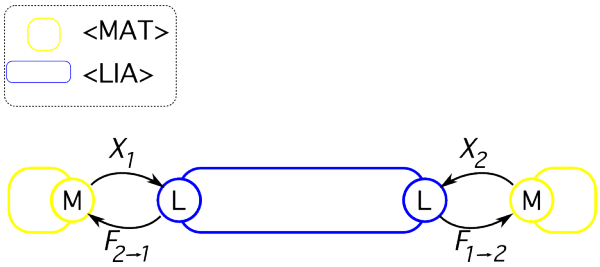


Figure 4 : Un réseau CORDIS-ANIMA élémentaire

La définition très ouverte des modules et la notion puissante de point de communication ont donné lieu à la spécification de modules non élémentaires. Une première extension des deux familles de modules consiste à définir des macro-<MAT> et des macro-<LIA>. Ces modules présentent le même algorithme que les modules élémentaires, mais qui opère sur un grand nombre N de variables d'état et les communique via un grand nombre de points de communication, N points M pour les macro-<MAT> et 2N points L pour les macro-<LIA>. C'est une première complexité qui est ainsi ouverte, la complexité en taille des réseaux CORDIS-ANIMA. D'autres types de modules peuvent être définis, nous aborderons plus en détail ce point dans le deuxième chapitre.

Mais avant cet élargissement de la palette de modules, CORDIS-ANIMA montre sa généricité à travers les différents composants d'interactions, modules <LIA>, qu'il permet de définir. Ces modules sont tous basés sur l'interaction visco-élastique, ou ressort-frottement <REF>, dont l'expression continue du module de la force est donnée en équation 1. Les différentes variantes consistent à introduire des non-linéarités au sein de l'algorithme de calcul de forces en faisant intervenir des conditions sur les distances et vitesses relatives des <MAT> connectés. La diversité des composants d'interactions, ainsi que leur capacité à intégrer des non-linéarités complexes, est une des qualités premières de CORDIS-ANIMA. Notre objectif étant d'ouvrir la voie vers davantage de complexités, nous nous sommes particulièrement intéressés à cet aspect de CORDIS-ANIMA. Nous analysons ainsi dans le chapitre II, consacré entièrement aux non linéarités, les différents types d'interactions non linéaires. Nous montrons ainsi comment des phénomènes complexes ont pu être obtenus grâce à une collection réduite d'interactions non linéaires.

$$F(t) = k(d(t) - L) + z \times v_r(t)$$

Équation 1 : Formule du module d'une force visco-élastique

k est le coefficient d'élasticité
z est le coefficient de viscosité
d est la distance entre les deux masses
v_r est leur vitesse relative

CORDIS-ANIMA est donc un langage de modélisation générique pour les réseaux masses-interactions, basé sur des composants simples et évolutifs. Examinons à présent comment ce langage constitue également un langage de simulation.

3.4. Simulation et algorithmes

CORDIS-ANIMA est un système de modélisation et de simulation à temps discret. Un pas de simulation correspond à l'exécution de quatre phases :

1) Exécution de l'algorithme de tous les modules <MAT>
2) Communication points M vers points L
3) Exécution de l'algorithme de tous les modules <LIA>
4) Communication points L vers points M.

En dehors du temps de calcul nécessaire à l'exécution de ces quatre phases, il faut donner une *durée* au pas de simulation, c'est à dire combien de temps se déroule entre deux valeurs consécutives d'une même variable d'état. Nous introduisons donc F_s, la fréquence de simulation, inverse de cette durée. Le choix de F_s est déterminant dans le processus de simulation modélisation, car il est lié à la fréquence de coupure F_c des phénomènes que l'on veut modéliser, selon la même loi que Shannon pour l'échantillonnage : $F_s > 2*F_c$. En effet, la fréquence de simulation était déjà présente implicitement dans les algorithmes des modules à travers les vitesses et accélérations. Parmi tous les nombreux schémas de discrétisation possibles, le formalisme CORDIS-ANIMA est basé sur le schéma de discrétisation à vitesse et accélération retardée, explicité dans l'équation 2.

$$V_n = \frac{X_n - X_{n-1}}{T_s}$$
$$\Gamma_n = \frac{V_n - V_{n-1}}{T_s} = \frac{X_n - 2X_{n-1} + X_{n-2}}{T_s^2}$$
$$X_n = X(t_0 + nT_s)$$

Équation 2 : Schéma de discrétisation de CORDIS-ANIMA

De plus, pour assurer la calculabilité des algorithmes, CORDIS-ANIMA implante les schémas de calcul suivants décrits dans l'équation 3. On remarque que les forces au pas de simulation n-1 déterminent l'accélération des masses au pas n. La simulation induit une causalité temporelle absente du second principe, comme nous le soulignions dans le paragraphe précédent.

$$m \times \Gamma_n = \sum_i F_{n-1}$$

$$X_n = 2X_{n-1} - X_{n-2} + \frac{T_s^2}{m} \sum_i F_{n-1}$$

Équation 3 : Algorithme des modules <MAT>

De la même manière, nous pouvons aussi expliciter l'algorithme des modules <REF>, déduit de la formulation continue, avec l'équation 4.

$$F_n = k(d_n - L) + z\frac{d_n - d_{n-1}}{T_s}$$

Équation 4 : Algorithme des modules <REF>

On identifie les paramètres principaux des modules CORDIS-ANIMA : la masse (ou inertie) *m*, la raideur (ou élasticité) *k*, et la viscosité *z*. Ces paramètres sont indépendants de la fréquence de simulation, nous les appelons paramètres physiques. En effet, par commodité, nous avons posé T_s=1, d'où l'obligation d'introduire la notion de paramètres algorithmiques, dépendants de la fréquence de simulation, comme le montre l'équation 5. En plus des paramètres physiques de raideur, élasticité et inertie, il faut également compter ce que nous appelons les paramètres spatiaux, c'est à dire les longueurs au repos ainsi que tous les seuils en distance et en vitesse relatifs aux interactions non linéaires.

$$M = \frac{m}{T_s^2}; \quad K = k; \quad Z = \frac{z}{T_s}$$

Équation 5 : Paramètres algorithmiques et paramètres physiques

Les algorithmes de CORDIS-ANIMA sont simples, la complexité émerge de la manière de les agencer et de les paramétrer. La simplicité est un atout important, le modélisateur peut ainsi les maitriser sans connaissances préalables majeures. Ces algorithmes offrent une base de paramètres compréhensibles, proches de notre expérience physique du quotidien, l'inertie, la raideur et la viscosité sont en effet des notions que l'on peut expérimenter dans la vie de tous les jours. Cette simplicité peut être critiquée par rapport à des critères mathématiques, qui comparent ces algorithmes à la formulation continue. Cependant, comme nous le rappelions en décrivant le processus de modélisation, notre objectif n'est pas de proposer un formalisme qui soit le plus proche des équations continues mais un outil de modélisation permettant d'obtenir des résultats crédibles.

3.5. Un formalisme tourné vers le temps réel et la multisensorialité

CORDIS-ANIMA a été initialement créé dans l'objectif de produire des instruments virtuels avec lesquels nous pourrions interagir avec la même qualité que des instruments réels.

C'est donc un formalisme orienté vers la multisensorialité, les objets physiques virtuels créés avec CORDIS-ANIMA peuvent produire du son et de l'image avec une interaction gestuelle comparable à l'interaction que l'on peut avoir avec des instruments réels. Or le canal gestuel, contrairement au canal acoustique et au canal visuel, est bidirectionnel, on manipule et on sent dans le même temps. Ainsi, un modèle CORDIS-ANIMA peut être relié au monde réel par l'intermédiaire d'un transducteur gestuel rétroactif (TGR), des points matériels du monde réel étant en interaction avec des points matériels virtuels. Le formalisme a donc été défini pour satisfaire deux contraintes fortes :

- La technologie des transducteurs impose la discrétisation et la finitude des communications entre l'ordinateur et l'univers réel.
- Le temps réel : la bande passante des phénomènes que l'on désire simuler étant haute, les algorithmes doivent être économes en mémoire et en temps de calcul.

La première contrainte se retrouve dans la notion de point de communication, qui permet de formaliser efficacement la communication avec le monde réel. En effet deux transducteurs sont à l'action dans le TGR, des capteurs de positions et des moteurs, ce qui permet de retrouver une sorte de point M réel, émettant une position et recevant une force. La contrainte temps réel est, en partie, satisfaite grâce d'une part à la simplicité des algorithmes et d'autre part grâce à leur capacité à être parallélisés facilement. Bien entendu, la contrainte temps réel nécessite la maitrise d'autres éléments, comme la réactivité des entrées/sorties et la synchronisation. Ces contraintes spécifiques ont abouti à deux versions du simulateur CORDIS-ANIMA, une version temps réel, attachée à une architecture matérielle, TELLURIS [Cas10, Gir99], et une version temps différé, CORDIS-OFF, qui est une bibliothèque multi-plateforme en langage C.

CORDIS-ANIMA est un formalisme multi-sensoriel dans le sens où il permet de modéliser des objets virtuels produisant du mouvement visuel, du son et de l'interaction gestuelle. La production sonore peut être prise comme une activité de modélisation à part entière, c'est le parti pris de GENESIS [CC02, CCA+09], interface de modélisation de réseaux CORDIS-ANIMA pour la création musicale. Des espaces de dimension 1 sont ainsi utilisés pour modéliser des objets produisant des vibrations acoustiques. GENESIS est un logiciel de création musicale à part entière, qui mêle création de timbres avec la modélisation de structures vibrantes, et composition, avec la modélisation d'instrumentistes virtuels, toujours selon le principe des réseaux masses-interactions.

4. MIMESIS

4.1. Historique : du langage à l'environnement interactif

Quel langage pour l'Art du Mouvement ? Les modèles générateurs et plus précisément les modèles physiques sont adaptés à exprimer des dynamiques complexes. Le groupe ACROE-ICA crée ainsi le formalisme masses-interactions dont la généricité et de la modularité en font un langage maniable, compréhensible et puissant. Avec CORDIS-ANIMA, le laboratoire a pu expérimenter le formalisme en le mettant à l'épreuve de la modélisation de nombreux et divers phénomènes dynamiques. Le choix s'est révélé payant, puisque 30 ans de travaux avec ce formalisme ont permis d'aboutir à une grande variété de modèles, des marionnettes manipulées en temps réel jusqu'aux turbulences (fumée [HLV96], foules [HLT+03]) en passant par des phénomènes de fracture [LG97], de striction ou encore des mouvements dansés [HL05]. Cette démarche est d'ailleurs validée à l'extérieur dès 1990 avec le premier prix du Video Show d'Eurographics avec le film "Modèles physiques et animation". D'autres travaux scientifiques et artistiques ont suivi, montrant une qualité dynamique toujours présente, conséquence de la cohérence physique des modèles. Après un premier essai limité par les possibilités techniques de l'époque, ANIMA [Raz86], le besoin d'un environnement de modélisation interactif puissant se fait sentir. En effet, si CORDIS-ANIMA présente des algorithmes basiques, la complexité des mouvements qu'il permet de produire n'est pas dans ses algorithmes mais dans le processus de *modélisation*. La difficulté pour le praticien de l'art du mouvement est de trouver le bon modèle pour l'effet qu'il souhaite, comme nous le rappelions au §3.1. Un environnement dédié à l'activité de modélisation est donc indispensable pour assister le praticien dans sa tâche. Le projet MIMESIS a ainsi commencé, lançant les bases de ce que doit être un environnement interactif pour la modélisation de mouvements visuels avec CORDIS-ANIMA. Ce projet, lancé en 1999, est validé par les ateliers RICA en 2001, malgré une interface encore peu stable, MIMESIS 4. Ainsi des étudiants en école d'Art ont pu expérimenter le modèle physique comme outil de création et ont abouti à des résultats intéressants. Cette validation s'est poursuivie avec la stabilisation du logiciel, sa diffusion à l'EESI, Ecole Européenne Supérieure de l'Image à Poitiers, les travaux de doctorat de Chi Min Hsieh sur les verbes de la danse [Hsi07], ainsi que d'autres expériences pédagogiques, scientifiques et artistiques. La thèse de Matthieu Evrard sur MIMESIS [Evr09] marque la fin de la première phase du projet MIMESIS.

4.2. Processus de modélisation avec CORDIS-ANIMA

D'une manière générale et par principe de la modélisation, un modèle physique effectif, prêt à être simulé, ne se réduit pas à l'écriture formelle ou algorithmique de l'équation qui régit son comportement. Mais la phase de modélisation inclut de manière structurelle une détermination fine des paramètres physiques et des conditions initiales, activités à part entière de la phase de modélisation. C'est pourquoi dans les logiciels de type MIMESIS ou GENESIS, le processus de modélisation donné à pratiquer à l'utilisateur est structuré en cinq phases bien clairement définies :

- La phase dite PSQL, pour PréStructuration QuaLitative, dans laquelle est définie la structure du réseau masses-interactions et qui correspond à la représentation formelle ;

- La phase PSQN, pour PréStructuration QuaNtitative dans laquelle se définissent les valeurs quantitatives de tous les paramètres physiques pour chacun des modules masses ou interactions que comporte le réseau ;

- La phase « conditions initiales » dans laquelle sont définies toutes les conditions initiales, nécessaires au démarrage de la simulation de tous les modules ;

- La phase de simulation ou plus généralement de « jeu » lorsque la simulation fait intervenir des variables physiques intensives ou extensives, en provenance ou à destination de l'extérieur ;

- La phase d'habillage consiste à donner une extension spatiale aux points matériels, par exemple en leur affectant des formes géométriques. Cette phase n'a aucune incidence sur le modèle physique.

A l'heure actuelle, après la phase de preuve du concept de la modélisation masses-interactions par le développement à la fois de nombreux modèles et de premiers prototypes du logiciel de modélisation MIMESIS, le point essentiel pour donner toute son ampleur à ce procédé de modélisation puissant est de donner à l'utilisateur le moyen de franchir un certain seuil de complexité des modèles.

Dans cet objectif, chacune de ces phases se doit d'être bien identifiée dans la mesure où elle présente des propriétés et des difficultés spécifiques dans le processus de modélisation. Ces propriétés doivent donc se retrouver dans l'interface de manière à ce que le modélisateur puisse travailler dans des conditions optimales chacune de ces phases. Il accèdera ainsi à la complexité en saisissant et en exploitant les particularités et difficultés de chacune d'entre elle.

Ces phases ne sont cependant pas indépendantes, il existe des facteurs communs importants ayant trait à complexité. La taille du réseau, c'est-à-dire son nombre de modules, est un premier facteur de complexité. Il influe sur PSQL, car définir une topologie comprenant un grand nombre de modules est complexe, tant au niveau des nombreuses connexions en jeu qu'au niveau des types de modules. Il influe également sur PSQN, car les paramètres ne sont pas forcément homogènes, même pour les modules de même type. Il influe enfin sur les conditions initiales, phase dans laquelle le

modélisateur doit gérer une grande quantité de données avec une précision critique. Un deuxième facteur de complexité est le caractère systémique du réseau, il se manifeste notamment par l'hétérogénéité topologique et paramétrique et par l'impossibilité constitutive de partitionner et structurer le réseau. Le modèle physique se différencie ici nettement du modèle géométrique, pour lequel les structures hiérarchiques sont adaptées et qui facilitent notablement le travail du modélisateur.

Toutes ces difficultés sont source de richesse, à condition qu'elles soient traitées de manière adéquate au sein de MIMESIS. Leur simple énoncé ne suffit évidemment pas à saisir tous les tenants et aboutissants et d'autres difficultés propres au modèle physique particulaire seront soulevées dans les travaux que nous décrivons ici. MIMESIS 4 [ELC06] constitue la première tentative de répondre à cette problématique.

4.3. MIMESIS 4

MIMESIS 4 illustre directement dans sa présentation graphique la structure en cinq phases du processus de modélisation, comme l'illustre partiellement la figure 5. L'utilisateur définit la topologie du réseau à l'aide d'un langage de script. Puis, il définit les paramètres physiques à l'aide d'un tableau ou de widgets dédiés. Les mêmes modalités sont disponibles pour définir les conditions initiales, plus une manipulation directe dans un espace tridimensionnel. Il peut enfin habiller le mouvement à l'aide d'une collection de formes élémentaires avant de simuler son modèle.

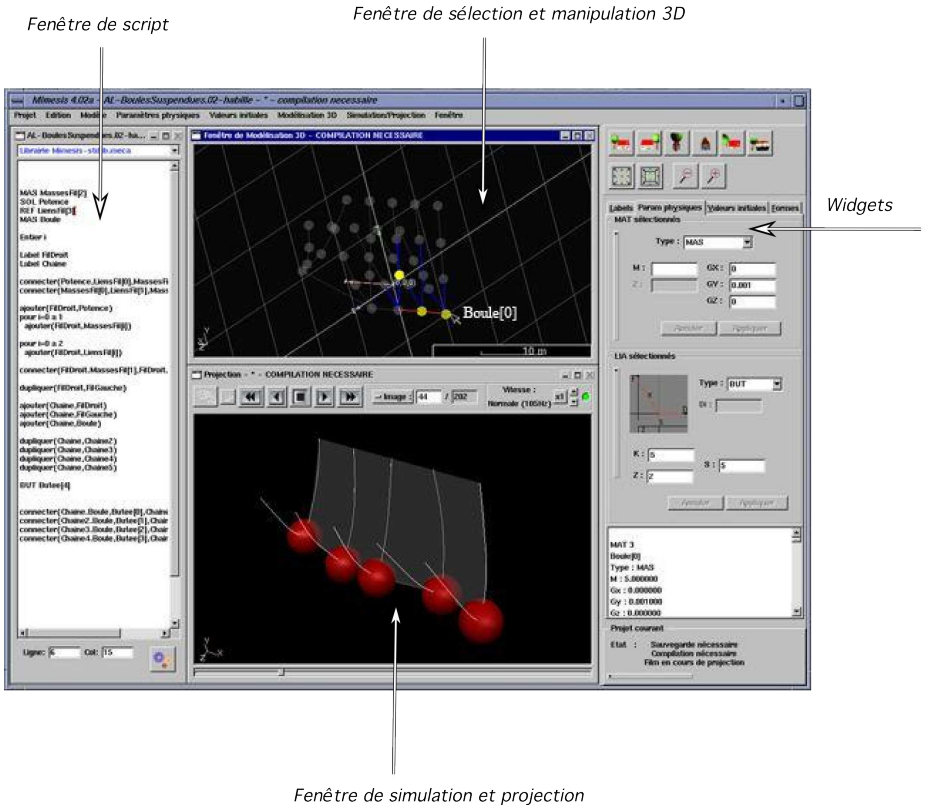


Figure 5 : Aperçu de l'interface MIMESIS 4

Nous reviendrons plus en détail sur les caractéristiques de cette interface, mais nous pouvons déjà souligner la claire séparation des différentes phases, notamment entre PSQL et les quatre autres. En effet, la séparation est marquée par l'utilisation exclusive du script pour PSQL. On remarque également que cette affectation du script à cette tâche se justifie particulièrement pour des modèles de grande taille, nécessitant un grand nombre de connexions.

Comme nous le verrons par la suite, MIMESIS 4 se définit également par la collection de modules CORDIS-ANIMA mis à disposition du modélisateur. La constitution de cette collection est un élément critique dans la conception d'un environnement de modélisation.

De nombreux problèmes sont soulevés par cette interface. Elle a montré son utilisabilité sur des modèles relativement peu complexes en terme de taille, mais montre des faiblesses certaines lorsque le nombre de modules dépasse l'ordre de la dizaine de milliers. MIMESIS 4 garantit en effet l'accès individuel à un module, indispensable pour respecter la nature systémique du réseau. Mais il y a ici un compromis à trouver entre cet impératif et la gestion de très grands réseaux. Ce handicap se manifeste également dans la phase de conditions initiales, pour laquelle les modalités d'interaction proposées restent insuffisantes.

5. Conclusion

La thèse de Matthieu Evrard sur MIMESIS [Evr09] marque la fin de la première phase du projet MIMESIS. Les travaux présentés dans ce manuscrit constituent le début de la deuxième phase du projet MIMESIS. Cette deuxième phase se situe dans un projet plus large du laboratoire visant à proposer une suite logicielle complète pour la simulation multisensorielle interactive. Cette suite est basée sur un noyau commun à tous ses éléments, dont le premier, GENESIS^3, est consacré à la création musicale. MIMESIS V est le deuxième élément prévu dans cette suite logicielle. Il est bien entendu consacré à l'Art du Mouvement. Nous avons orienté sa conception pour donner au modélisateur l'accès à la complexité, ou plutôt aux complexités : la complexité des phénomènes produits, la grande taille des réseaux, leur hétérogénéité, la complexité de leurs paramètres et des conditions initiales. Pour cela, nous avons mis l'accent sur trois axes. Le premier consiste à introduire des non linéarités au delà du composant d'interaction, en la faisant sortir du composant. Il s'agit d'étendre CORDIS-ANIMA pour permettre une modification en ligne des paramètres physiques des modules. Cet axe est l'objet du chapitre II. Le deuxième axe, traité au chapitre III, est entièrement consacré à la partie interface. Nous traiterons notamment du langage de script commun à la suite logicielle, PNSL (Physical Network Scripting Language), auquel nous avons contribué. Nous insisterons particulièrement sur la collaboration entre deux modalités de manipulation du réseau, la première étant basée sur des scripts PNSL et la seconde sur les manipulations graphiques. Le troisième axe, abordé au chapitre IV, est consacré au problème des conditions initiales. Leur définition est un point sensible du processus de modélisation, le modélisateur est confronté à une grande quantité de données à gérer et une relation complexe avec la géométrie, domaine orthogonal à la physique.

Chapitre II. Réseaux masses-interactions et phénomènes complexes

1. Non-linéarités et contrôle paramétrique en ligne dans un réseau masses-interactions

1.1. Introduction

L'implantation informatique de modèles physiques de tout type, a abouti à la synthèse d'un grand nombre de phénomènes dynamiques différents. De tous ces procédés, le formalisme masses-interactions se démarque de par la largeur du spectre de phénomènes modélisables [Luc00, LJC+91]. La nature systémique de ce formalisme de modélisation et de simulation, c'est à dire sa structure réseau d'automates cellulaires, à l'opposé de la structure maillage des systèmes masses-ressorts, constitue une première explication de cette propriété de généricité. Cette formulation réseau du modèle physique se différencie également de la formulation équationnelle, pour laquelle les non-linéarites sont complexes à formuler, voire impossible à résoudre. Cet avantage est déterminant dans la généricité des systèmes masses-interactions, car l'introduction de ces non-linéarités au sein des composants interactions permet la modélisation de phénomènes complexes, intrinsèquement non linéaires, comme des fractures et des turbulences, des avalanches ou effondrements.

L'initialisation, le développement et la pratique des réseaux masses-interactions, et particulièrement le formalisme CORDIS-ANIMA, ont donné lieu à un corpus important d'interactions non linéaires pour le mouvement visuel, mais également pour la musique et l'interaction gestuelle. Notre thèse s'inscrit dans cette histoire, aux côtés de la mise au point de nouveaux types d'interactions (interaction non linéaire visqueuse avec seuil en distance ou plus généralement interactions conditionnelles par exemple) et des outils de conception de ces interactions. Au point actuel de cette histoire, il y a MIMESIS 4 et sa collection d'interactions non linéaires, telle que décrite par Matthieu Evrard dans sa thèse. Il s'agit de fonctions d'interactions conditionnelles dans lesquelles la force dépend des variables extensives positions ou vitesses en provenance des éléments matériels que ce composant d'interaction connecte, selon des conditions sur ces mêmes variables. Le profil de cette fonction d'interaction peut donc être est prédéterminé à l'intérieur du composant lui-même.

Cette panoplie d'interactions a déjà permis la modélisation d'un grand nombre de phénomènes. Certains types de complexité ont pu être ainsi atteints : foules, plasticité, fractures ... Pour nos travaux, nous abordons la question de la modification des valeurs des paramètres des fonctions d'interaction par des contrôles externes au composant.

Cela signifie que le profil de la fonction d'interaction devient alors modifiable par n'importe laquelle des variables d'état du modèle, voire même de l'extérieur de celui-ci, par exemple les variables gestuelles. La non linéarité fait ainsi partie du modèle dans son entier et non plus inscrite de manière prédéterminée dans le module <LIA>. Cela permettra de modéliser d'autres types de comportements non linéaires que ceux modélisables au sein du module d'interaction. Un exemple caractéristique est la modélisation de phénomènes complexes comme des changements de phases : par exemple un effet de gélification par la modification de la viscosité ambiante par la valeur moyenne des vitesses de toutes les masses ou par un paramètre externe au modèle lui-même.

1.2. Historique

Les premières introductions d'interactions non linéaires prédéterminées ont été proposées par C. Cadoz avec les modules dits « LIC » pour liaisons (ou interactions) conditionnelles. Le principe consistait à introduire dans les modules <LIA> des interactions linéaires élastiques et visqueuses (Modules REF) un automate d'état fini, réglant les valeurs des paramètres physiques d'élasticité K et de viscosité Z en fonction de conditions sur les variables d'état entrantes et /ou sortantes, directes ou dérivées, du module concerné. Les LIC ont été utilisées dans les premières versions du simulateur CORDIS-ANIMA pour modéliser un frottement d'archet collophané [FC90]. Elles ont été également utilisées pour modéliser des effets de frottement sec par S. Jimenez [JL93]. Ce formalisme a été étendu par Gilles Pommereuil [PL99], qui a introduit des fonctionnalités très génériques, sous la forme d'un langage appelé LCM, Liaisons Conditionnelles à Mémoire, qui, comme leur nom l'indique, permettent d'introduire des processus à mémoire dans les automates d'état finis des modules d'interaction selon le schéma de la figure 6.

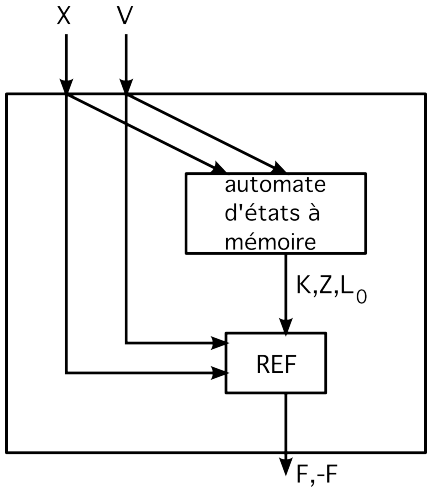


Figure 6 : Modèle d'une interaction conçue avec le formalisme LCM

L'introduction de la mémoire a permis de formaliser l'interaction de plasticité introduite par B. Chanclou [CLH96] ou les effets de striction par modulation externe de l'élasticité de cohésion [PL99]. Ce langage, qui se présente comme un langage annexe du logiciel ANIMA, est assez complexe et n'a jamais pu être utilisé en dehors des modèles validés

par G. Pommereuil. Les LIC et les LCM ont eu pour vocation d'attaquer l'introduction d'interactions modélisées par des automates d'état au sein des composants d'interaction. Un cas particulier très fréquent est celui de certaines non-linéarités continues dans lesquelles la relation entre la force et la variable extensive qui lui est reliée est une fonction unicursale potentielle F(d) ou dissipative F(v) telles que représentées sur la partie gauche de la figure 7. Dans ce cas, ce type de fonctions se prête de manière pertinente à des approximations par des fonctions linéaires par morceaux, qui ont été appelées LLM (pour Liaisons Linéaires par Morceaux) dont la forme générale est donnée dans la partie droite de la figure 7 et dans les équations 6 et 7. Celles-ci ont été implantées pour la première fois en temps réel sur les premiers processeurs puissants dédiés aux calculs temps réel, comme le processeur vectoriel AP120 (Array Processor 120) de FPS (Floating Point Systems) que l'ACROE a acquis en 1982, puis dans la version en langage interprété de CORDIS-Off, développé par S. Jimenez.

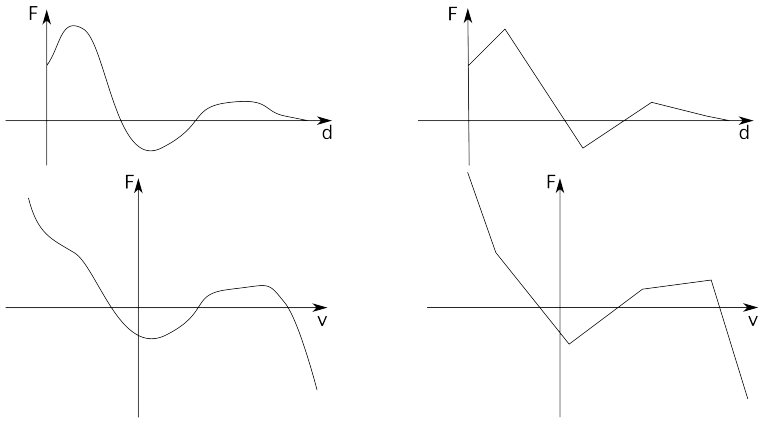


Figure 7 : Deux fonctions non linéaires, potentielles et dissipatives, et leurs linéarisations par morceaux

$$F(d) = \begin{cases} K_1 \times d + F_1 & si\ 0 < d < S_1 \\ K_i \times d + F_i & si\ S_{i-1} < d < S_i \\ K_n \times d + F_n & si\ d > S_{n-1} \end{cases}$$

Équation 6 : Fonction potentielle linéaire par morceaux

$$F(v) = \begin{cases} Z_1 \times v + F_1 & si\ v < V_1 \\ Z_i \times v + F_i & si\ S_{i-1} < v < V_i \\ Z_n \times v + F_n & si\ v > V_{n-1} \end{cases}$$

Équation 7 : Fonction dissipative linéaire par morceaux

Ont été modélisées ainsi diverses interactions non-linéaires telles que des répulsions ou des attractions à plusieurs zones. De plus, cette formulation linéaire par morceaux pouvait être raffinée sur certaines parties critiques de la fonction avec des parties de fonction tabulées [FRL+86].

L'édition graphique d'une interaction LLM générique telle que représentée sur la figure 8 consiste à définir ou à modifier des points de contrôle. La donnée des pentes des segments, donc les valeurs des paramètres physiques pour chaque segment, n'est pas manipulable car ce n'est pas un paramètre univoque. Elle n'est donnée sur la figure qu'à titre indicatif. La mise entre les mains du modélisateur d'une telle fonction très générale présente un inconvénient majeur, tout au moins dans des stades d'expertises peu matures. En effet, si elle leur permet de pouvoir définir un profil de courbe quelconque,

lors de la manipulation des paramètres de la courbe, ils peuvent être conduits à changer radicalement de catégories de phénomènes physiques. Ainsi par exemple, rien n'empêche de passer d'une interaction de répulsion à trois morceaux à une interaction de cohésion également à trois morceaux, alors que les phénomènes physiques que ces deux interactions représentent sont très disjoints et que l'on ne passe pas impunément de l'un à l'autre. Cela peut perturber notablement le fil conducteur dans le processus de modélisation.

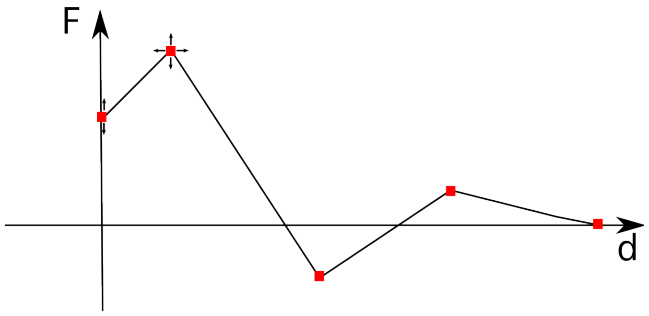


Figure 8 : Manipulation d'une interaction LLM générique

Confrontés à cette généralité trop grande de cette représentation des LLM, Matthieu Evrard et al. [ELC06] ont été amenés à réfléchir à une panoplie de fonctions LLM prédéterminées dont les paramètres significatifs du point de vue de la modélisation étaient manipulables par le modélisateur tout en restant au sein d'une catégorie physique. Ainsi ont été définies un petit nombre d'interactions LLM correspondant à des grandes catégories bien identifiables : <BUT>, <BUL>, <REP3>, <REP4>, <ATR3>, <ATR4>, <COH3> et <COH4> (cf. annexe 1). L'intérêt des interactions prédéfinies est ici flagrant. En imposant des contraintes supplémentaires en plus de la continuité, elles permettent de définir les pentes comme des paramètres de raideur. Par exemple, pour l'interaction <ATR3>, ces contraintes consistent à définir une force nulle sur le dernier segment et des pentes toutes positives. A l'usage, l'intérêt de la LLM générique s'est révélé relativement limité, les interactions prédéfinies convenant dans la majorité des cas. De plus, une analyse des modèles développés montre que ceux-ci n'ont jamais dépassé plus de quatre morceaux. Le faible nombre de ces catégories (huit) ainsi que ce faible nombre de morceaux est un avantage évident pour l'utilisateur.

Les interactions visqueuses linéaires et non linéaires, ainsi que les interactions de frottement sec <FROS> et de plasticité <PLAST> complètent cette bibliothèque (cf. annexe 1).

Comme nous l'avons déjà mentionné, les LLM ne traitent pas des non linéarités non unicursales. Les deux grands cas sont :

(1) les interactions continues qui dépendent d'une autre variable d'état que celle en jeu dans l'interaction elle-même. Un exemple typique est celui d'une fonction d'interaction dissipative F(v,d) dont la condition de non-linéarité dépend de la distance ou l'interaction de frottement sec.

(2) Les interactions strictement non unicursales. Des exemples typiques sont les interactions présentant des phénomènes d'hystérésis tels que dans la plasticité [CLH96] (cf. figure 10) ou les interactions de type frottement

collophané dans lesquelles la force dépend de la vitesse et d'une autre force F(v,p) [FC90] (cf. figure 9).

Ces interactions ne peuvent être définies que par l'usage d'automates d'état selon le schéma général de la figure 6, dont les LIC et les LCM sont des implantations significatives.

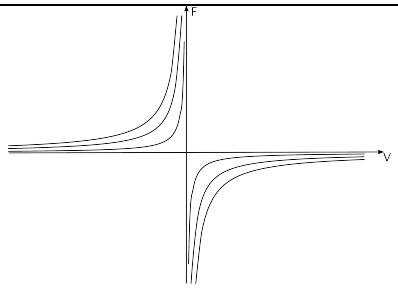


Figure 9 : Caractéristique Force/Vitesse de l'interaction de frottement d'archet pour différentes valeurs de pression

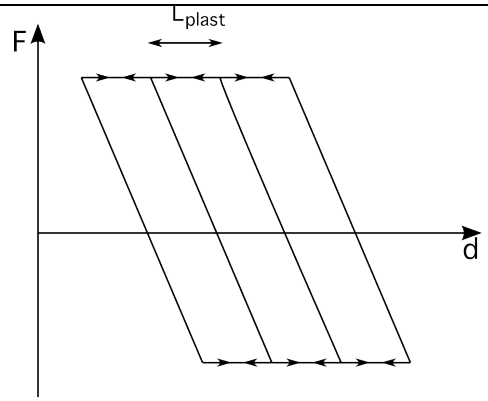


Figure 10 : Interaction de plasticité

Enfin, par principe du formalisme masses-interactions, les interactions peuvent être mises en parallèle, il est possible de combiner tous les formats d'interactions précédents pour créer des interactions non linéaires complexes. Deux usages illustrent bien cette possibilité, par ailleurs largement utilisée dans la création de modèles CORDIS-ANIMA. Tout d'abord, pour une question d'économie de représentation pour l'utilisateur et sans ajouter de restrictions en matière de modélisation, l'interface de conception de MIMESIS propose à l'utilisateur une interaction composée de deux interactions en parallèle : une interaction F(d) linéaire par morceaux potentielle et une interaction dissipative[1] F(v,d) dans laquelle les seuils de distances sont identiques à ceux de F(d). La représentation fonctionnelle d'une de ces interactions parallèles est donnée en figure 11.

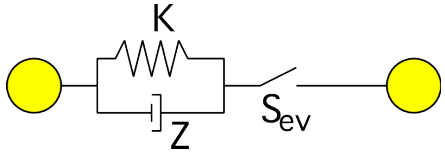


Figure 11 : Schéma fonctionnel de l'interaction de butée dans MIMESIS

Ensuite, sur le plan de la modélisation de phénomènes complexes, [Luc00] montre que la mise en parallèle de deux fonctions d'interaction non linéaires permet de balayer toute une variété de comportements complexes très différents, des sables aux fluides en passant par des pâtes, par simple changement paramétrique. Il s'agit ici:

- D'une fonction d'interaction potentielle non linéaire simple, dite d'élasticité seuillée, représentable par une LLM, où la valeur d'élasticité K dépend d'un seuil de distance S_k,

[1] Interaction dissipant de l'énergie.

- et d'une fonction d'interaction visqueuse non – linéaire, dite viscosité seuillée, représentable par un automate d'états simple où la valeur de la viscosité dépend d'un seuil de distance S_z

Ainsi, la majeure partie des interactions non linéaires développées pendant 30 ans au groupe ACROE-ICA et dont la non linéarité est définie à l'intérieur du module d'interaction sont disponibles dans MIMESIS 4. Reste alors à étudier une importante situation complémentaire : celle où cette non linéarité peut dépendre de variables d'état quelconques, c'est à dire en provenance d'autres modules du modèle, voire même du monde réel. Ce cas, qui est l'objectif central de notre apport, est détaillé dans les paragraphes suivants de ce chapitre.

1.3. Contrôle paramétrique en ligne dans CORDIS-ANIMA

La modification en ligne des paramètres physiques d'un module CORDIS-ANIMA a déjà été pratiquée par le passé, sous différentes formes. L'exemple le plus marquant est sans aucun doute l'interaction de frottement d'archet. Les études en acoustique sur le frottement d'archet montre que le frottement dépend de la pression. Jean-Loup Florens s'est attaché à la modélisation et à la simulation temps réel de cette interaction [Flo02]. Le frottement d'archet est un module fonctionnel constitué de deux liaisons 1D, l'une modélisant le mouvement longitudinal, le frottement à proprement dit, et l'autre le mouvement vertical, la pression exercée sur l'archet (cf. figure 12). C'est pourquoi la caractéristique de la liaison frottement est contrôlée par la pression. La figure 9 illustre les variations de cette caractéristique pour 3 valeurs de pression différentes.

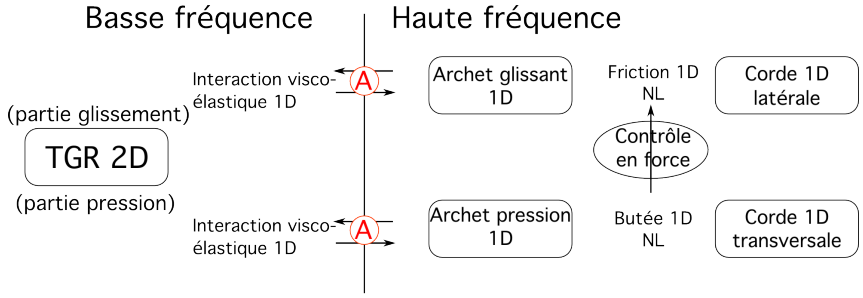


Figure 12 : Le module frottement d'archet couplé au TGR (traduit de [Flo02])

Le contrôle de paramètres est ici réalisé de manière spécifique à l'intérieur d'un seul et unique module fonctionnel. Ce type de contrôle illustre la limite mouvante entre modélisation et élaboration d'un module fonctionnel, puisqu'ici le modélisateur ne manipule pas le contrôle de paramètres, il ne manipule qu'un module qui l'incorpore. Dans le même esprit d'intégration, le module piston, utilisé comme moteur péristaltique (cf. figure 13) par Chanclou dans [Cha96], présente deux points L 3D et un point L 1D. Le module est équivalent à un ressort-frottement dont la longueur à vide est modulée par le scalaire reçu via le point L 1D (cf. figure 14). Chanclou parle d'interaction contrôlée en position.

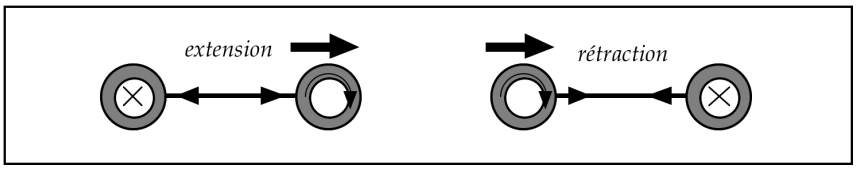


Figure 13 : Mode d'avancement du péristaltique

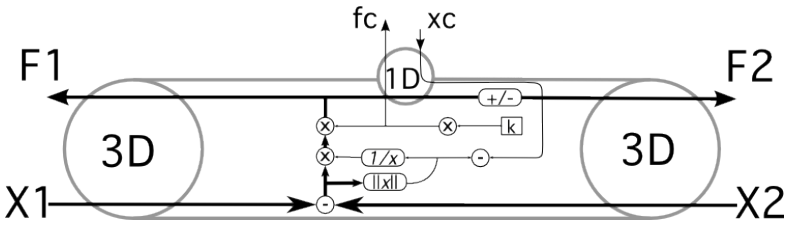


Figure 14 : Module piston

Dans ces deux exemples, le signal de contrôle est généré de deux manières très différentes. Pour le modèle d'archet, il provient directement du geste, par le biais du transducteur gestuel à retour d'effort. Pour le péristaltique, il provient d'un module de contrôle non physique qui produit des oscillations. Ce module peut être connecté au reste du modèle pour faire varier la commande en fonction de l'état du système. Chanclou a ainsi mis au point des modules de mesure réservés à cet usage.

Nous constatons dans ces deux exemples que le contrôle de paramètres s'est toujours manifesté de manière intégrée dans CORDIS-ANIMA et non de manière modulaire. De plus, les deux modules que nous venons de présenter ont été conçus pour des modèles précis et n'ont jamais été intégrés dans un environnement de modélisation. Il en est de même pour les modules de mesure de B. Chanclou.

Nous proposons d'intégrer le contrôle de paramètres dans CORDIS-ANIMA de manière à ce qu'il devienne un véritable outil de modélisation. Cette intégration se manifeste sous la forme d'un ensemble de modules et de méthodes, permettant de modifier à chaque pas de simulation les paramètres d'un module CORDIS-ANIMA via un processus externe au module. Cet ensemble est d'abord formé de deux familles de modules, les modules de contrôle de paramètres (<MCP>) et les modules de mesures (<MES>). Avec l'introduction de ces modules, la modification des paramètres en fonction, ou non, de l'état du système, devient partie intégrante du processus de modélisation. Dans un deuxième temps, nous proposons une méthode de construction de contrôleur, c'est-à-dire de processus engendrant le contrôle, à partir de modèles CORDIS-ANIMA unidimensionnels. Nous introduisons ainsi une méthode qui correspond réellement à un changement dans la manière de penser le contrôle : depuis un contrôle explicite à un contrôle dynamique implicite directement conçu par le modélisateur. Différemment des LLM et des LIC, la non-linéarité est construite en assemblant et paramétrant des modules, et non plus par le paramétrage complexe d'un module.

Notre travail s'est focalisé sur la modélisation de contrôleurs masses-interactions, mais les <MCP> et les <MES>, associés au format de description de mouvements GMS[2][LEC+06] sous la forme de fichier ou de flux, permettent d'introduire d'autres types

[2] Gesture and Motion Signal

de contrôleurs via une application tierce. En effet, il est important de proposer une généricité importante, tout en préservant la nature de MIMESIS : un environnement de modélisation de réseaux masses-interactions.

Ce chapitre va développer le thème du contrôle paramétrique selon trois axes. Premièrement, nous établirons une étude comparative des différents processus de contrôle de modèles physiques, principalement particulaires, et des outils nécessaires à la conception de ces contrôleurs. En second lieu, nous détaillerons l'augmentation de la famille des modules CORDIS-ANIMA par les modules <MCP> et <MES> : leur nature, leur impact sur le formalisme. Enfin, nous terminerons sur deux exemples de modélisation de phénomènes complexes illustrant la conception de contrôleurs CORDIS-ANIMA.

2. Processus de contrôle de modèles physiques : une étude comparative

2.1. Contrôle de paramètres et contrôle de la simulation

Nous appelons contrôle de paramètres l'ensemble des outils et méthodes permettant de modifier à chaque pas de simulation les paramètres d'un module CORDIS-ANIMA via un processus externe au module. Le contrôle de paramètres est donc un outil élargissant la palette du modélisateur, et non pas, comme son nom pourrait le suggérer, un moyen de contrôle de la simulation. En effet, les outils de contrôle de la simulation visent à donner à l'animateur un contrôle dit de haut niveau sur l'animation en spécifiant des comportements ou des contraintes géométriques (voire cinématiques) sur les trajectoires. Des méthodes numériques comme l' "Adjoint Method" pour le contrôle des simulations de gaz et fluides [MTP+04, SDE05]) ou les "spacetime constraints" [BB88, WK88] sont ainsi utilisées pour donner à l'animateur un contrôle phénoménologique sur la simulation.

Ces méthodes sont donc regroupées par rapport à leur objectif, et non par rapport à leur relation avec la simulation. Si on s'intéresse plus particulièrement à cette relation, il apparaît que le contrôle est réalisé principalement par l'ajout de forces supplémentaires[3]. L'objectif de recherche dans ces travaux est triple :

1) permettre le contrôle sur le plus grand nombre de modèles physiques (fluide, solide, particulaire, ...) et plus précisément sur le plus grand nombre de modèles de phénomènes

2) Donner un contrôle de haut niveau varié et complet à l'animateur en lui cachant le plus possible les algorithmes

3) Optimiser le calcul des forces supplémentaires, avec un objectif temps réel, la définition des paramètres de haut niveau pouvant alors se faire en cours de simulation.

Derrière cette conception du contrôle, c'est une conception de la modélisation physique en général qui se profile, et du rapport de l'animateur au modèle physique. La différence de conception avec la modélisation physique avec CORDIS-ANIMA est fondamentale : l'ajout de forces ne peut en aucun cas être considéré comme du contrôle, cette activité fait partie de la modélisation et ne peut être remplacée par des contrôleurs dit de haut

[3] Le contrôle peut aussi porter de manière plus primaire sur les conditions initiales

niveau, autrement dit par un formalisme au dessus de CORDIS-ANIMA. Le système modélisé doit être identique au système simulé, nous recherchons à rendre l'animateur maître du modèle physique et non le lui cacher. Prenons l'exemple typique du contrôle de trajectoire, il existe plusieurs manières de *modéliser* un tel contrôle. Dans [JLL91], Jimenez et Luciani utilisent une interaction élastique entre un point décrivant la trajectoire et l'objet (dans ce cas là un véhicule) devant suivre la trajectoire. Modéliser ainsi le contrôle de trajectoire a bien évidemment un effet sur l'animation finale. Cet effet doit pouvoir être maitrisé par l'animateur.

La confusion entre force et paramètre nous amène à insister sur un concept primordial dans CORDIS-ANIMA : la distinction entre variable d'état et paramètre. En effet, comme nous l'avons déjà constaté, notre formalisme distingue clairement ces deux types de données : les premières sont les variables qui circulent dans le réseau, positions et forces, alors que les deuxièmes sont des propriétés des modules. Nous insistons sur cette distinction, car il est commun d'englober ces deux types de données sous le terme générique de « paramètres de la simulation », quel que soit le système de modélisation (masse-ressort, éléments finis, équationnel). Dans la suite, nous ne traiterons que des travaux mettant en jeu une modification des *paramètres* au cours de la simulation.

Les différents travaux dont nous allons faire la revue se placent dans une perspective différente de la nôtre dans le sens où le système de modélisation, accessible à l'utilisateur, est distinct du système de simulation. Cette séparation oppose clairement ces systèmes à CORDIS-ANIMA qui est un système de modélisation *et* de simulation. Nous devons donc aborder ces travaux avec un double point de vue : d'une part la simulation avec un point de vue purement algorithmique, d'autre part la modélisation avec un point de vue utilisateur. Ce dernier n'est pas explicite dans ces travaux, qui privilégient le point de vue algorithmique. Cependant les outils proposés ont été imaginés pour être intégrés à des logiciels d'animation, dont ils constitueraient une brique parmi d'autres. En effet, les systèmes présentés ne visent pas une grande généricité, ils ont pour but de reproduire efficacement une classe de phénomènes déterminée. La finalité est donc assez différente, car les auteurs s'adressent davantage à des concepteurs de logiciels d'animation plutôt qu'à des animateurs. Dans la plupart des cas, l'animateur se retrouve dans le cadre d'un contrôle phénoménologique de la simulation, comme décrit au paragraphe précédent. L'utilisateur n'a donc pas accès au modèle physique et au processus qui le contrôle, son rôle est borné à la définition de données phénoménologiques (trajectoires, vitesses limites, etc...). Nous nous intéresserons donc aux processus engendrant le contrôle, en étudiant quatre cas illustrant chacun des processus différents.

2.2. Modélisation de mouvements de vers et de serpents

Les travaux pionniers de Miller en 1988 ([Mil88]) ont ouvert la voie vers la modélisation physique de créatures virtuelles. Dans ces travaux, les créatures virtuelles sont représentées par des modèles masse-ressort dans lesquels certains ressorts représentent les muscles de la créature. L'apport d'énergie nécessaire à la locomotion est modélisé par la modulation des longueurs à vide ces ressorts via des fonctions sinusoïdales bien choisies. Miller joue principalement sur les différences de phase entre

les contrôleurs attachés à chaque muscle pour obtenir des comportements différents : plutôt ver ou plutôt serpent. Cette technique a été popularisée avec le logiciel d'animation en ligne Sodaplay ([Sod]). Les contrôleurs sont réalisés en amont de la simulation, celle-ci ne rétroagit pas sur les fonctions d'évolution qui définissent la modulation de la longueur à vide. En conséquence, la créature virtuelle ne peut anticiper sur l'environnement dans lequel elle évolue, notamment le relief ou les obstacles. Le modélisateur doit donc prendre en compte cet environnement lorsqu'il écrit les fonctions d'évolution.

2.3. Créatures virtuelles et modèles comportementaux

En 1994, Terzopoulos s'intéresse à la modélisation d'un écosystème aquatique dans lequel les créatures réagissent à leur environnement, notamment la relation proie prédateurs avec les comportements de fuite et de poursuite [TT94]. Les modèles physiques utilisés pour les poissons sont semblables à ceux utilisés par Miller. Le principe de fonctions sinusoïdales pour contrôler les muscles est également repris mais avec une granularité différente. En effet, Terzopoulos définit une série de contrôleurs élémentaires correspondant à une famille de fonctions réalisant chacune un comportement élémentaire : avancer, tourner à droite ou à gauche, descendre, monter. Un modèle comportemental est ensuite mis en jeu pour sélectionner un contrôleur en fonction de la situation. Le contrôleur n'est donc pas entièrement défini avant la simulation, elle influe sur lui par le biais du modèle comportemental, qui est donc nourri par des données en provenance de la simulation. Ces données proviennent d'une modélisation du système sensoriel des créatures. Le schéma en figure 15 résume cette approche.

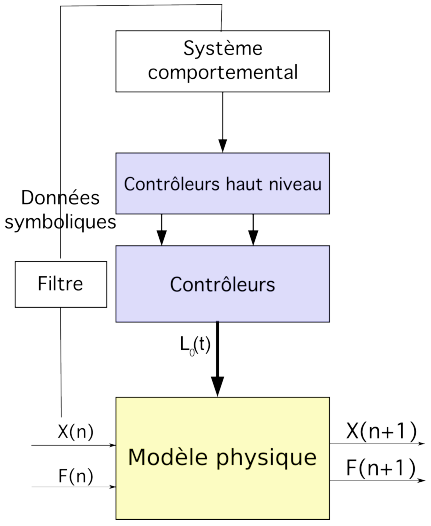


Figure 15 : Contrôle des paramètres via un système comportemental

Cet exemple illustre une première approche dans les possibilités de rétroaction du modèle physique sur le contrôleur. L'animateur peut alors se focaliser sur la création de l'environnement virtuel, des créatures qu'il y insère et des modèles comportementaux qu'il leur attribue. Le modèle physique et son contrôleur "bas niveau" sont complètement transparents pour lui.

2.4. Réseaux de neurones et contrôleurs

L'utilisation de réseaux de neurones comme représentation du système nerveux de créatures virtuelles apparaît comme un choix relativement naturel. En prenant cette option dans [vdPF93] et [vdPKF94], Van de Panne et Fiume obtiennent des créatures avec des mouvements surprenants. A la différence des deux premiers travaux présentés, leurs créatures sont modélisées à partir de solides articulés et les muscles sont représentés par des servo-contrôleurs du deuxième ordre, assimilable à des ressorts visqueux. Le réseau de neurones envoie des objectifs en distance ou en angle aux servo-contrôleurs, ces derniers les traduisent par un changement de leur longueur à vide ou de leur angle à vide. Symétriquement, le réseau de neurones reçoit en entrée des données captées par les senseurs des créatures. Contrairement aux deux précédents travaux, ces créatures se rapprochent plus de robots virtuels que d'animaux virtuels, Van de Panne parle d'ailleurs de jouets mécaniques ("Wind-up toys"). On notera également que l'évolution des signaux de contrôle en réponse aux senseurs se fait à un grain plus fin qu'avec un modèle comportemental, car il n'y a pas d'interprétation symbolique des signaux issus des senseurs.

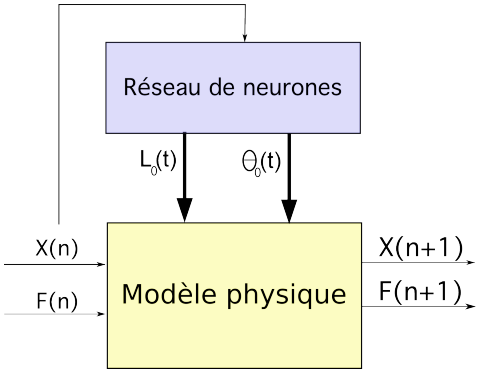


Figure 16 : Contrôle des paramètres par un réseau de neurones

Un réseau de neurones est un système complexe, impossible à régler manuellement. Les différents comportements sont donc obtenus par une stratégie "générer et tester" en fixant au départ une fonction objectif, comme par exemple "faire avancer la créature le plus loin possible". Une stratégie stochastique est ainsi utilisée pour obtenir les 200 poids du réseau de neurones. L'animateur a ici un outil assez différent de celui présenté précédemment, le contrôle phénoménologique est moindre mais les mouvements obtenus sont plus variés et surprenants. En revanche, le contrôleur n'est toujours pas directement accessible car ses paramètres ne font pas sens, ils ne sont pas directement compréhensibles, l'animateur ne peut jouer que sur la fonction objectif.

2.5. Contrôleurs et modèles physiques thermiques

Lorsqu'une matière change de comportement, par exemple au passage d'un comportement solide rigide vers un comportement quasi-fluide, les non-linéarités sont déterminantes dans la modélisation du phénomène. En 1989, Terzopoulos propose de modéliser de telles évolutions lorsqu'elles résultent d'une évolution thermique au sein

d'un matériau thermo-élastique [TPF89]. Le modèle physique du matériau est un maillage masse-ressort tétrahédrique. Il est fortement couplé à un réseau thermique isomorphe. En effet, les valeurs des conductances sont contrôlées par les distances relatives entre les masses; symétriquement, les élasticités des ressorts sont contrôlées par les températures. On est donc dans un schéma de contrôle très proche du réseau de neurones vu précédemment mais avec un système dynamique de contrôle basé sur la physique.

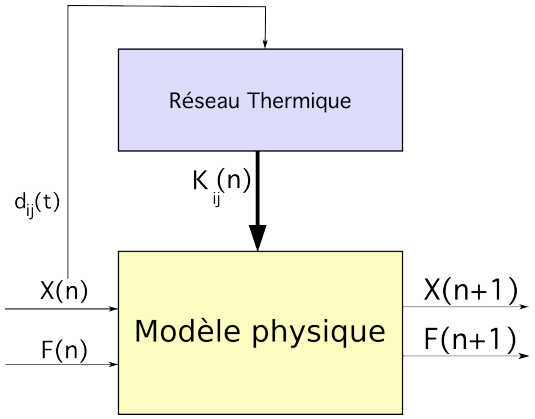


Figure 17 : Contrôle des paramètres par un réseau thermique

Le rôle de l'animateur n'est pas du tout évoqué dans cet article, mais encore une fois, le contrôleur n'est pas conçu pour être directement modifié. Le réseau thermique est en effet construit à partir du modèle physique, lui-même construit à partir d'un modèle géométrique du matériau. L'animateur a cependant une certaine influence sur le réseau thermique car il doit en déterminer les conditions initiales, c'est à dire les températures au temps 0.

2.6. Conclusion

Cette étude comparative a montré une majorité de modèles de créatures virtuelles, alors même que nous n'avons qu'évoqué le domaine du "controler synthesis" au §2.5, domaine très riche, car faisant appel à différentes méthodes d'optimisation, des réseaux de neurones [GT95,GTH98] aux algorithmes génétiques [Sim94,LLD08]. Dans leur ensemble, ces travaux mettent toujours en jeu un comportement cible, c'est ce qui est décrit par l'animateur, avec une volonté permanente de cacher le modèle physique, c'est-à-dire les équations. Cet objectif traduit une certaine vision de l'animateur, que nous appelons l'animateur scénariste, par opposition à l'animateur du mouvement pour soi. L'opposition de ces deux visions suppose une véritable rupture quant à la conception d'un environnement de modélisation.

Nous verrons dans les deux parties suivantes comment nos outils proposent une perspective radicalement différente en confiant à l'animateur l'entière conception du contrôleur et de sa relation au modèle physique.

3. Les briques pour le contrôle paramétrique dans CORDIS-ANIMA

3.1. Introduction

Les composants que nous allons introduire dans cette partie doivent nous permettre de réaliser trois objectifs :

1) Contrôler n'importe quel paramètre d'un modèle physique

2) Concevoir les processus qui définiront les valeurs des paramètres contrôlés au cours de la simulation. Concevoir le processus, pas son effet.

3) Intégrer dans le processus de contrôle des informations, des mesures, en provenance de la simulation. Le contrôle au pas N est influencé par l'état du modèle au pas N-1.

Apporter le contrôle paramétrique à CORDIS-ANIMA est une tâche risquée à plus d'un titre. D'abord, les possibilités qu'ouvre un tel moyen de modélisation compromettent la cohérence physique d'un modèle, chose que le formalisme dans sa version la plus élémentaire garantissait. Ce risque ne peut être évité, c'est une responsabilité que nous transférons au modélisateur et qu'il devra donc assumer en connaissance de cause. Ces outils sont donc à destination d'un modélisateur confirmé, ayant déjà une certaine pratique du modèle physique, lui permettant de comprendre à la fois les enjeux et les possibilités que lui ouvre le contrôle paramétrique. En second lieu, cette augmentation de CORDIS-ANIMA est critique car elle introduit de nouvelles classes de modules à côté des <MAT> et des <LIA>, c'est donc la modularité qui est en jeu et la simplicité des règles de connexion. Pour concevoir les processus de contrôle, nous avons choisi les réseaux CORDIS-ANIMA unidimensionnels, ce qui permet de conserver une modularité forte, car ils suppriment l'intervention d'éléments exogènes qui aurait nécessité de nouvelles interfaces de connexions. Nous nous attacherons donc dans cette partie à introduire deux nouvelles classes de modules :

1) Les modules de contrôle paramétrique <MCP>, dont le rôle est de transmettre les signaux de contrôle issu du réseau 1D vers les paramètres des modules concernés.

2) Les modules de mesures <MES>, dont le rôle est d'intégrer des données de la simulation dans le modèle de contrôle.

Le schéma en figure 18 résume le rôle de ces deux nouvelles classes de modules. On remarquera que les connexions sont dites dégénérées pour souligner le fait qu'elles ne sont plus bidirectionnelles. Cette remarque est importante car elle met en évidence la non-physicalité des deux nouvelles classes, contrairement aux <MAT> et aux <LIA>.

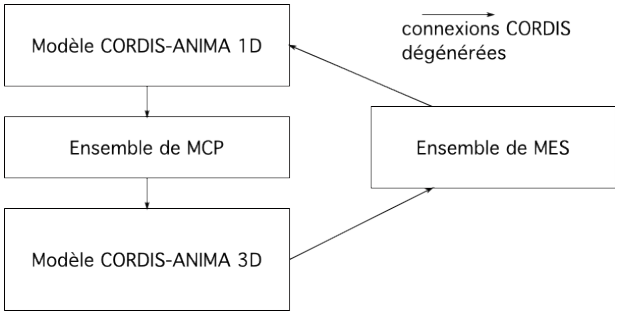


Figure 18 : Schéma général du contrôle paramétrique dans CORDIS ANIMA

Les réseaux CORDIS-ANIMA unidimensionnels présentent des spécificités déterminantes, tant au niveau des algorithmes que de la manière de les modéliser.

3.2. Réseaux CORDIS-ANIMA unidimensionnels

Le qualificatif unidimensionnel pour un réseau CORDIS-ANIMA signifie que les variables d'état (positions et forces) circulant dans le réseau sont des scalaires, elles sont à valeur dans $\mathbb{R}$. Contrairement à un espace tridimensionnel, l'espace scalaire est *ordonné*, c'est à dire qu'on peut y définir une relation d'ordre. On peut donc définir une distance algébrique entre deux positions et non plus uniquement absolue. Cette distinction est fondamentale pour les composants <LIA> qui représentent des interactions potentielles, c'est à dire calculant des forces en fonction de la distance. D'abord, les <LIA> peuvent avoir un sens de montage, c'est à dire que les deux points L ne sont pas équivalents puisque la distance entre les deux <MAT> est calculée par la différence entre la position issue du premier point L et la position issue du deuxième point L. En second lieu, le domaine de définition de la fonction d'interaction est logiquement étendu de [0;+ ∞[à]-∞;+ ∞[. L'exemple de la butée, expliqué en figure 19, permet de mieux appréhender ces deux conséquences.

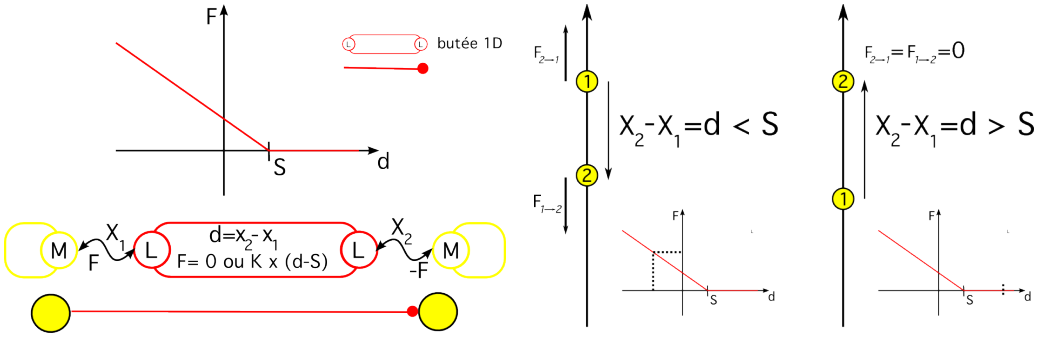


Figure 19 : Interaction de butée 1D

Cette complexification des composants d'interaction par rapport à leur version 3D nous sera très utile lorsque nous construirons des dispositifs de contrôle.

3.3. CORDIS-ANIMA 1D comme langage de conception des processus de contrôle

Nous avons pour objectif de permettre au modélisateur de concevoir directement les processus contrôlant les paramètres du modèle physique. Les différents processus exposés dans la partie précédente ne sont pas associés à des systèmes de modélisation. Les réseaux de neurones sont impossibles à paramétrer directement. Les fonctions d'évolution sinusoïdales constituent un cas bien spécifique (modélisation de muscles) et leur caractère descriptif rend difficile une rétroaction du modèle physique. Nous avons donc choisi de modéliser implicitement ces processus dynamiques à l'aide de réseaux CORDIS-ANIMA unidimensionnels. Modèle physique et contrôleur constituent ainsi un système dynamique complet, conçu avec un unique formalisme. Ce choix garantit également une certaine unité pour le formalisme en lui même, qui conserve ainsi à la fois modularité et minimalisme. Cette unité bénéficie également au modélisateur, qui n'a pas besoin de changer de paradigme pour passer du modèle physique au contrôleur.

Les qualités des réseaux CORDIS-ANIMA 1D pour modéliser des processus de contrôle seront démontrées dans la partie consacrée aux exemples. Leur utilisation pour la composition musicale, notamment dans la pratique de GENESIS [Cad02], montre déjà certaines qualités intéressantes. En effet la composition musicale met en jeu des mécanismes analogues à ceux du contrôle : détection d'événements, déclencheurs, réglage fin de dynamiques.

3.4. Les modules de contrôle paramétrique

3.4.1. De nouveaux points de communication ?

Contrôler un paramètre d'un module au cours de la simulation implique que le module doit augmenter son interface avec l'extérieur pour recevoir une donnée d'un nouveau type, un paramètre, qui n'est, rappelons-le, ni une force ni une position. Contrairement aux points L et M cette interface n'est pas bidirectionnelle, le module ne renvoie aucune variable en réponse à la réception de la nouvelle valeur du paramètre. Cette interface ne peut donc être qualifiée de point de communication, au même titre que les points L et M, elle est uniquement un *point d'entrée de paramètre*. Nous définissons donc deux nouveaux types d'interface pour les modules CORDIS-ANIMA : les points d'entrée de paramètre (Points EP) et les points de sortie de paramètre (Points SP). Les paramètres étant de nature scalaire, la dimensionnalité de ces points est également scalaire, contrairement aux points L et M qui peuvent être de dimension 1, 2 ou 3. Un module a autant de points EP qu'il a de paramètres contrôlables (cf. §3.5). Les règles de connexion sont les suivantes :

1) un point SP est connecté au moins à un point EP

2) un point EP est connecté au plus à un point SP

Concrètement cela signifie que le même processus peut contrôler plusieurs paramètres différents sur différents modules et qu'un paramètre ne peut être contrôlé que par un unique processus.

3.4.2. Les modules de contrôle de paramètre (<MCP>)

Le rôle du <MCP> est de connecter le processus de contrôle au paramètre à contrôler, ces modules sont donc davantage des vecteurs, des transmetteurs, que des algorithmes. Un <MCP> se connecte donc d'une part à une entrée de paramètre, un point EP, et d'autre part à un module <MAT> unidimensionnel dont il transforme la position en paramètre. En conséquence les <MCP> présentent un point L pour recevoir un signal de contrôle et un point SP pour transmettre ce signal.

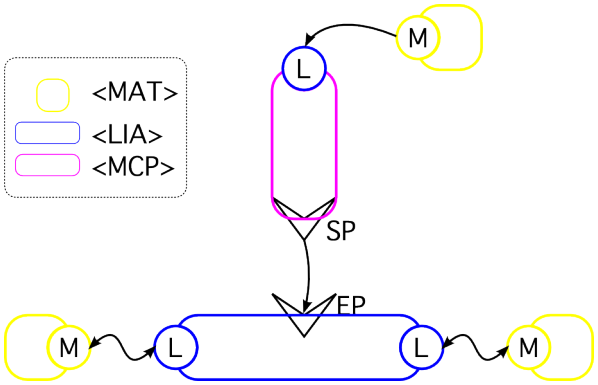


Figure 20 : Exemple élémentaire de connexion d'un <MCP>

Le <MCP> a également un rôle de transformateur, il transforme une position en paramètre. Cette transformation peut impliquer une opération de mise à l'échelle, le <MCP> implémente pour cela une transformation linéaire. Nous proposons deux types de <MCP>, le premier n'opérant aucune transformation (<CPI> pour Identité) et le deuxième opérant une transformation linéaire (<CPL> pour Linéaire) et comportant donc deux paramètres. D'autres types de <MCP> sont envisageables, mais nous excluons le <MCP> générique qui permettrait de choisir une fonction mathématique quelconque. Ce n'est en effet pas le rôle de MIMESIS de proposer des outils mathématiques complexes et complets, une telle ouverture détournerait le modélisateur de son objectif.

3.5. Les modules de mesures

3.5.1. Introduction

Les modules de mesures sont des modules de capture de données spatiales locales relatives à la simulation du modèle physique. Ces modules sont l'équivalent fonctionnel des senseurs présentés dans les travaux de Terzopoulos et Van de Panne (cf. § 2.4 et 2.5). Les données sont dites locales car elles sont relatives aux positions d'un nombre limité de <MAT>, elles sont locales au sens du réseau pas au sens de l'espace. On s'intéresse en effet à mesurer des distances, des distances projetées, des angles, et non des mesures qui intégreraient un grand nombre de variables du modèle, comme par

exemple son étendue, son centre. Les données vectorielles ne font pas partie du champ des modules de mesures car les mesures sont intégrées à l'espace scalaire de contrôle par des modules <MAT> dont elles définiront la position.

Le principe de fonctionnement des modules de mesure (schématisé en figure 22) est de capter des positions 3D via des points L dégénérés (qui ne retournent aucune force) et de retourner dans l'espace scalaire un traitement de ces positions via un point M unidimensionnel et dégénéré (il ne capte aucune force). Nous proposons trois types de <MES> selon le nombre de points L qu'ils présentent. Leur représentation graphique est donnée en figure 21.

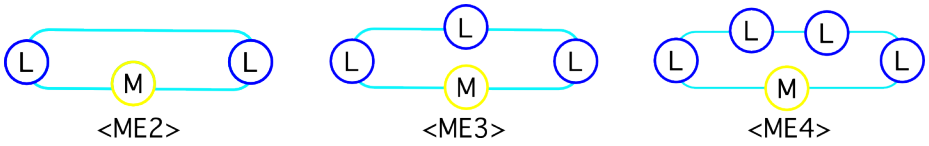


Figure 21 : Les 3 modules de mesure

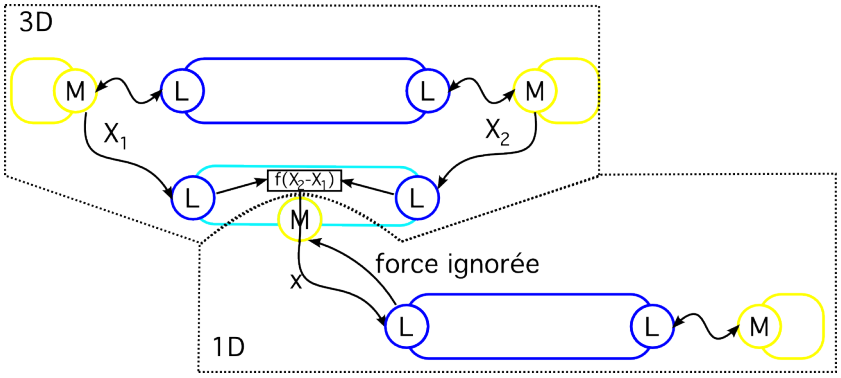
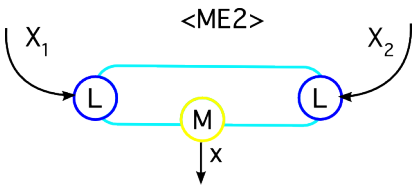
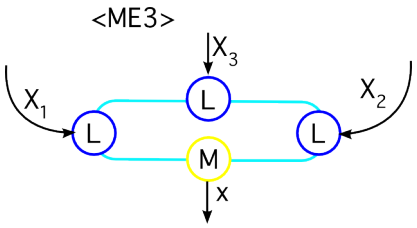
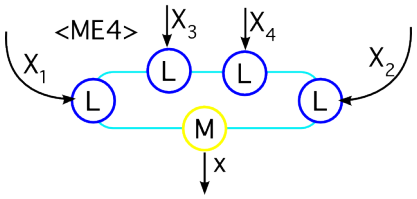


Figure 22 : Principe de fonctionnement d'une module de mesure

3.5.2. Traitement des positions dans un module de mesure

La problématique principale des modules de mesure est de transformer des données vectorielles en données scalaires. Deux outils mathématiques se présentent : la norme euclidienne, permettant de calculer une distance comme dans un <LIA> classique, et le produit scalaire, permettant d'effectuer une projection. En plus de ces transformations, les modules de mesure adaptent également les positions qu'ils reçoivent pour les mettre à l'échelle du modèle de contrôle. Ils permettent donc d'opérer une transformation linéaire sur la mesure. En plus de ces transformations linéaires, nous proposons également la fonction cosinus pour accéder à des mesures d'angles. Le tableau ci-dessous résume les opérations réalisables par les trois modules de mesure et les paramètres associés.

	paramètres a, b U ou norme euclidienne	$x = a \times \|X_2 - X_1\| + b$ $x = a \times (X_2 - X_1)U + b$
	Paramètres : a, b normer $X_3 - X_1$? normer $X_3 - X_2$? cosinus ou produit scalaire ?	$x = a \times (X_3 - X_1).(X_3 - X_2) + b$ $x = a \times \cos((X_3 - X_1),(X_3 - X_2)) + b$
	Paramètres : a, b normer $X_2 - X_1$? normer $X_4 - X_3$? cosinus ou produit scalaire ?	$x = a \times (X_2 - X_1).(X_4 - X_3) + b$ $x = a \times \cos((X_2 - X_1),(X_4 - X_3)) + b$

Ces trois modules ouvrent un certain nombre de possibilités de mesure locale. Notre objectif n'est pas l'exhaustivité mais davantage d'ouvrir une voie modulaire pour la captation de données issues de la simulation et leur usage dans des modèles de contrôle.

3.6. Contrôlabilité des paramètres

Tous les paramètres de la collection d'interactions non-linéaires sont contrôlables dynamiquement. Seule l'interaction générique LLM ne peut voir ses paramètres contrôlés. En effet, une LLM ne présente pas le même type de paramètres que ses dérivées prédéfinies (<BUT>, <COH3>,...). Les paramètres d'une LLM sont les points de contrôle de la fonction linéaire par morceaux, on ne peut les considérer comme des paramètres physiques. Pour rendre contrôlable une LLM, il faudrait l'exprimer sous forme d'un automate d'état explicitant les paramètres physiques de l'interaction, comme l'automate illustré en figure 23. Mais il est très difficile de disposer d'une représentation automate d'état pour toutes les LLM. On remarque ainsi que la famille d'automates présentée en figure 23 ne convient pas pour les interactions présentant une ou plusieurs longueurs au repos. En effet, la longueur au repos apparaît soit comme un paramètre implicite, soit comme un seuil. Elle est dans ces deux cas difficile à maîtriser. Ce problème est celui de la généricité des interactions non linéaires, ainsi que des paramètres qui les caractérisent de la façon la plus pertinente. Etant donnée la complexité de ce problème, qui demande de revoir depuis la base la conception d'une LLM, nous avons donc limité le contrôle des paramètres aux interactions prédéfinies.

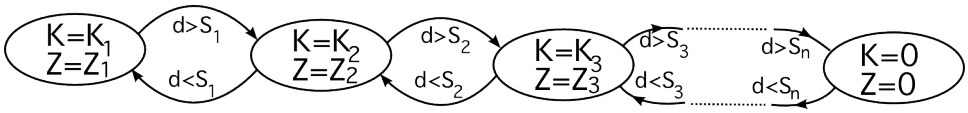


Figure 23 : Une famille d'automates d'état définissant des LLM

Nous avons insisté sur la nécessité d'interactions non linéaires prédéfinies correspondant à une fonction physique donnée, ce qui permet notamment de distinguer une interaction de cohésion d'une interaction répulsive. Le contrôle paramétrique en ligne, qui est une fonctionnalité réservée à un usage avancé, donne potentiellement à l'utilisateur la possibilité de transformer un type d'interaction en une autre, en changeant par exemple le signe d'une élasticité (cf. figure 24). Nous donnons cette responsabilité au modélisateur car elle peut être porteuse de modélisations de phénomènes complexes, comme des changements d'état. Le modélisateur doit donc maitriser son modèle de contrôle de manière à faire évoluer le contrôleur dans des intervalles bien définis. Nous avons envisagé de proposer des fonctions linéaires à saturation pour les <MCP>, donnant ainsi à l'utilisateur une maitrise des bornes d'évolution d'un paramètre indépendamment de la simulation. Cette fonctionnalité apparaît trop complexe dans un premier temps pour justifier sa présence dans un ensemble que nous voulons réduit.

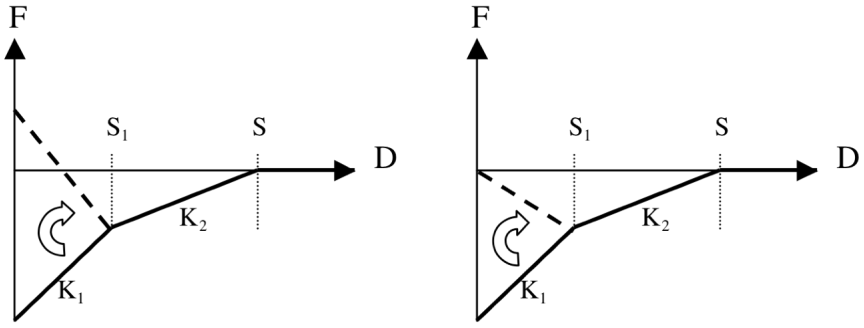


Figure 24 : Effet du changement de signe d'une raideur sur l'interaction <ATR3>

4. Pratique du contrôle paramétrique avec CORDIS-ANIMA, deux exemples de modélisation

L'objectif de cette partie est de montrer les différents types de contrôle réalisables en CORDIS-ANIMA ainsi qu'une ébauche de méthodologie pour la construction de contrôleurs en CORDIS-ANIMA intégrant une rétroaction du modèle physique 3D. Les deux phénomènes modélisés, la striction et le saut à pied joints répété, ont été choisis pour leur relative simplicité et leur complémentarité.

4.1. Modélisation du phénomène de striction

4.1.1. Description

Une fracture avec un effet de striction – comme pour les matériaux ductiles – se produit par exemple lorsqu'on étire un ruban élastique jusqu'à son point de rupture (figure 25).

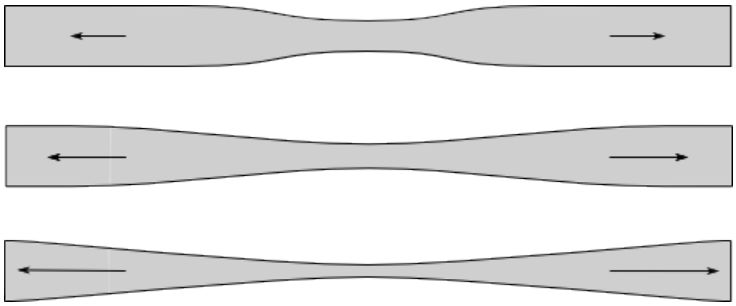


Figure 25 : Illustration visuelle du phénomène de striction

Comme illustration de ce phénomène, nous avons cherché à modéliser une goutte se détachant d'un robinet. On observe en effet le même étirement, la même fatigue progressive du matériau, qui va amener à la rupture. Par ailleurs cette illustration se prête bien à une modélisation minimale du phénomène : deux éléments ponctuels, la goutte et le robinet sont en interaction de cohésion, une cohésion qui s'affaiblit jusqu'à provoquer la rupture. Nous proposons de contrôler certains paramètres de l'interaction de cohésion pour provoquer un tel affaiblissement.

4.1.2. Modèle CORDIS-ANIMA

Pour représenter cette situation nous construisons un modèle CORDIS-ANIMA élémentaire, représenté en figure 26. Le robinet est modélisé par une masse fixe (module <SOL>) et la goutte par une masse mobile avec une viscosité de milieu (module <MASV>). Cette figure illustre également les différentes phases du phénomène :

1) Entrée en cohésion de la goutte et du robinet
2) La goutte glisse sur le robinet
3) Striction
4) Chute libre de la goutte

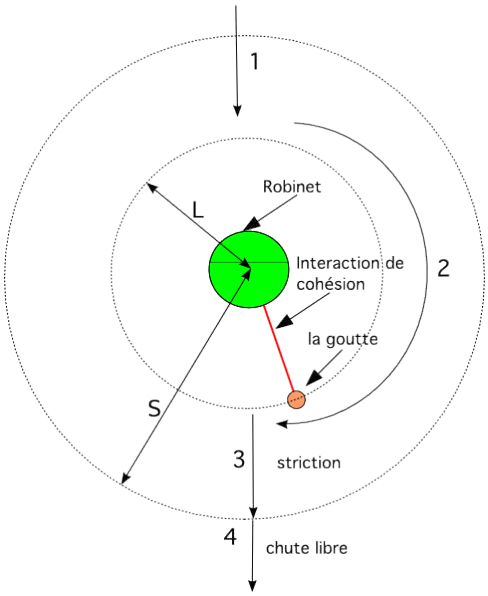


Figure 26 : Modèle physique de striction, 4 étapes du phénomène

Les paramètres L et S correspondent aux paramètres de l'interaction de cohésion, dont la caractéristique force/distance est représentée en figure 27.

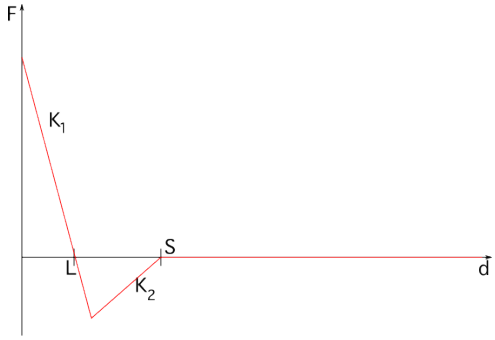


Figure 27 : Interaction de cohésion linéarisée

4.1.3. Contrôle des paramètres de l'interaction de cohésion

Comment affaiblir la cohésion entre le robinet et la goutte ? Le contrôle de paramètre tel que nous l'avons défini permet d'explorer différents types d'affaiblissement. L'interaction de cohésion présente en effet quatre paramètres qui peuvent être contrôlés séparément ou non. Nous avons expérimenté deux configurations: la configuration A consiste à diminuer progressivement K_1, la configuration B consiste à augmenter S tout en diminuant K_2. Pour bien comprendre ces deux contrôles, il faut mettre en évidence deux paramètres secondaires de l'interaction de cohésion, secondaires dans le sens où ils ne déterminent pas l'interaction du point de vue du modélisateur, qui n'a accès qu'aux paramètres K_1,K_2, L et S. Ces deux paramètres sont la force de rappel maximum, $F_{r\,max}$, et son abscisse associée, S_0 (cf. figure 28). La force de rappel maximum est un paramètre crucial dans l'affaiblissement de la cohésion. En effet, le bilan des forces sur la goutte lors de la phase 3 montre une opposition entre son poids et sa quantité de mouvement d'une part et la force de rappel de la cohésion d'autre part.

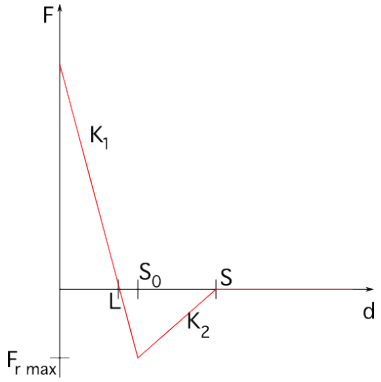


Figure 28 : Paramètres critiques de l'interaction <COH3>

On observe qu'avec la configuration A, plus K1 est faible, plus la force de rappel maximum diminue (cf. figure 29). L'interaction devient à la fois plus molle et plus fragile. En revanche, avec la configuration B, les paramètres S et K_2 sont contrôlés conjointement de manière à garder la force de rappel maximum fixe (cf. figure 30). L'interaction devient plus molle et plus étendue, mais reste toujours aussi résistante.

Ces deux configurations supposent des causes différentes pour la rupture. Pour la configuration A, la rupture survient lorsque la somme du poids et de la quantité de mouvement devient supérieure à $F_{r\,max}$. Pour la configuration B, la rupture est inéluctable, avant même que les paramètres soient modifiés. Le contrôle des paramètres K2 et S ne fait que retarder la chute libre en élargissant la zone de cohésion.

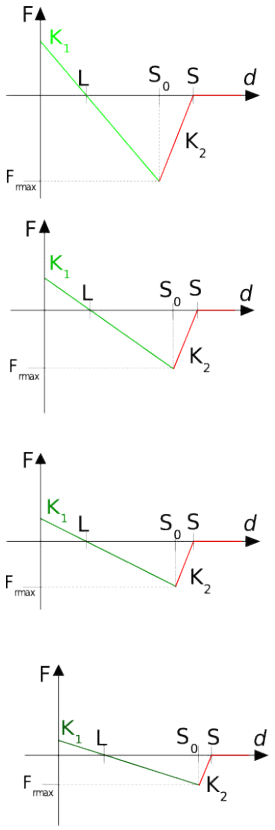


Figure 29 : Contrôle du paramètre K_1 de <COH3>

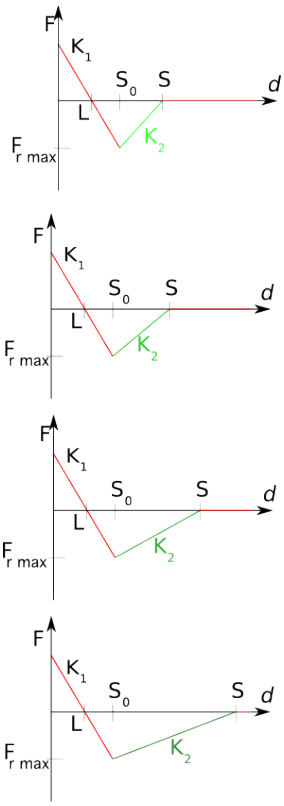


Figure 30 : Contrôle conjoint des paramètres K_2 et S de <COH3>

4.1.4. Plage et dynamique d'évolution des paramètres

Comment mettre en œuvre de tels contrôles à l'aide des outils définis en partie 3 ? Avant même de concevoir le contrôleur dans l'espace 1D, la question centrale pour le modélisateur est de définir des plages d'évolution pour les paramètres contrôlés, ainsi que des dynamiques d'évolution. D'abord dans les deux cas, la modification des paramètres est délimitée dans le temps, elle a lieu pendant la phase 3 (striction). Pour la configuration A, la plage d'évolution de K_1 va être déterminée de sorte que sa valeur maximum maintienne la cohésion au début de la phase 3, étant donnés le poids de la goutte et sa vitesse d'arrivée. La valeur minimum sera fixée de manière à ce que $F_{r\ max}$ soit plus faible que le poids de la goutte. La dynamique d'évolution est assez lente pour minimiser l'influence de la quantité de mouvement. Dans la configuration A, la plage d'évolution est moins critique que la dynamique. Celle ci doit être assez rapide pour que la chute libre ne survienne pas avant la fin du contrôle.

4.1.5. Mise en œuvre commune

La mise en œuvre de ces deux contrôles revient à répondre à deux questions :

1) Comment faire coïncider le début de la modification des paramètres avec la fin de la phase 2 ?

2) Comment construire le modèle unidimensionnel qui produira le mouvement avec la plage et la dynamique d'évolution choisie.

Avec la première question nous abordons un cas élémentaire de rétroaction du modèle physique sur le contrôleur : la détection d'un événement comme déclencheur d'une modification de paramètre. Dans notre cas l'événement correspond au moment où la goutte arrive en bas du robinet, ce que l'on peut traduire par une distance nulle entre la goutte et le point représentant le bas du robinet. Il faut donc mesurer cette distance à l'aide d'un module <MES2> (cf. figure 31 a)). La détection de l'événement se traduit dans le modèle unidimensionnel par une interaction de butée de seuil nul entre la masse représentant cette mesure et une masse de position initiale 0 (cf. figure 31 b)). Cette masse est donc mise en mouvement au moment de l'événement. Elle percute ensuite la masse qui produit le contrôle car elles sont en interaction de butée (cf. figure 31 c)). On choisit x_{min} et le seuil de cette butée pour régler le délai entre l'événement et le début de la modification du ou des paramètres. La fin de la modification est provoquée par une butée très visqueuse entre la masse de contrôle et une masse fixe située à x_{max} (cf. figure 31 d)).

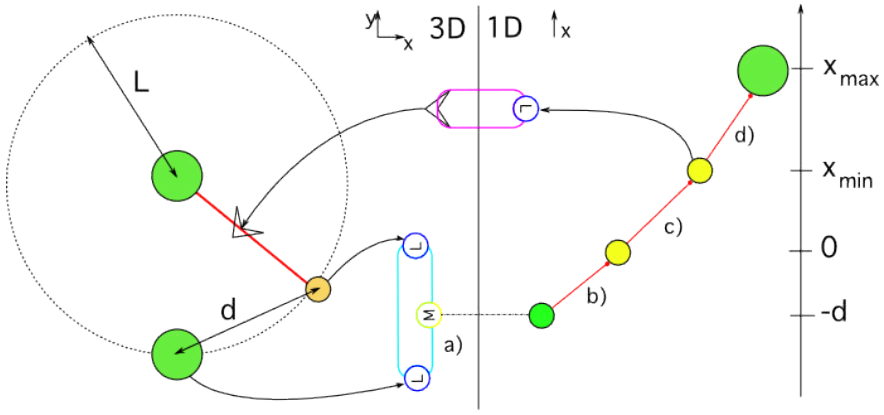


Figure 31 : Le modèle de striction et son contrôleur

La dynamique du contrôle est réglée en ajustant x_{max} et la raideur de la butée c). La plage de contrôle est réglée en ajustant les paramètres du module <CPL> pour faire correspondre l'intervalle $[x_{min}, x_{max}]$ avec la plage de paramètres cible.

4.1.6. Mise en œuvre du contrôle conjoint

Le contrôle conjoint des paramètres S et K_2 consiste à déterminer un contrôle de K_2 connaissant le contrôle de S et en gardant fixe S_0 (cf. figure 30). Il est donc nécessaire pour le modélisateur d'effectuer des calculs, en se basant sur le caractère continu de la fonction linéaire. Ces calculs aboutissent à l'équation 8.

$$K_2 = \frac{K_1(S_0 - L)}{S - S_0}$$

Équation 8 : Expression de K_2 en fonction des autres paramètres de <COH3> et de S_0

Or S_0 dépend des quatre paramètres de COH3, le contrôle de K_2 dépend donc également des valeurs initiales de K_2 et S. En conséquence, la mise en œuvre du contrôle conjoint se fait en cinq étapes :

1) Détermination des valeurs initiales de K_2 et S.
2) Création d'un <CPL> pour contrôler S.
3) Création d'un <MCP> fraction rationnelle (<CPF>).
4) Connexion de ce <CPF> à la masse contrôlant le paramètre S.
5) Détermination des paramètres du <CPF> en fonction de l'équation 8, des paramètres du CPL contrôlant S, et des valeurs initiales de K_2 et S.

La mise en œuvre du contrôle conjoint est donc une opération complexe. Dans notre exemple, elle nécessite l'introduction d'un nouveau module <MCP>, le <CPF>, faisant intervenir une fonction rationnelle simple dans le calcul du paramètre. Dans le cas du modèle de striction, les comportements qui en résultent sont moins intéressants que le contrôle simple de K_1, mais la différence de comportement constitue à elle seule une validation de ce type de contrôle.

4.1.7. Conclusion

Ce premier exemple (illustré en Annexe 3) nous a permis de proposer une première approche méthodologique pour le contrôle de paramètres dans CORDIS-ANIMA. D'abord avec l'association de modules de mesure et d'interaction de butée pour détecter un événement et déclencher ainsi une séquence de contrôle. La création de processus de contrôle à l'aide de réseaux unidimensionnels s'est révélée à la fois simple et efficace. Les réactions en chaine et leur réglage apparaissent comme des moyens naturels de concevoir un processus de contrôle. La situation est cependant relativement simple, notamment au niveau de la rétroaction du modèle physique sur le modèle de contrôle. L'exemple suivant va nous permettre d'expérimenter un type plus complexe de rétroaction pour poursuivre la validation de notre démarche.

4.2. Modélisation de sauts à pieds joints en série

4.2.1. Description et objectif

Dans sa modélisation CORDIS-ANIMA des verbes de la danse [HL05], Chi Min Hsieh introduit un modèle de saut élémentaire. Le sauteur effectue un unique saut et retombe avec un léger amortissement. Notre objectif est de faire effectuer plusieurs sauts en réalisant un programme moteur cyclique simple : contraction, extension, saut, réception.

La réalisation d'un tel programme ne peut se faire qu'avec un modèle actif, c'est à dire en injectant de l'énergie au cours de la simulation. Le modèle de Hsieh est en effet passif, l'énergie nécessaire au saut provenant strictement d'une énergie potentielle de compression, à l'image des jouets à ressort. Pour garder cette image, nous cherchons à réaliser un jouet virtuel qui se remonterait cycliquement de manière autonome.

En expérimentant une rétroaction cyclique du modèle physique sur le contrôleur, nous cherchons à valider la méthodologie commencée dans l'exemple précédent en complexifiant la situation. Nous montrerons également de quelle manière il est possible de complexifier le contrôleur, en tirant notamment parti du caractère unidimensionnel de l'espace de contrôle.

4.2.2. Modèle physique

Le modèle passif de Hsieh est représenté en figure 32. Les muscles de la jambe sont modélisés par un unique ressort connectant le bassin au pied. La contraction de ces muscles est provoquée par une vitesse initiale du bassin. Plus cette vitesse initiale est importante, plus les jambes se contractent, et plus haut sera le saut.

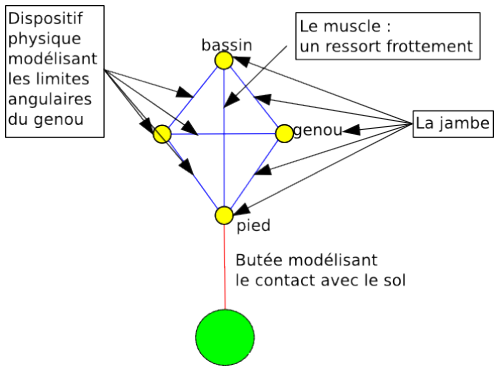


Figure 32 : Modèle de saut passif

Pour rendre actif ce modèle nous opérons comme Miller (cf. §2.3) en modulant la longueur à vide du ressort représentant le muscle. Cette modulation revient à modéliser les phases de contraction et d'extension d'un muscle réel.

4.2.3. Contrôleur, principes

A chaque phase d'un cycle correspond une séquence de contrôle précise. La phase de contraction correspond à une transition lente de la longueur à vide depuis sa valeur maximum à sa valeur minimum. La phase d'extension correspond à une transition rapide de la longueur à vide depuis sa valeur minimum à sa valeur maximum. La phase de saut correspond à une longueur à vide fixée à sa valeur maximum. La réception est une phase de faible durée qui coïncide avec le début de la phase de contraction.

La conception du contrôleur et de sa relation au modèle physique de jambe se fait selon le même principe que précédemment. L'événement à détecter est la réception du sauteur. Elle est détectée avec un module de mesure de distance entre les modules représentant respectivement le pied et le sol (cf. figure 33 a)). La détection de l'événement se traduit dans le modèle unidimensionnel par une interaction de butée de seuil nul entre la masse représentant cette mesure et la masse effectuant le contrôle (cf.

figure 33 b)). Celle ci est en interaction de butée avec deux modules <SOL> situés aux abcisses 0 et x_{max}, elle évolue donc dans l'intervalle [0, x_{max}] (cf. figure 33 c) et d)). Le module <CPL> remet ce mouvement à l'échelle pour délivrer un paramètre dans l'intervalle [L_{min}, L_{max}] (cf figure 33 e)).

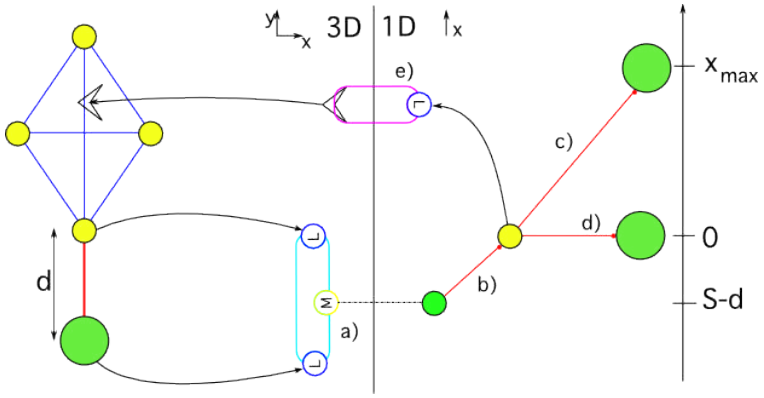


Figure 33 : Modèle de saut et son modèle de contrôle

La dynamique du contrôle est l'enjeu central dans ce modèle. La difficulté principale est de conserver cette dynamique d'un cycle à l'autre. Les figures 34 à 37 montrent comment le réglage des différentes interactions de butée permet d'obtenir la dynamique voulue. Le premier cycle est commencé par la phase de contraction en donnant une vitesse initiale V_c à la masse de contrôle (figure 34). Cette vitesse doit rester suffisamment faible pour que le muscle se contracte raisonnablement.

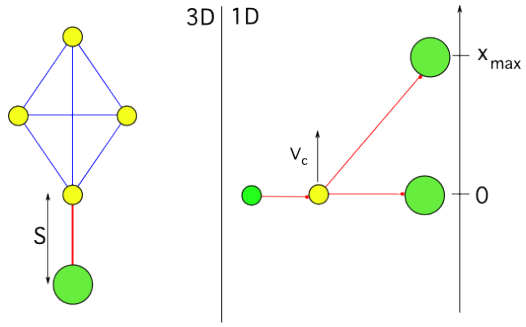


Figure 34 : Phase de contraction

A la fin de la phase de contraction, la masse de contrôle rebondit en accélérant. Une vitesse V_e élevée est en effet nécessaire pour fournir l'énergie nécessaire au saut. L'accélération est obtenue en jouant sur la viscosité négative de la butée (cf. figure 35). Un réglage fin de cette viscosité permet d'obtenir la vitesse voulue en fonction de la vitesse de contraction.

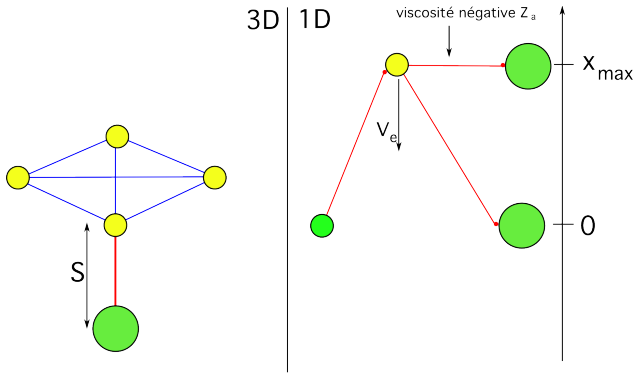


Figure 35 : Phase d'extension

L'extension est stoppée par une butée visqueuse (cf. figure 36), la phase de saut a alors déjà commencé.

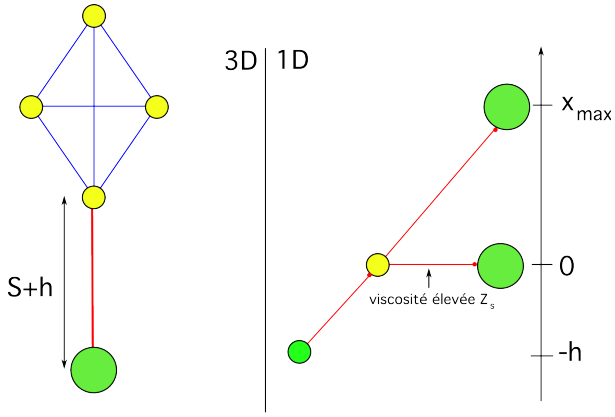


Figure 36 : Phase de saut (h est la hauteur du saut)

A la réception, la masse transmettant la mesure percute la masse de contrôle et déclenche ainsi la phase de contraction (cf. figure 37). La raideur de la butée de percussion est choisie suffisamment forte pour soustraire la masse de l'influence de la butée visqueuse.

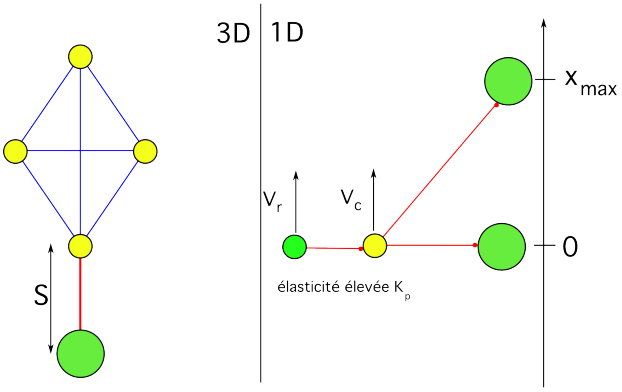


Figure 37 : Phase d'atterrissage et de contraction

Des images de la simulation, avec et sans habillage, sont disponible en annexe 2.

4.2.4. Contrôleur, améliorations.

En conservant un tel modèle de contrôle, nous ne sommes pas parvenus à obtenir un grand nombre de sauts stables. Soit les sauts décroissent en hauteur jusqu'à cesser, soit ils croissent constamment provoquant la déstructuration du sauteur. La phase critique est la phase de réception. En effet la hauteur du saut au cycle N détermine la hauteur du saut au cycle N+1, comme illustré en figure 38.

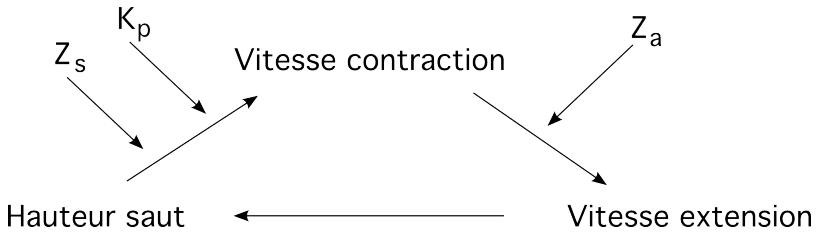


Figure 38 : Relations entre variables phénoménologiques du saut et paramètres du contrôleur

Une régulation fine de la dynamique sur chaque cycle est donc nécessaire pour prolonger le phénomène sur un grand nombre de cycles. Concrètement, cette régulation signifie une maîtrise de la vitesse de contraction V_c (la vitesse d'extension étant parfaitement déterminé par V_c et Z_a). Nous avons joué sur deux plans pour maintenir V_c dans un certain intervalle. D'abord, l'influence de la butée visqueuse est gênante au moment de la réception. Nous avons donc cherché à éliminer cette influence. Pour ce faire, nous avons modifié l'algorithme de la butée pour qu'elle ne soit active qu'en phase d'approche, c'est à dire quand la vitesse relative absolue ($V=|d|-|d_r|$) est positive. L'automate d'état en figure 39 résume cet algorithme.

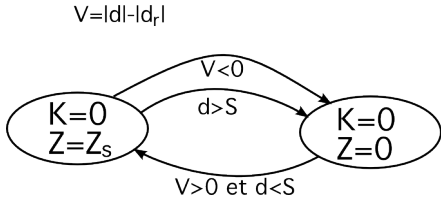


Figure 39 : Butée visqueuse en phase d'approche

Ce biais écarté, nous nous somme concentrés sur la régulation de V_c à proprement dit. La raideur K_p ne permet pas un réglage fin de cette vitesse en fonction de la hauteur du saut. Comment parvenir à limiter V_c sans affecter V_e ? Une analogie dans la vie réelle serait une bande freinante à picots orientés (cf. figure 40). Pour modéliser une interaction avec une telle bande, nous avons encore une fois modifié l'algorithme de la butée comme indiqué en figure 41.

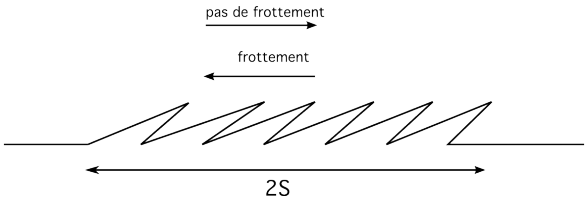


Figure 40 : Image de la bande freinante

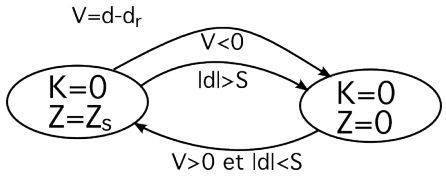


Figure 41 : Automate d'état décrivant l'interaction de bande freinante

Une fois cette interaction mise en place dans le modèle de contrôle, nous parvenons à contrôler suffisamment la vitesse de contraction pour obtenir un nombre de sauts consécutifs au moins supérieur à cinquante.

4.2.5. Conclusion

Ce deuxième exemple valide les qualités de CORDIS-ANIMA 1D comme langage de conception de contrôleurs. La topologie du contrôleur est simple et se déduit directement de l'objectif de contrôle en termes de plages et de dynamiques. Les paramètres critiques sont facilement identifiés et sont en nombre suffisamment faible pour que leur mise au point soit humainement réalisable.

Cet exemple révèle en particulier les difficultés d'un contrôle bouclé avec la simulation, même si cette boucle présente un grain de temps important (la durée d'un cycle). Les solutions avancées que nous avons apportées posent la question d'une collection d'interactions unidimensionnelles dédiées au contrôle. Cette question se pose dans les mêmes termes que pour la collection d'interactions non linéaires proposée par Matthieu Evrard. Le compromis entre une généricité nécessaire et un minimalisme de rigueur est au cœur de cette délicate question. Le contrôle paramétrique est encore au stade de l'expérimentation, c'est pourquoi nous avons fait le choix de la minimalité en intégrant les deux interactions proposées au paragraphe précédent.

5. Bilan et Perspectives

Nous pouvons résumer notre contribution à trois types d'apport. Un apport pratique d'abord, avec deux nouvelles familles de modules CORDIS-ANIMA, les modules de contrôle paramétrique et les modules de mesure, mais aussi des interactions non linéaires scalaires pour le contrôle. Un apport problématique ensuite, avec la mise en lumière du problème des paramètres complexes des modules <LLM>, illustrant la difficulté importante qui existe pour définir des interactions non linéaires de manière générique. L'apport problématique se situe également au niveau de la signification physique du contrôle : modification de la matière, modification des dimensions, apport d'énergie. Notre troisième apport est méthodologique, à travers les deux exemples que nous avons présentés, nous proposons une méthode simple de conception de contrôleurs, permettant d'aboutir à des résultats probants.

Les trois objectifs fixés au §3.1 ont été atteints, à l'exception du contrôle des paramètres des modules génériques <LLM>. Nous avons également montré qu'il était possible de concevoir un processus de contrôle plutôt que son effet et que les réseaux CORDIS-ANIMA 1D constituaient un langage de choix pour cette conception, permettant de surcroit d'intégrer naturellement une rétroaction du modèle physique sur le modèle de contrôle.

Par rapport à notre objectif plus général, nous estimons avoir franchi un pas dans les complexités explorables avec CORDIS-ANIMA. Les comportements moteurs sont ainsi modélisables, nous avons d'ailleurs tenté de modifier le modèle du sauteur en modèle de marche de quadrupède. Ce travail préliminaire a produit des résultats surprenants de crédibilité, mais il est difficile de contrôler les directions de la marche. Le modèle de striction préfigure des modèles plus complexes de changement d'état, comme par exemple le passage d'un comportement granuleux à un comportement pâteux ou fluide. Une voie de modélisation est ouverte, il reste maintenant à l'explorer.

Chapitre III. Outils pour la création de réseaux CORDIS-ANIMA complexes

1. Des outils pour créer et paramétrer un réseau CORDIS-ANIMA

1.1. Introduction

Vingt-cinq ans de modélisation CORDIS-ANIMA pour l'animation ont montré une collection de modèles divers et complexes, nécessitant des outils adaptés pour leur mise au point. Des environnements de création de réseaux CORDIS-ANIMA se sont succédés, depuis l'environnement graphique ANIMA jusqu'à MIMESIS, environnement interactif faisant appel à un langage de script, en passant par des environnements purement textuels comme ANIMA-OFF [LJC+91] et CORDIS-OFF [Man93, CLF94]. Tous ces environnements permettaient de décrire une topologie de réseau, de donner des paramètres aux modules et de définir un habillage pour les trajectoires produites par la simulation. Mais les manières d'effectuer ces définitions, et les outils périphériques disponibles pour chacune de ces tâches étaient spécifiques à chaque environnement. Certains outils ont même été ajoutés par le modélisateur au fur et à mesure de ses besoins spécifiques. Pourtant, on retrouve les mêmes tâches dans tous ces environnements : déclarer un module, déclarer une connexion entre un point L et un point M, affecter une valeur à un paramètre d'un module, associer une liste de <MAT> à une forme géométrique, etc. Mais les représentations, textuelles comme graphiques, ainsi que les modalités de manipulation de ces représentations affectent grandement la manière d'effectuer ces tâches. Ces différents environnements, exploités depuis leur création, présentent leurs forces et leurs faiblesses. Il s'agit aujourd'hui de définir un nouvel environnement s'appuyant sur toutes les expériences passées pour les porter plus loin vers une plus grande complexité et une plus grande variété de modèles.

1.2. Nommer les modules

Le premier outil commun qui apparaît à première analyse est de donner à l'utilisateur la possibilité de nommer les modules ou les groupes de modules. Il est indispensable pour repérer les modules et pouvoir ainsi les manipuler.

Dans la première version développée dans la thèse de A. Razafindrakoto, ANIMA [Raz86], un nom est automatiquement donné au module à sa création. Il s'agit d'un identifiant absolu donné aux modules de base, indépendamment de toute structure du réseau final. Il s'agit donc d'identifiants objectifs élémentaires qui permettent de manipuler le modèle,

c'est à dire l'ensemble de ses modules et de leurs connexions, à leur plus bas niveau. Si cette manière de faire a l'avantage de présenter le modèle comme un réseau à plat, on voit vite que l'utilisateur aura beaucoup de mal à créer et à manipuler des modèles complexes avec beaucoup de modules et des topologies de connexions très hétérogènes.

La deuxième version développée dans la thèse de S. Jimenez [Jim93], ANIMA-OFF, a pris le contrepied de cette vision, en permettant à l'utilisateur de donner lui même un nom à un module ou groupe de modules lors de sa création Un module est alors repéré par le nom de son groupe et son indice dans le groupe. L'avantage immédiat est ici que l'utilisateur peut manipuler non seulement des modules élémentaires mais également des groupes de modules. Cependant, ce système de nomination est relatif à la vision que l'utilisateur a de son modèle au moment de la création du groupe de module. Il introduit surtout de ce fait une interprétation des fonctions des modules ou groupes de modules. Il introduit d'autre part, une structuration à priori du réseau en sous-groupes identifiés et cela, de manière unique. Or, une fois créé, le réseau correspondant au modèle final est rarement structuré ainsi. Les interactions sont rarement de groupes à groupes. Elles peuvent être placées entre certains éléments d'un groupe vers certains éléments d'autres groupes, limitant ainsi la pertinence des groupes en question. Ainsi, si la nomination d'un groupe peut sembler être une facilité, elle ne l'est que pour les manipulations concernant tout le groupe. De plus, et surtout, le processus de création du modèle étant un processus de création interactif et évolutif, rien n'indique que le groupe ainsi défini et ainsi nommé à un moment particulier du processus de modélisation soit une entité stable tout au long du processus de création du modèle et valide pour toutes les autres tâches liées à la modélisation. Il en est ainsi pour la donnée des valeurs des paramètres ou des valeurs initiales, ou bien encore pour l'association ultérieure à des formes. Enfin, une fois le réseau mis à plat pour être simulé, cette fonctionnalité de groupe disparaît. Différemment des structures en graphes des modèles géométriques ou géométrico-cinématiques, aucun calcul ni aucun séquencement de calcul n'est lié à cette structure. Cela montre que cette fonctionnalité n'est relative qu'à la manipulation du réseau lors du processus de sa création et qu'elle est donc circonstancielle aux étapes de celui-ci. Il faut donc par conséquent définir un processus de nomination qui évite le piège d'une structuration implicite du réseau, aussi bien pour l'utilisateur que pour le simulateur. Pour le simulateur, il n'y a aucune raison que la structure correspondant à une optimalité des calculs soit homéomorphe à une structure de modélisation. Pour l'utilisateur, cela induirait des fonctionnalités implicites définitives qui limiteraient ainsi les possibilités créatrices interactives. Par exemple, dans le modèle des "boules suspendues", certains labels sont utilisés pour construire le réseau (cf. figure 42). Ces mêmes labels sont ensuite exploités pour une première visualisation, sous forme de boules suspendues à des fils. Mais pour définir une autre visualisation, sous forme de drap, la structure induite par ces labels est inadaptée, cette visualisation est même difficilement imaginable sans une réorganisation du réseau, et surtout sans la *possibilité* de cette réorganisation par de nouveaux labels.

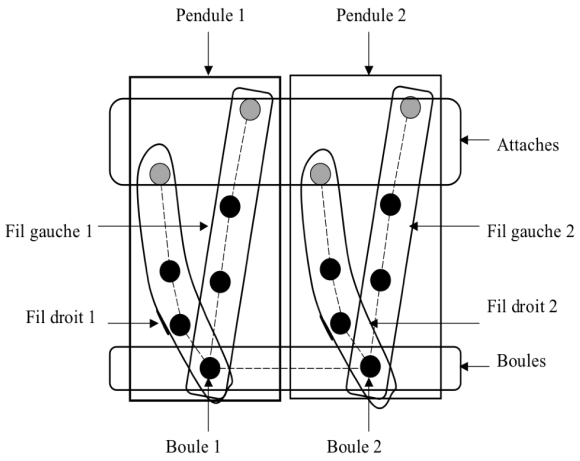


Figure 42 : Topologie du modèle "boules suspendues"

Ce processus de nomination doit néanmoins permettre de manipuler tout ou partie du réseau en cours de construction, de manière à pouvoir aboutir à la construction de réseaux complexes, non nécessairement complètement spécifiés au départ.

C'est dans cet esprit que MIMESIS 4, tout en maintenant la fonctionnalité d'identification objective absolue donnée automatiquement par le système et introduite dès ANIMA, introduit la notion de ***label***, défini comme un conteneur nommé de modules. A tout instant du processus de modélisation, l'utilisateur peut ajouter un (ou des) module(s) dans un label, générant ainsi un nouveau nom pour le(s) module(s) : `nom_du_label.nom_de_création_du_module`. L'utilisateur peut aussi créer des noms plus complexes en ajoutant un label dans un label. Nous reconnaissons ici une structure de graphe orienté acyclique, dans lequel les modules élémentaires, identifiés par leur identifiant absolu, sont les feuilles terminales et où un parcours de la racine vers un module donne un des noms du module. Contrairement à un arbre, un nœud (label ou module) peut avoir plusieurs parents, il peut donc y avoir plusieurs parcours d'un module vers la racine, et donc plusieurs noms (cf. figure 43 et 44 ci dessous)

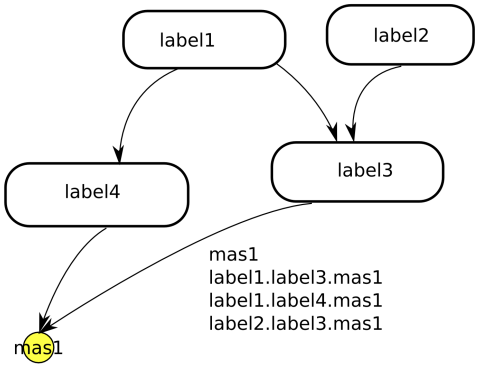


Figure 43 : Un graphe des labels et les noms engendrés

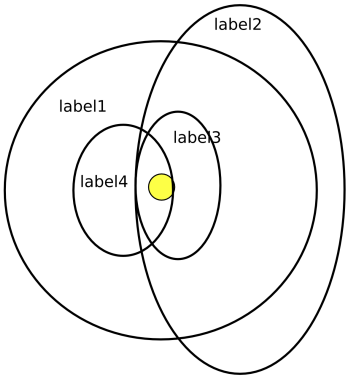


Figure 44 : Une représentation ensembliste du graphe des labels

Nous soulignons le fait que l'utilisateur crée cette structure de graphe dont découlent les noms. Il ne définit pas pour les modules des noms desquels résulterait un graphe. Le

processus est donc un processus de construction de structure, non un processus de nomination. Il y a un intermédiaire entre l'utilisateur et les noms générés, et cet étage supplémentaire rend le processus de nomination plus difficile à maîtriser.

Le maintien de la nomination absolue objective et automatique des modules élémentaires est également une nécessité. D'une part, il est prudent que l'utilisateur, pour sa propre compréhension du comportement du modèle qu'il est en train de concevoir ou qui a déjà été conçu, puisse avoir accès à la représentation de base "à plat" du réseau de masses–interactions, tel qu'il sera simulé, indépendamment de toute représentation de conception. Or notons ici que dans la philosophie CORDIS, les modules élémentaires ainsi que leur connexion, sont exactement les modules qui seront simulés. C'est en ce sens que CORDIS-ANIMA est un langage de modélisation et de simulation. C'est également pour cela que l'interprétation pour la simulation d'un réseau conçu par manipulation des modèles et des règles élémentaires est immédiate et ne nécessite pas la définition d'un langage pivot comme c'est souvent le cas dans les modeleurs-simulateurs. D'autre part, dès lors qu'un réseau en cours de création devient complexe, que ce soit en termes de nombres de modules élémentaires, ou/et en termes de topologie d'interaction, ou/et en termes de nombre de labels, il peut devenir impératif d'en avoir une image absolue, sans représentations liées aux contextes des processus de modélisation interactive, telle qu'elle correspond au plus près du langage de modélisation et de simulation CORDIS-ANIMA, qui seul est le garant de l'appréciation du contenu physique de la construction, puisque seuls sont absolument visibles à ce niveau là les principes de base de la modélisation physique : principe d'action-réaction, signification physique des modules <MAT> et <LIA>, mise au point fine des valeurs des paramètres pour atteindre le comportement voulu ou pour estimer les zones de divergence/convergence numériques, etc ...

Nous avons ici mis en évidence le besoin supplémentaire d'une nomination libre de tout ou partie quelconque du réseau, qui peut également être vue comme un regroupement libre des modules, libre car un module peut appartenir à plusieurs ensembles. Nous verrons par la suite comment les différents outils de création, de manipulation et de représentation peuvent tirer parti de ces regroupements.

1.3. Connecter les modules.

Dans un modèle composé d'un nombre relativement petit de modules et de connexions, celles-ci peuvent être effectuées une par une. C'était le cas dans ANIMA, dans lequel la taille des réseaux n'excédait pas 50 modules. On sélectionnait sur une représentation graphique les <MAT> que l'on souhaitait connecter par un <LIA> nouvellement créé.

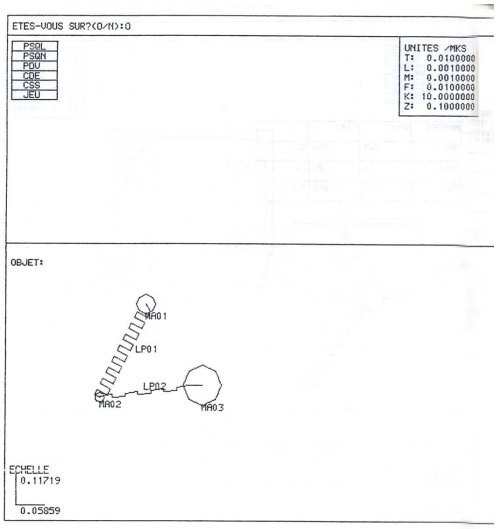


Figure 45 : Interface graphique d'ANIMA

L'arrivée d'ordinateurs plus puissants a permis de concevoir des modèles de plus grande taille. Il a alors fallu adapter les outils pour pouvoir construire et manipuler des réseaux de grande taille. Ce fut d'abord le cas avec le langage de description de modèles ANIMA-OFF, qui, par le fait de pouvoir nommer des groupes de modèles, permet également deux opérations de connexion massive :

a. Lors de la création d'un groupe de <MAT>, l'utilisateur peut spécifier une topologie de connexion, corde (figure 46 a) ou agglomérat (figure 46 b), et un type d'interaction

b. Deux groupes de <MAT> peuvent être mis en interaction selon une certaine loi d'interaction (figure 46 c).

Dans les deux cas, les ensembles d'interactions sont créés avec une restriction forte : ces ensembles d'interaction sont homogènes paramétriquement.

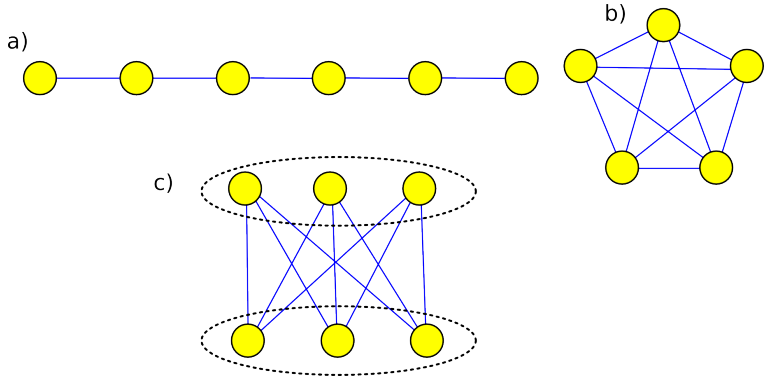


Figure 46 : Topologies de base générables avec ANIMA-OFF

Ces deux opérations sont très puissantes, puisque l'on peut exprimer un grand nombre de connexions en une seule ligne de code. Du point de vue pratique, elles sont cependant limitées car prédéfinies. Ainsi, la conception d'autres topologies, oblige à revenir à la conception point à point. D'un point de vue plus général, elles renforcent encore

davantage une structure a priori et figée du réseau, déjà présente dans la création de groupes de modules nommés. Enfin, le langage étant de type déclaratif, on ne peut pas utiliser de structure de contrôle pour décrire une topologie, l'utilisateur recourt alors à des outils externes, scripts et programmes en C, pour générer des fichiers au format ANIMA-OFF.

De son côté, MIMESIS 4 offre une unique fonction de connexion, permettant de connecter un <LIA> à deux <MAT>, mais le fait qu'il dispose de structures de contrôle permet de reproduire facilement les fonctionnalités d'ANIMA-OFF de définition de structures d'interaction ou d'interactions homogènes de groupes à groupes. Nous avons pu observer que la possibilité offerte à l'utilisateur d'écrire dans MIMESIS 4 des structures de contrôle, est d'un grand intérêt pédagogique et a été très appréciée par tout type d'utilisateur, du novice à l'expert, du technicien à l'artiste. Elle compense largement l'absence de fonctions explicites de connexion par groupe.

En conclusion, il est nécessaire de disposer d'outils pour représenter, créer et manipuler un grand nombre de connexions avec un petit nombre d'opérations. Nous avons pointé l'insuffisance d'un nombre limité de procédures prédéterminées pour effectuer ces connexions. Si les structures de contrôles de MIMESIS 4 ont montré leur intérêt, elles restent relativement simples (connecter un <LIA> à deux <MAT>, boucle pour, variables intermédiaires) et en tout cas incomplètes.

Le paragraphe suivant illustre la possibilité d'élargir la base d'outils de connexion, toujours en se basant sur le nommage des modules, que nous appellerons par la suite labellisation.

1.4. Structures répétitives et outils associés

Les réseaux CORDIS-ANIMA peuvent exhiber des topologies avec de nombreuses répétitions et/ou symétries. Par exemple, les travaux d'Arnaud Godard [LG97] sont basés sur des structures atomiques, linéiques, surfaciques et volumiques (cf. figure 47). Ces atomes sont par la suite utilisés comme motif de base pour obtenir des structures plus grandes, par exemple pour modéliser l'écroulement d'un mur (cf. figure 48).

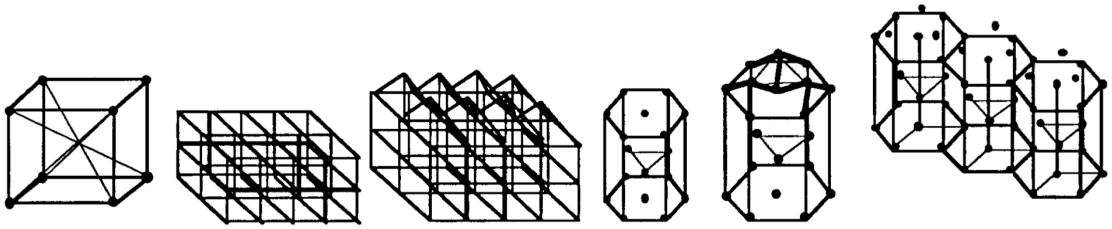


Figure 47 : Quelques atomes volumiques

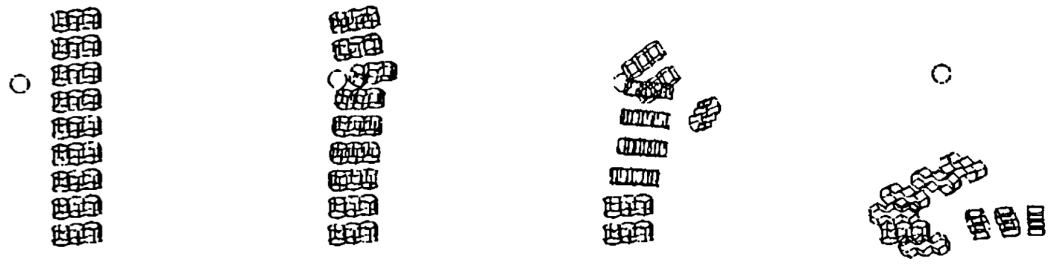


Figure 48 : Un mur, fracturé, s'écroule

Dans un autre domaine, les modèles de tissus présentent eux aussi de fortes régularités. La topologie du modèle de drapeau, présentée en figure 49, est l'exemple élémentaire de ce type de modèle. Dans tous ces modèles le point commun est la répétition d'un motif. La notion de motif est directement compréhensible dans le domaine graphique, elle l'est beaucoup moins dans le domaine des réseaux masses-interactions. Il faut en effet distinguer clairement le réseau de sa représentation graphique. Cette distinction faite, le motif peut correspondre à une sous-partie autonome représentée par un réseau de masses et d'interactions simulable, appelé par A. Luciani (Cours CORDIS) « Réseau Bien Formé » c'est à dire qui n'a pas de <LIA> dont un des points ne serait pas connecté, rendant ainsi le réseau non calculable. On pourrait dire qu'il s'agit d'une vision « objet » qui dès lors ne peut être mis en relation avec d'autres que via des nouvelles interactions. Les briques hexagonales d'Arnaud Godard sont assemblées selon cette vision, à deux niveaux différents. Au premier niveau, les briques sont reliées par des ressorts de longueur à vide nulle. Au deuxième niveau, les briques sont reliées par des interactions de cohésion irréversible, permettant ainsi d'obtenir des phénomènes de fracture, comme illustré en figure 48. Ces mêmes motifs peuvent être interprétés différemment, en s'affranchissant de la vision objet. On peut alors concevoir de créer la structure répétitive en reliant les motifs par des interactions de manière à créer des motifs intermédiaires entre deux motifs.

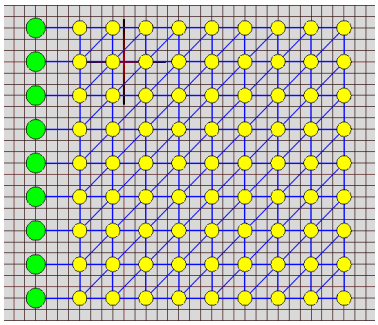


Figure 49 : Une topologie pour un modèle de drapeau

Un motif peut aussi être défini très différemment comme une sous-partie de réseau présentant des points L non connectés. Une telle définition est problématique car la création du motif suppose des demi-connexions de <LIA>. Bien que ceci ne pose aucun problème ni technique, ni formel dès lors qu'à la simulation, toutes les interactions seront complètement connectées, cela a comme inconvénient d'occulter le principe d'action-réaction pour le modélisateur. En effet, une interaction est sensée relier deux masses

entre lesquelles circulent une force. Rendre une extrémité de l'interaction « pendante », c'est à dire non connectée à une masse, ne permet pas de penser la force comme circulante entre deux masses. En modélisation physique, le motif est matière. Deux motifs ne se superposent donc pas comme en graphique : une masse n'est pas un sommet et une interaction n'est pas une arête. Pour combiner deux motifs en gardant cette image graphique, il faut expliciter des opérations de fusion et de suppression. Fusionner deux masses A et B revient à supprimer B et à rediriger vers A toutes les interactions relatives à B. Si il existe une interaction entre A et B, elle devra être supprimée. Cette opération qui permet de créer une nouvelle topologie à partir d'une topologie existante par confusion d'éléments <MAT> et reconnexion d'éléments <LIA>, n'est donc pas triviale du point de vue physique. La figure 50 illustre ce processus de fusion et montre ainsi le début d'un processus de construction de drapeau.

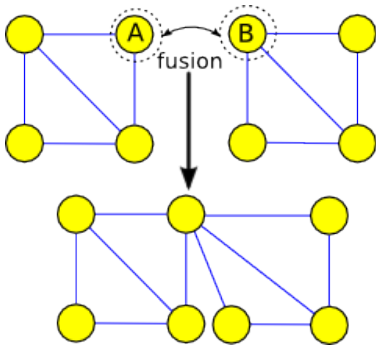


Figure 50 : Fusion de deux modules <MAT>

En résumé, notre analyse dénombre deux approches différentes pour combiner une série de motifs identiques :

1 Créer de nouvelles interactions pour connecter les motifs
   a. En insérant ainsi de nouveau motifs intermédiaires
   b. En recréant des motifs identiques entre les motifs
2 Fusionner des sous-parties de motifs

Quoi qu'il en soit, la création de telles structures ne peut être comparée à des opérations géométriques. Générer de la matière en interaction est plus complexe que générer une forme par extrusion, ou autre opération de prolongement spatial d'un motif. Cet aspect de la modélisation n'a jamais été étudié systématiquement. Certaines solutions ont été adoptées ponctuellement, mais elles restent relatives à un cadre technique et un objectif de modélisation bien déterminés. Or, la problématique des structures répétitives implique deux besoins : l'un de pouvoir dupliquer des structures, l'autre de pouvoir les mettre en relation. Le premier besoin a été abordé dans MIMESIS 4. Nous en faisons l'analyse dans le paragraphe suivant. Le deuxième besoin n'a été abordé en tant que tel dans aucun environnement de modélisation, nous tenterons d'y apporter une réponse dans la deuxième partie de ce chapitre.

1.5. Duplication de sous-parties de réseau

La fonctionnalité « dupliquer un label » introduite dans le langage MECA de MIMESIS 4 permet de répondre au premier besoin puisqu'il permet de dupliquer les modules contenus dans un label et de reproduire toutes les connexions internes. Mais le problème principal de la duplication ne réside pas dans la duplication elle-même mais dans les noms des duplicata et de leurs modalités de création. Dans MIMESIS 4, les modules élémentaires du duplicata héritent des noms absolus du modèle dupliqué avec pour préfixe le nom du nouveau label (ex: le module nommé `mas1` et `label1.mas1` aura un duplicata nommé `label2.mas1` si on duplique le label `label1` dans le label nommé `label2`). Pratiquement, c'est tout le graphe issu du label dupliqué qui est copié dans le nouveau label. Cette opération est donc doublement intéressante, 1) elle duplique une sous-partie du réseau, qui dans l'exemple ci-dessus correspondrait à un sous-réseau calculable de modules interconnectés uniquement entre eux et 2) elle crée des noms pour tous les modules de ce nouveau sous-réseau, permettant ainsi de modifier sa topologie, de le relier au reste du réseau, de le paramétrer, de l'habiller. Cette opération révèle un aspect important du système de labellisation de MIMESIS 4 : la confusion entre le nom de création et le nom absolu d'un module. Nous avons insisté sur l'importance d'un nom absolu, du point de vue du système comme du point de vue de l'utilisateur. Ce nom joue le rôle d'identifiant unique, alors que le nom de création est un nom d'usage pour le modélisateur MIMESIS 4. Il lui sert de repère dans sa tâche de création du réseau. La procédure dupliquer poursuit cette confusion en se basant sur ce nom de création pour créer les nouveaux noms, conditionnant ainsi la structure de noms du duplicata à celle des modules originaux. Ceci constitue une limitation importante, car le modélisateur peut légitimement souhaiter nommer différemment les duplicatas, que ce soit pour les connecter, les paramétrer ou encore pour gérer l'habillage du modèle. Le modélisateur reste finalement tributaire de la structuration qu'il crée avec les premiers modules en leur donnant des noms et en les créant sous formes de tableaux (ex: `MAS mesMasses[10]`).

Le concept de duplication est donc intéressant, voire indispensable, mais il doit être accompagné d'un procédé permettant de donner un nom absolu aux modules du duplicata. Cela permettrait ensuite de leur appliquer de nouveaux labels ou de les inclure dans d'autres labels, de manière à maintenir une indépendance structurelle entre les modules dupliqués et leurs duplicatas.

De manière générale, nous constatons avec ces deux exemples que certaines opérations sur des parties de réseau peuvent être intéressantes pour aboutir à des topologies complexes. Cependant ces opérations doivent être soutenues par un système de labellisation à la fois souple et cohérent :

1. souple, le système de labellisation de MIMESIS 4 ne l'est pas assez puisque l'on voit sur les exemples ci-dessus qu'il ne fournit pas les outils qui pourraient être utiles pour définir des structures organisées complexes.
2. cohérent, le système de labellisation de MIMESIS 4 ne l'est pas assez non plus puisque la simple opération de labellisation contredit certaines propriétés liées à la création de modules.

1.6. Manipulations des paramètres physiques

Dans leur ensemble, les environnements de création de réseaux CORDIS-ANIMA développés précédemment proposent de fixer la valeur d'un paramètre pour un groupe de module de même type : valeurs associées à tous les modules d'un groupe nommé dans ANIMA-Off, valeur associée à une désignation graphique dans MIMESIS 4 ou valeurs associées aux modules d'un label via une sélection dans une représentation de l'arbre des labels dans MIMESIS 4.

Ainsi dans MIMESIS 4, le système de labellisation n'est utilisé que pour sélectionner des modules ou des ensembles de modules, la donnée des valeurs des paramètres de ces modules s'effectuant ensuite une par une. De plus, dans MIMESIS 4, la représentation graphique fonctionnelle du réseau est superposée à une représentation spatiale dans laquelle les masses sont à leur position initiale. La fonction de l'espace graphique était donc double : représentation de la topologie et représentation des positions initiales. Cet amalgame ne permet pas à l'utilisateur de disposer ses modules comme il l'entend pour mieux représenter la topologie et une structure du réseau qui lui convient pour la tâche qu'il veut accomplir : par exemple pour les sélectionner en fonction des règles sur les paramètres qu'il souhaite appliquer.

D'une manière plus générale, le passage à la complexité en termes de travail sur les paramètres adresse encore une fois les outils de désignation des groupes de modules. L'activité sur la manipulation des paramètres physiques comporte deux aspects :

- Qualitativement : pouvoir s'intéresser à un ensemble de paramètres physiques et non à un seul.
- Quantitativement : pouvoir définir des lois sur ces paramètres.

Dans les versions précédentes et actuelles de MIMESIS, la modalité textuelle, ou plus généralement alphanumérique, n'a jamais été associée au système de labellisation pour la tâche de paramétrage des modules. Cette éventualité est pourtant intéressante à plus d'un titre : 1) elle permettrait d'effectuer des calculs pour obtenir la valeur d'un paramètre 2) elle permettrait de sélectionner un groupe de modules à partir d'une chaîne de caractères complexes faisant référence au système de labellisation.

MIMESIS 4 propose un calcul automatique de paramètre pour les longueurs au repos. L'utilisateur dispose de cette valeur et peut, s'il le souhaite, la prendre comme longueur au repos de l'interaction. Derrière cette commodité, il y a un concept plus large : celui d'affecter une valeur à un paramètre d'un module en fonction des valeurs initiales – voire des paramètres - d'autres modules.

Nous identifions trois tâches distinctes dans ce concept : 1) la sélection des modules selon certains critères (topologiques, spatiaux, paramétriques) ; 2) la récupération de certains paramètres de ces modules ; 3) le calcul d'une valeur à partir des paramètres récupérés. Dans l'exemple que nous avons décrit, les modules sont choisis sur un critère topologique simple (les <MAT> qui lui sont connectés), on récupère leurs positions initiales et on calcule la distance entre ces deux positions Pour généraliser cette fonctionnalité, il faudrait donc offrir des outils d'interrogation du réseau permettant de poser des questions comme "quels sont les modules connectés au module A ?" (critère topologique) ou "quels sont les <MAT> dont les positions initiales sont circonscrites au

cercle de centre (x,y) et de rayon r ?" (critère spatial). Remarquons que de tels outils pourraient également être intéressants pour agir sur la topologie du réseau.

La tâche de paramétrage requiert donc d'une part des outils de sélection adaptés, et d'autre part des outils de calcul, qui doivent être suffisants pour réaliser la majorité des tâches. Nous ne recherchons pas l'exhaustivité car il ne serait pas élégant d'intégrer dans MIMESIS toutes les possibilités d'un tableur ou d'un logiciel de calcul numérique. La fonctionnalité d'interopérabilité avec des logiciels plus adaptés est plus optimale. Ainsi, si un besoin de calcul complexe se fait sentir, l'utilisateur fera appel à un logiciel tiers (Matlab, Excel) pour lequel nous proposons de mettre en place les procédés d'importation et d'exportation adaptés.

1.7. Synthèse

Les différents environnements de modélisation dédiés à CORDIS-ANIMA se sont succédés selon une logique historique qui est éclairante pour situer les intentions et les buts que nous nous somme fixés pour MIMESIS V. ANIMA, premier environnement développé en 1984, a illustré dès le départ le besoin d'un environnement interactif pour la modélisation. Les plateformes matérielles de l'époque, ainsi que son orientation temps réel, circonscrivaient son usage à des modèles de petite taille, mais très variés au niveau topologique et paramétrique, ainsi que dans le plongement géométrique. En 1990, Luciani et Jimenez veulent aller vers plus de complexité par rapport à la taille du modèle et expérimenter des interactions complexes. Ils laissent donc de côté l'interactivité, le temps réel ainsi que de la séparation entre la modélisation structurelle (PSQL) et la définition des paramètres quantitatifs (PSQN) pour mettre au point ANIMA-OFF, qui permettra de montrer la généricité du formalisme masses-interactions (phénomènes granulaires et plastiques, émergence grâce à une taille critique). Cependant la perte de l'interactivité et la conception de ce langage pour une classe de modèle déterminée (agglomérats, objets structurés en interaction) limitaient les possibilités exploratoires et créatrices de CORDIS-ANIMA. Le projet MIMESIS a ainsi vu le jour, avec comme premier objectif de concilier interactivité et conception de modèles de grande taille. La première tentative stable, MIMESIS 4, propose un langage de script, un système de labellisation et une collection d'interactions très complète (comme nous l'avons rappelé au chapitre précédent). Similairement aux premières versions d'ANIMA, elle distingue les quatre phases dans la création d'un modèle physique : la définition de la structure du réseau, la définition des paramètres, la définition des conditions initiales, la simulation. MIMESIS 4 est donc la première version intégrant les besoins fondamentaux. MIMESIS 4 a connu un grand succès auprès des utilisateurs, leur permettant de découvrir et d'exercer la modélisation physique particulaire avec un champ de possibilités déjà très large. Mais assez vite, de nouvelles limitations sont apparues, suite à la quantité, qualité et complexité des modèles désormais imaginables : le langage de script est circonscrit à l'édition de la topologie, le système de labellisation n'est pas vraiment complètement opérationnel et pourrait être étendu et généralisé, les manipulations graphiques ne sont pas assez évoluées pour la manipulation de grands modèles. Il nous faut donc aller plus loin, passer à un autre stade dans le domaine des interfaces interactives pour la création de modèles masses-interactions pour le mouvement.

Notre projet est de proposer les bases d'une interface permettant de franchir de nouveaux caps de complexité dans les mouvements produits : Concilier efficacité, convivialité et puissance d'expression au sein de cette interface, qui s'inscrit dans un projet plus large, visant à proposer une suite logicielle pour la modélisation masses-interactions pour le mouvement, la musique et l'interaction gestuelle. Les différents logiciels de cette suite sont conçus pour être interopérables, et constituer ainsi un outil pour la multisensorialité. Cette collection partage donc une base commune : CORDIS-ANIMA bien entendu, mais également tout le noyau fonctionnel.

Nous avons insisté sur la nécessité d'un langage de script pour la modélisation physique. Celui-ci tient une place centrale dans notre projet, MIMESIS V, mais également dans toute la suite logicielle. Il est en effet indispensable de disposer d'un langage de modélisation puissant et générique pour ouvrir la voie de la complexité : modèles de grande taille, connexions complexes requérant des structures de contrôle, etc... Le système de labellisation, permettant de nommer les modules, est le compagnon indispensable du langage de script, puisqu'il permet de désigner les modules textuellement. Ces deux éléments, langage de script et système de labellisation, sont au cœur de la refondation de MIMESIS. Cependant, MIMESIS V reste un environnement interactif, et donne donc une large part aux manipulations graphiques. Ainsi, la modalité graphique est et restera la modalité privilégiée pour les modèles de petite taille. Par conséquent, les représentations doivent être multiples tout en restant pertinentes et malléables par rapport aux besoins du modélisateur.

2.Les piliers de MIMESIS V

2.1. Analyse des tâches

MIMESIS V est construit, comme la majorité des interfaces actuelles, sur le modèle WIMP (Window, Icon, Menu, Pointer). C'est sur ce modèle que nous allons poser les piliers pour le nouvel environnement de modélisation et de simulation. Comme tout environnement de conception interactif, il se compose classiquement d'une interface et d'un noyau fonctionnel. Ce dernier contient les abstractions fonctionnelles qui caractérisent la méthode de modélisation, en d'autres termes les objets d'intérêt de MIMESIS et de CORDIS-ANIMA, qui devront être interactivement manipulables depuis l'interface.

Les abstractions fonctionnelles caractérisant la méthode MIMESIS sont de quatre types :

1. Les premiers sont, bien entendu, les modules physiques élémentaires, <MAT>, <LIA> et autres modules, entrant dans la modélisation structurelle d'un réseau CORDIS.
2. Les seconds sont les grandeurs scalaires physiques, comme par exemple la raideur d'un module <REF> ou l'inertie d'un module <MAS> (cf. Annexe 1 pour la liste des modules <MAT>). On dénombre trois sous-types : la raideur (aussi appelée élasticité), la viscosité et l'inertie. Ces grandeurs sont fortement typées, c'est à dire qu'on ne pourra pas affecter directement la valeur d'une inertie au paramètre viscosité d'un module. La fréquence de simulation est elle aussi une grandeur physique, commune à l'ensemble d'un modèle.
3. Les troisièmes sont les objets spatiaux. Ils sont constitués d'une base formée par les distances (scalaires positifs), les distances algébriques (scalaires), les vecteurs 3D et les points 3D. Les éléments de cette base sont homéomorphes ou dérivent des variables d'état extensives comme par exemple les positions initiales des modules physiques.
4. Les quatrièmes sont les formes visuelles, qu'il s'agira d'associer à des évolutions temporelles calculées par les modules physiques, et qui donc permettront d'habiller spatialement les mouvements calculés physiquement.

Les grandeurs physiques et les objets spatiaux sont incorporés dans l'abstraction principale, le module. Du point de vue du noyau, ces deux abstractions sont donc d'abord des attributs des modules. Elles désignent pour les premières les paramètres physiques non spatiaux et pour les secondes les conditions initiales et les paramètres physiques spatiaux. Cependant, les grandeurs physiques comme les objets spatiaux peuvent

également exister en dehors des modules. Prenons le cas élémentaire d'un copier/coller sur la raideur d'un module ressort-frottement. La copie de cette grandeur physique la sort du module pour la mettre dans le presse-papier, où elle existe en tant que raideur. Ainsi, il sera impossible d'affecter cette grandeur à un autre paramètre qu'une raideur. Cette possible indépendance entre grandeurs physiques et objets spatiaux d'une part et modules d'autre part est également très utile pour les procédures d'importation et d'exportation. Nous en donnerons une application concrète dans le chapitre suivant.

Ces quatre types d'abstractions (nous dirons parfois "objets" au sens informatique du terme) ont en commun un même problème de base : pour les manipuler il faut les adresser. L'adressage de bas niveau, c'est à dire les adresses mémoire de ces entités, est évidemment inadéquat au niveau d'un interface de modélisation interactive. Il est donc nécessaire d'introduire une dernière catégorie d'abstractions, les noms, qui permettront à l'utilisateur depuis l'interface de modélisation interactive, de représenter et de manipuler ces adresses.

Les actions basiques que l'on doit pouvoir effectuer sur ces abstractions sont relativement simples et générales:

- créer module (nombre et type)
- connecter (point L et point M) (un <LIA>, deux <MAT>)
- paramétrer (un paramètre physique, un module, une valeur)
- Définir (un type de condition initiale, un <MAT>, une valeur)
- supprimer (un module)
- nommer (un module, un nom) (une forme, un nom)
- renommer (un nom, un nom)
- supprimer un nom (un nom)
- créer une forme (un type)
- paramétrer une forme (une forme, un paramètre, une valeur)
- associer une forme à des modules <MAT> (une forme, une liste de <MAT>)
- supprimer une forme

Toutes ces actions impliquent une ou plusieurs actions, qui sont majoritairement des actions de sélection :

- sélectionner un type de module
- sélectionner un point L
- sélectionner un point M
- sélectionner un module
- sélectionner un type de paramètre physique
- sélectionner un nom
- sélectionner un type de condition initiale
- sélectionner une forme

- sélectionner un type de forme
- sélectionner un paramètre de forme
- sélectionner une liste de <MAT>
- etc.

Elles impliquent aussi des actions de quantification pour spécifier une valeur ou un nombre. Notons que ces actions peuvent revenir à une action de sélection, c'est à dire la sélection d'une grandeur physique ou d'un objet spatial particulier.

Les actions de base sont atomiques, car elles n'agissent que sur un objet. Pour concevoir des modèles de taille moyenne ou grande, des actions "macro", agissant sur plusieurs objets, sont indispensables. En effet, l'absence de telles actions contraint l'utilisateur à exécuter un grand nombre de fois la même tâche. Ce premier changement d'échelle en implique donc un second au niveau des actions de sélection : sélectionner plusieurs points L, sélectionner plusieurs modules, sélectionner plusieurs noms, ...

Nous avons spécifié ces différentes actions en dehors de toute modalité, d'une manière proche des méthodes du noyau fonctionnel, mais cependant compréhensible du point de vue utilisateur. L'interface propose plusieurs modalités, au sens de moyens d'action, pour effectuer ces tâches primaires.

Les trois premières sont au cœur de toute interface WIMP, principalement basée sur la manipulation directe :

1. Un espace graphique de sélection associé à des boîtes de dialogues et des widgets
2. Un espace graphique associé à des gestes (glisser & déposer principalement) dont la signification change selon l'outil choisi dans une palette
3. Des boîtes de dialogue dédiées à une action.

La dernière modalité est la modalité script, dont l'idée primitive se trouve dans la description textuelle de la topologie du réseau dans MIMESIS 4, et qui a été par ailleurs exploitée plus largement dans MIMESIS Forme [Cha04]. C'est une modalité langagière qui n'est donc pas directement compatible avec les principes d'une interface WIMP, d'abord basé sur l'action. La complexité que doit permettre d'atteindre MIMESIS V est inaliénable d'un langage de script central dans l'interface, et non optionnel comme c'est le cas dans la majorité des interfaces WIMP. La spécification de ce langage a donc occupé une part importante de nos travaux. La question de la synergie entre les modalités graphique et textuelle, que l'on peut voir du point de vue des interfaces WIMP comme la question de l'intégration d'un langage de script, a constitué le principal verrou technologique que nous devions comprendre et tenter de résoudre. Nous proposons d'abord de présenter les principaux outils de MIMESIS V, pour que cette problématique soit située le cadre bien particulier de la modélisation physique particulaire.

2.2. Un langage de script : PNSL (Physical Network Scripting Language)

Le langage PNSL est le premier langage de script générique dédié à la modélisation avec les réseaux physiques masses-interactions, tels que définis avec CORDIS-ANIMA. PNSL est un *langage de script*. A ce titre, à l'inverse de langage existants tels que PML (Physics Model Language) [CP04], son premier objet n'est pas de permettre une description "à plat" du modèle, mais bien d'être utilisé dans l'activité de modélisation elle même. Il permet de programmer la conception du modèle. Cette programmation concerne toutes les étapes de la conception :

- Construction et édition de la topologie du réseau
- Edition des paramètres physiques des modules
- Edition des valeurs initiales des modules <MAT>
- Gestion de la simulation
- Habillage des trajectoires

En plus de ces étapes, PNSL permet également de gérer le système de labellisation (ajouter des noms, supprimer des noms). Il en sera de même pour tous les outils de manipulation du réseau que nous pourrons définir par la suite. Cela signifie d'ailleurs que ces outils sont considérés comme des objets d'intérêt, manipulables et enregistrables, à côté de l'objet d'intérêt principal que constitue le réseau CORDIS-ANIMA et son habillage.

Mais le langage n'a pas seulement une fonction de construction, d'édition du réseau, il a également une fonction épistémique. L'utilisateur doit pouvoir interroger le réseau sur sa topologie, ses paramètres. La fonction épistémique concerne aussi les autres objets d'intérêt de MIMESIS : l'habillage et sa relation au réseau, les différentes simulations, le système de labellisation, les objets spatiaux, ...

La puissance de PNSL réside principalement dans sa nature impérative, synonyme de structures de contrôle (boucles et structure conditionnelles). Cette possibilité démarque fortement la modalité script de la modalité graphique, car elle permet de spécifier de manière efficace et condensée un grand nombre d'actions répétitives.

La place centrale que nous accordons à PNSL au sein de MIMESIS se reflète dans cette partie, car tous les autres piliers s'appuient sur ce premier. En effet, PNSL définit un style d'interaction centré sur l'action (Action Object Interface [Shn87]) qui est au même niveau que le modèle WIMP, centré sur l'objet (Objet Action Interface). Les deux styles d'interaction doivent supporter les outils que nous présentons par la suite. Ces outils ont donc à la fois une manifestation graphique et une manifestation langagière.

2.3. Un système de labellisation : créer un espace de noms cohérent

Nous avons énoncé le besoin d'un système de labellisation pouvant supporter une activité de nommage souple des modules. Une préoccupation majeure dans la mise au

point de ce système était d'éviter qu'il constitue une structuration du réseau. En effet, la notion de conteneur de modules (le label au sens de MIMESIS 4) est structurante, les modules *appartiennent* à des ensembles. Cependant, un même module pouvait être ajouté dans plusieurs conteneurs, et les modules contenus ne formaient pas nécessairement un sous-réseau bien formé. Mais le centrage de l'activité de l'utilisateur sur la définition d'ensembles de modules est structurant, car il porte l'attention sur la structure davantage que sur les noms qu'elle génère. Nous avons donc redéfini le label comme une chaîne de caractères référençant un module, remettant ainsi au centre le nom plutôt que le conteneur. Du point de vue de l'utilisateur, il nomme directement des modules plutôt que les placer dans un conteneur. Cependant, la notion de conteneur réapparaît dans cette activité de nommage à travers la notion de préfixe commun, absolument nécessaire par rapport à la tâche de groupement de modules. Le système de labellisation est donc conçu pour faciliter la gestion des préfixes et propose de structurer les noms à l'aide du caractère séparateur "/". L'utilisateur peut définir des labels comme `/monNonObjet/masse1` ou `/monGroupe1/monSousGroupe2/masse8`, avec l'obligation de commencer le label avec le caractère "/". Expliciter ainsi les préfixes permet de retrouver la notion de conteneur, une chaîne de caractères entre deux "/" définit le nom d'un conteneur. Mais à la différence des conteneurs de MIMESIS 4, ceux ci ne contiennent pas des modules mais des références vers des modules ou bien d'autres conteneurs. Notre système de labellisation est donc un arbre, comparable à l'arborescence d'un système de fichiers. Les nœuds de l'arbre sont les conteneurs, ses feuilles des références vers des modules. Les branches sont décorées avec des noms contextuels, ainsi un parcours depuis la racine de l'arbre jusqu'à une feuille donne un label complet. En passant d'un conteneur de modules à un conteneur de références vers des modules nous avons brisé le caractère structurant du système de labellisation, un module n'*appartient* plus à un conteneur, il n'y est que référencé, confirmant ainsi le rôle de l'outil conteneur comme un moyen de grouper des modules. Le système est donc plus souple car il permet de référencer un même module avec deux labels ayant même préfixe. D'un autre point de vue, le choix d'une structure d'arbre plutôt que de graphe pourrait être interprété comme un choix moins générique et donc permettant moins de complexité. En réalité, le passage d'un conteneur de modules à un conteneur de références disqualifie le graphe, dont le rôle premier était de pouvoir définir un module comme appartenant à plusieurs ensembles différents. Plus encore, une structure de graphe permet de définir un sous-ensemble appartenant à deux ensembles distincts, cette possibilité offerte par le graphe était intéressante car l'utilisateur pouvait ainsi définir un sous-réseau appartenant à deux sous-réseaux différents, par exemple le sous-réseau "roue" appartient à la fois au sous-réseau "voiture" et au sous-réseau "roue sur une route". Si ces caractéristiques peuvent paraître intéressantes, elles ont le désavantage d'appuyer une vision "objet" du réseau, en effet l'utilisateur manipule des conteneurs et non plus des noms et les assimile à des sous-réseaux. Mais le graphe est surtout victime de la trop grande complexité qu'il engendre : si l'utilisateur ne maîtrise pas complètement le graphe qu'il a construit, il peut créer des noms sans le vouloir et donc faire des erreurs dans la sélection des modules. La structure d'arbre a donc été choisie car plus simple et suffisante pour gérer des noms complexes.

Dans un objectif d'un processus de modélisation complexe, dans lequel les noms donnés par l'utilisateur à des modules sont éphémères, car liés à une tâche précise, il nous est nécessaire de disposer d'un système de labellisation souple et cohérent. Il est donc légitime d'utiliser des préfixes avec une forme déterminée par le caractère séparateur

"/", plutôt qu'un système totalement libre, sans caractère séparateur prédéfini. Nous avons fait ce choix principalement pour des raisons d'implémentation. En effet, un système totalement libre signifie qu'il n'y a aucune structure, juste un ensemble de noms. Cela a pour conséquence une faible optimisation qui sera critique pour les grands modèles En effet, une absence de structure réduit considérablement les outils pour agir sur cet ensemble de noms. On ne peut par exemple pas renommer simplement un conteneur. L'action « renommer un ensemble de labels » est toujours possible mais elle est très couteuse car elle suppose de modifier individuellement tous les labels de l'ensemble. Par ailleurs l'absence de structure pose aussi un problème de représentation graphique préoccupant, car la seule option est alors de représenter l'ensemble des noms sous la forme d'une liste, forme peu adaptée pour la manipulation de milliers, voire de millions de noms.

Par ailleurs, le système de labellisation gère à la fois les labels utilisateurs, c'est à dire définis par l'utilisateur, et les labels automatiques, qui représentent le nom absolu objectif d'un module et qui commencent par le caractère "@". Ces derniers sont générés à la création du module et ne peuvent pas être modifiés pendant la vie du module. Les noms automatiques ne sont pas forcément créés par le système, l'utilisateur a la possibilité de l'expliciter lui même lors de sa création d'un ou plusieurs modules, sous la condition que ce(s) label(s) automatiques n'existent pas déjà. Cette distinction entre le système de noms donnés par l'utilisateur («Personymic Names ») et celui donné par le système (« System Legal Identifier ») est fondamentale, car elle permet de séparer la fonction du nom en tant qu'identifiant unique de la fonction d'outil de manipulation, de regroupement. On se protège ainsi des problèmes fréquemment rencontrés avec MIMESIS 4 ou CORDIS-OFF, dans lesquels les deux fonctions étaient confondues, le nom de création donné par l'utilisateur tenant également le rôle d'identifiant unique.

La puissance du système de labellisation apparaît dans son utilisation avec PNSL. En effet, la majeure partie des procédures PNSL prend en argument une sélection de modules. Cette sélection de modules est produite par une expression de sélection de labels (ESL), équivalente à une liste ordonnée de labels. Une ESL est exprimée à l'aide d'outils évolués : expressions régulières, opérateurs de filtre, opérateurs de tri, opérateurs ensemblistes. Nous avons là un outil puissant permettant de regrouper et d'ordonner des modules de manière avancée, étape indispensable pour une manipulation complexe du réseau. Mais le système de labellisation reste uniquement un outil, nous le réaffirmons : le système de labellisation n'est pas le réseau ! D'abord ce système ne référence que les modules, il ne représente pas les connexions. Ensuite ce système propose des structurations du réseau, mais aucune ne correspond à la réalité du réseau puisque celui-ci n'est pas structuré. Ces structurations ne sont que des outils pour accomplir des tâches intervenant dans la construction du réseau.

2.4. Filtres et opérateurs ensemblistes

Parmi les fonctions épistémiques, les questions "quels sont les modules vérifiant les propriétés A et/ou B et/ou C et/ou ..." tiennent une place importante. La formulation de ce type de question révèle en elle même le premier outil qu'elles nécessitent. En effet les conjonctions "et" et "ou" sont équivalentes à des opérateurs logiques, mais comme nous opérons sur des ensembles de modules, ils sont équivalents aux opérateurs ensemblistes

"intersection" et "union". En revanche, ce type de question reste très général sur la constitution des ensembles : ils vérifient une propriété. Le paragraphe précédent nous a permis d'introduire une de ces propriétés : les modules sont associés à des labels qui correspondent à une expression régulière donnée. Afin d'introduire d'autres propriétés de manière générique, nous introduisons la notion de filtre. Concrètement, dans sa définition la plus primaire, un filtre est une procédure PNSL qui prend en entrée un label et qui retourne un booléen. Les filtres les plus basiques portent sur le type de module (e.g. le filtre <REF> renvoie vrai si le module désigné par le label est de type <REF>). La notion de filtre est cependant plus générique et renvoie aux autres fonctions épistémiques concernant non seulement le réseau (ses paramètres, sa topologie) mais aussi tous les autres objets d'intérêt auquel il est relié dans MIMESIS : l'habillage, les simulations. Un filtre peut donc accepter d'autres arguments que le label, de manière à pouvoir répondre à des questions comme "quelles masses ne dépassent pas la vitesse v au cours de la simulation x" ou bien "quelles interactions <REF> ont un paramètre de raideur compris entre 0.1 et 0.2". Notre objectif n'est pas de définir un ensemble fini de filtres mais de définir suffisamment largement cette notion pour permettre à notre système d'être étendu le plus souplement à l'avenir.

Nous avons défini des outils conçus spécialement pour une modalité basée langage (script ou autre). Mais les tâches que ces outils permettent d'accomplir peuvent également être accomplies graphiquement, et donc de manière beaucoup moins formelle. Les espaces graphiques que nous présentons dans le prochain paragraphe constituent le support adapté pour réaliser graphiquement ces tâches.

2.5. Des espaces graphiques pour la sélection, la manipulation et la visualisation

Comment représenter graphiquement un réseau CORDIS-ANIMA, et surtout pour quoi faire ? Comme nous l'avons souligné précédemment, on dénombre cinq activités principales relatives à l'espace graphique :

1. Sélectionner des modules
2. Manipuler la topologie du réseau : ajouter des modules, en supprimer, connecter un <LIA> à deux <MAT>, connecter des points L à des points M
3. Percevoir et manipuler les valeurs initiales des modules <MAT> du réseau
4. Percevoir la topologie du réseau dans son ensemble et dans ses détails
5. Organiser le réseau

Les activités 1) et 5) sont complémentaires, au même titre que la formation d'expressions de sélection de labels et la déclaration de labels. En effet, le système de labellisation définit une sémantique alphanumérique là où l'organisation du réseau dans un espace 2D ou 3D définit une sémantique spatiale. Il faut donc une, voire plusieurs, représentation graphique adaptée à ces deux activités, et dont la dimensionnalité (2D ou 3D) convienne. Pour que cette activité d'organisation spatiale soit le plus en phase avec l'activité de modélisation il est indispensable de séparer la représentation des positions

initiales de la représentation de la topologie, contrairement au choix fait dans ANIMA et MIMESIS 4. C'est donc un espace et une représentation graphiques dédiés à la topologie que nous proposons, adapté également aux activité de perception et de manipulation de la topologie du réseau (activités 2) et 4)). Cet espace est inspiré de l'établi de GENESIS : un espace de travail 2D, dont les deux dimensions n'ont aucune influence sur la simulation physique. L'utilisateur y dispose ses modules comme il l'entend, selon ses besoins dans le processus de modélisation. En raison de la grande taille et de l'importante versatilité des modèles, les moyens de navigation sont cruciaux. Nous nous sommes inspirés des travaux sur les interfaces utilisateur zoomables (ZUI) [Zia07] pour proposer des modalités adaptées. Il est à noter que pour la plupart des environnements orientés signaux, comme Simulink [Sim] ou Max [Max], le zoom est rarement une fonctionnalité centrale. L'encapsulation lui est usuellement préférée. Ce mécanisme tend à hiérarchiser l'objet d'intérêt et à favoriser une vision objet, c'est pourquoi nous ne l'avons pas privilégié. Nous reviendrons plus tard sur ce point complexe.

Cet espace graphique, que nous appelons **espace topologique**, est associé à une palette d'outils, présentant les modules CORDIS-ANIMA et différents outils de manipulation (sélection, suppression, navigation). Les représentations graphiques des modules sont naturellement identiques sur la palette et sur l'espace topologique. Ainsi, un <MAT> est représenté par un disque et les <LIA> par des segments reliant les deux <MAT> auxquels il est connecté. A la création d'un <LIA>, l'utilisateur doit sélectionner ces deux <MAT> sur l'espace topologique (cf. figure 51), il ne peut pas créer d'interaction déconnectée. En revanche, il peut déconnecter des points L, pour les reconnecter à des points M, voire les laisser *pendants* (i.e. déconnectés). Un point L pendant est représenté graphiquement par un cercle blanc. Mais la notion de point de communication n'est pas primaire dans MIMESIS, c'est la notion de module qui prime, le point de communication est une notion évoluée utile pour les modèles complexes.

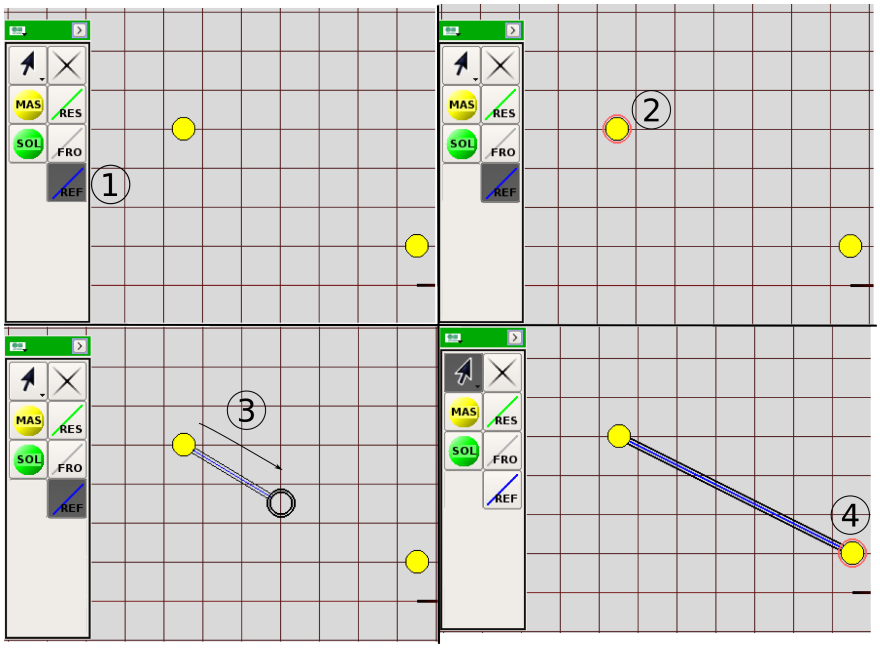


Figure 51 : Les quatre étapes de la création et connexion d'un <LIA>

La libre disposition des modules permet de 1) visualiser une topologie particulière à un sous-ensemble de modules (par exemple, disposer en cercle les masses d'un agglomérat permet de constater qu'elles sont toutes interconnectées) 2) placer des groupes de modules pour les repérer.

GENESIS est dédié à la modélisation de réseaux unidimensionnels (c'est à dire dont les variables d'état sont scalaires), c'est pourquoi un tel espace convient parfaitement. En revanche, les topologies des réseaux MIMESIS, 3D pour l'image animée, présentent une complexité qui peut se révéler gênante dans une représentation uniquement 2D. Ces cas ne sont pas la règle, de plus il nous faut considérer la difficulté de manipulation dans un espace 3D (alors que la manipulation à la souris reste 2D), nous avons donc décidé de conserver un espace 2D mais de proposer à l'utilisateur d'autres stratégies de visualisation que le placement dans un espace 3D et les choix de caméra conséquents. Nous en verrons un exemple dans la dernière partie de ce chapitre.

L'espace topologique peut cependant se révéler insuffisant pour l'activité de sélection. En effet, il convient lorsque l'utilisateur sélectionne selon un critère topologique ou sur la base de l'organisation du réseau qu'il a établi. Mais il peut également vouloir sélectionner, regrouper des modules sur des critères d'une autre nature, spatiale par exemple, en se basant sur les positions initiales, voire même spatio-temporels en se basant sur l'état des modules au $n^{ième}$ pas de la simulation. Il faut donc proposer un espace représentant les positions initiales des <MAT> et un espace représentant la simulation du réseau, et permettre la sélection depuis ces espaces. La représentation graphique des <LIA> dans ces deux espaces doit être à même de fournir les informations pertinentes (paramètres spatiaux, variables d'état). Cependant ces représentations, ainsi que la pertinence de ces informations, posent des questions, notamment par rapport à la simulation, que nous n'avons pas abordées durant nos travaux.

L'activité 3 (perception et manipulation des conditions initiales) requiert clairement un **espace 3D**, malgré les difficultés de manipulation que cela entraine. Nous reviendrons plus en détail dans le chapitre suivant, consacré au problème de la géométrie dans MIMESIS. Cependant, même pour cette tâche géométrique, l'espace topologique reste utile pour sélectionner les modules et faire ainsi la correspondance entre leurs propriétés topologiques et leurs propriétés géométriques.

Concernant l'activité de sélection, un autre espace graphique est intéressant, il ne représente pas le réseau mais le système de labellisation. C'est donc une représentation arborescente de ce système, qui permet donc la sélection d'une liste de module via la sélection d'une liste de labels, la sélection d'un conteneur permettant la sélection de tous les labels contenus. Cet arbre des labels existait déjà dans MIMESIS 4, son usage était très fréquent, ce qui valide sa réintroduction dans MIMESIS V. Cependant cette représentation doit signifier clairement à l'utilisateur ses propriétés, en premier lieu la nature du label en tant que référence. Cette distinction est importante à un autre titre, car l'arbre des labels présente une deuxième fonction après la sélection de modules, c'est la gestion du système des labels : création, suppression et édition de labels. C'est pourquoi l'arbre des labels présente deux modes de sélection : celui des labels et celui des modules désignés par les labels. Nous mettons ainsi l'accent sur la nature des labels comme étant des références uniquement, et permettons également des actions graphiques sur les labels distinctes des opérations sur les modules. La sélection sur l'arbre doit être accompagnée de l'expression de sélection de labels correspondante,

d'abord dans un objectif de confirmation, car observer la sélection via deux modalités améliore en effet la confiance envers le résultat. Il y a aussi un objectif pédagogique, visant à faire intégrer à l'utilisateur la relation entre la représentation arborescente du système de labellisation et les labels en tant que chaînes de caractères.

Nous mettons donc à disposition de l'utilisateur quatre espaces graphiques différents : l'espace topologique, l'espace 3D, l'espace de simulation et l'arbre des label. Chacun répond à des besoins particuliers. L'utilisateur pourra donc passer d'un espace à un autre selon la tâche qu'il a à accomplir.

2.6. Des outils de manipulation du réseau

Nous avons identifié dans la partie précédente deux principes intéressants de manipulation de la topologie du réseau, la fusion et la duplication. Nous proposons ici de les incorporer dans MIMESIS V par l'intermédiaire de PNSL. Pour réussir cette incorporation, il est indispensable de gérer au mieux le système de labellisation. En effet, ces manipulations ajoutent des modules au réseau, ces modules doivent donc être associés à des labels qui sont dans la logique de la manipulation, tout en étant maîtrisés par l'utilisateur.

2.6.1. Duplication

L'examen de l'opération de duplication dans MIMESIS 4 a montré comment l'utilisateur pouvait créer implicitement des noms. En effet, cette opération ne duplique pas seulement les modules contenus dans un label (label au sens de conteneur de modules) mais également tout le graphe issu du label. Si l'utilisateur ne contrôlait pas complètement ces créations de noms, il savait en tirer parti, elles lui étaient même indispensables pour connecter le sous-réseau créé au reste du réseau. Il nous faut donc expliciter la création de noms tout en tirant parti des labels déjà existants. Autrement dit, l'utilisateur a créé une organisation complexe des modules (à travers plusieurs opérations de labellisation), il souhaite dupliquer ces modules et communiquer cette organisation aux duplicatas des modules, c'est à dire dupliquer également les labels mais en les modifiant légèrement pour les identifier au duplicata. Dans MIMESIS 4, la modification concerne le préfixe, car on duplique en fait un conteneur, ce qui a pour conséquence de dupliquer les modules et les noms issus de ce conteneur. Cette fusion de deux opérations (dupliquer des modules et dupliquer des noms) rend l'opération peu souple : les modules dupliqués doivent être dans un même conteneur, les noms sont forcément basés sur le nom de ce conteneur. Dans MIMESIS V, la notion de conteneur n'existe plus, mais son rôle dans la labellisation des modules est repris par la notion de préfixe, directement rattaché aux noms plutôt qu'à la structure qui les sous-tend. Nous avons voulu conserver l'efficacité de la procédure de duplication mais en séparant la duplication des modules de la duplication des labels. La procédure PNSL prend donc trois arguments : 1) l'expression de sélection de labels désignant les modules à dupliquer 2) une chaîne de caractères désignant le préfixe commun à tous les labels que l'on souhaite dupliquer 3) une chaîne de caractères désignant le préfixe cible de tous les labels dupliqués.

```
Dupliquer ESL_de_modules avec préfixe_label vers nouveau_préfixe
```

Ce premier outil montre comment la manipulation du réseau et la manipulation du système de labellisation sont intriquées, et les difficultés qui en découlent. Si on sépare complètement les deux activités en proposant deux procédures distinctes, l'une d'elles devient difficile à accomplir, parce que l'association entre le réseau et le système de labellisation est alors perdue.

2.6.2. Fusion

L'opération de fusion consiste à fusionner deux modules <MAT> en un seul. Pratiquement, le deuxième module <MAT> est supprimé et toutes les interactions qui lui étaient connectées sont redirigées vers le premier module <MAT>. Ici encore, on ne veut pas perdre les informations de labellisation relatives au deuxième <MAT>. Intuitivement, l'opération de fusion suggère que tous les labels relatifs au deuxième <MAT> soient redirigés vers le premier <MAT>. Nous ne pouvons faire l'économie de cet effet de bord si nous voulons que l'opération de fusion soit symétrique, c'est à dire qu'interchanger les deux <MAT> n'ait aucun impact sur l'état du réseau et du système de labellisation. La signature de la procédure fusionner est donc :

```
Fusionner ESL_1erMAT ESL_2emeMAT
```

La procédure renvoie une erreur et ne s'exécute pas si les deux labels ne pointent pas vers des <MAT> de même type, toujours dans un souci de garantir la symétrie de l'opération.

L'extension de l'opération de fusion de deux <MAT> à la fusion de deux sous-réseaux est intéressante mais délicate. Les triplets de briques hexagonales de [LG97] (cf. §1.4 figure 47 à droite), peuvent être vus comme trois briques assemblées deux à deux par la fusion de deux "faces", c'est à dire deux sous-réseaux. Premier point, on suppose que les deux sous-réseaux sont identiques du point de vue topologique. Deuxième point, les deux sous-réseaux sont bien formés, c'est à dire qu'aucun des <LIA> n'a de connexions pendantes. En effet, dans le cas contraire la fusion de deux <LIA> pose problème si les points L connectés à l'extérieur ne sont pas connectés au même <MAT> : l'opération n'est plus symétrique, il faut choisir à quel <MAT> se connecte le <LIA> fusionné. Troisième point, la redirection des labels est toujours indispensable pour garantir la symétrie de l'opération. Il faut donc que les deux sous-réseaux soient spécifiés par des listes de labels également symétriques, c'est à dire qu'elles décrivent les sous-réseaux dans le même ordre. Cette dernière condition rend la vérification de l'identité topologique plus aisée, en effet il suffit de vérifier que les deux listes forment une liste de paire de modules de même type, et dont les <LIA> sont connectés aux <MAT> de même indice dans la liste. Ce détail d'implantation n'est pas uniquement un point technique, il permet de préciser ce que signifie l'identité topologique : les sous-réseaux sont identiques sans considérations par rapport à leurs connexions à l'extérieur. Ainsi les sous-réseaux A et B présentés en figure 52 sont identiques et peuvent être fusionnés pour obtenir le sous-réseau C présenté en figure 53.

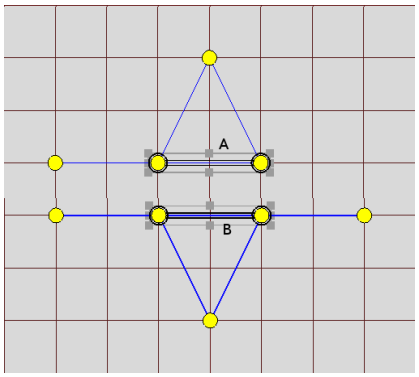


Figure 52 Les sous-réseaux A et B avant fusion

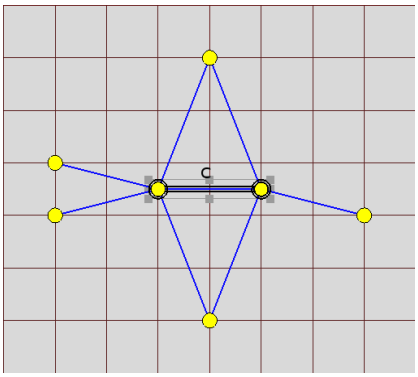


Figure 53 Le sous-réseau C résultant de la fusion

En conclusion nous proposons donc une procédure de fusion de deux sous-réseaux, duale de la procédure de duplication : on réduit le réseau au lieu de l'étendre. Son fonctionnement est simple. Il donne à l'utilisateur la responsabilité de spécifier correctement les deux sous-réseaux à fusionner. L'adverbe "correctement" a ici une double signification : les expressions de sélection de modules doivent être symétriques et désigner deux sous-réseaux topologiquement identiques, mais elle doivent également correspondre à l'intention du modélisateur. Nous verrons dans la partie consacrée aux exemples comment cette tâche de fusion peut être assistée par l'espace topologique. Les exemples seront également l'occasion de montrer comment l'association d'opérations de duplication et de fusion permet de définir et de réaliser des opérations de symétrisation et de translation topologique.

2.7. Des outils de connexion évolués : des labels pour les points de communication

La construction de modèles de grande taille met en jeu des procédures de connexion complexes avec un très grand nombre de points de communication. Pour mettre en œuvre ces connexions, deux méthodes sont possibles : la méthode basée module, qui propose de connecter un <LIA> à deux <MAT>, et la méthode basée points de communication, qui propose de connecter un point L à un point M. Dans un script PNSL, une même interconnexion donnera lieu à deux blocs de code assez différents selon la méthode choisie. Il n'y a aucune raison que l'utilisateur ne puisse pas choisir la méthode qui convient le mieux à son activité.

La méthode basée module se traduit par une procédure proche de l'esprit de la procédure graphique de connexion :

```
Connecter_LIA label_LIA label_MAT1 label_MAT2
```

Pour mettre en œuvre la méthode de connexion basée point de communication, il faut un nouveau type d'expression au sein de PNSL pour désigner un point de communication, et plus généralement une liste de points de communication. En effet, cette méthode se prête bien à des connexions par lot, ce qui est particulièrement appréciable pour les

grands modèles, pour lesquels une seule instruction pourra suffire pour effectuer des milliers, voire des millions, de connexions. D'autre part, cette expression de désignation de liste de points de communication doit pouvoir permettre une expression efficace dans le cas des macro-modules. Rappelons qu'un macro-module intègre N modules élémentaires, <MAT> ou <LIA>, de même type et de mêmes paramètres (cf. Chapitre I §3.3). Ceux-ci sont courants dans les grands modèles, car ils permettent de concentrer un grand nombre de modules de même type en un seul. Durant la modélisation, l'utilisateur peut estimer nécessaire et pertinent pour son projet d'intégrer un certain nombre de modules dans un seul. L'utilisateur a alors besoin de désigner de manière efficace des ensembles de points M et L, ensembles qui peuvent être de grande taille. Nous appelons ces expressions ESP pour Expression de Sélection de Points. Elles se présentent sous la forme suivante :

```
ESL_modules opérateur_distributivité liste_tcl_d_indices
```

L'opérateur de distributivité détermine si la liste de modules doit être décomposée avant la liste d'indices ou l'inverse. Exemple :

chose1..3 -> 1..4 donne :

- au premier niveau : (chose1 , 1..4) (chose2 , 1..4) (chose3 , 1..4)
- au deuxième niveau : (chose1, 1) (chose1, 2), ... (chose2 , 1) (chose2, 2) ,etc...

Mais chose1..3 -< 1..4 donne :

- au premier niveau : (chose1..3, 1) (chose1..3, 2) (chose1..3, 3) (chose1..3, 4)
- au deuxième niveau : (chose1 , 1) (chose2 , 1) (chose3 , 1) (chose1 , 2) etc ...

Il est ainsi possible de sélectionner de manière ordonnée un grand nombre de points de communications sur un ou plusieurs modules. Ces expressions demeurent cependant plus numériques qu'alphabétiques, et il sera souvent nécessaire d'écrire de petits algorithmes pour créer les listes d'indices. Il est intéressant ici de reprendre la notion de label. Ainsi, pour objectiver ces listes nous introduisons une notion de « label de liste de points de communication (LPC) ». Dans un premier temps, un LPC est relatif à un module. Par exemple, pour un macro-module <N-REF>, présentant 2N points L, deux LPC sont définis par défaut, gauche et droite, regroupant respectivement les indices pairs et impairs. On utilisera donc l'ESP suivante pour désigner les points d'indice pair : `/monNREF -> gauche`. L'utilisateur peut également définir lui-même un LPC avec la procédure `définir_LPC` :

```
définir_LPC ESL_module liste_tcl_d_indices –nommer nom_LPC.
```

Nous pouvons à présent définir une procédure de connexion d'une liste de points L à une liste de points M : `connecter_ptL ESP_PtL ESP_PtM`. Une ESP peut référencer plusieurs fois le même point de communication, ce qui permet par exemple de connecter deux groupes de <MAT> avec un groupes d'interactions en deux instructions. Ce mode de connexion "liste à liste" reste une alternative au mode "module à module". Dans le premier, l'attention est mise sur l'écriture de la liste, à l'aide d'un algorithme ou bien d'une expression régulière. Dans le deuxième, l'attention est mise sur l'algorithme de connexion. L'intérêt du premier mode est d'objectiver la connexion, avec la possibilité de

créer deux LPC représentant la connexion. L'objectivation permettant de mémoriser une configuration de connexion, l'utilisateur peut donc créer plusieurs configurations et tester chacune d'elle au sein d'un même projet. Le second mode présente l'avantage d'être plus proche de la pensée physique (deux masses en interaction), là où le premier mode est plus abstrait, plus ancré dans le formalisme.

Nous avons introduit des expressions de sélection de points de communications, les ESP, et un moyen de les objectiver, les labels de liste de points de communication (LPC). L'utilisateur peut ainsi définir des connexions complexes, aussi bien avec le mode classique basé module qu'avec le mode avancé, basé point de communication, plus adapté à la conception de modèles de large taille.

2.8. Contrôle de paramètres dans MIMESIS V

Nous avons montré l'intérêt du contrôle de paramètres dans le chapitre II. Il nous faut maintenant montrer comment les différents modules que nous avons proposés s'intègrent dans MIMESIS V : création, manipulation, représentations graphiques. Cette présentation est axée sur PNSL, les représentations et manipulations graphiques n'étant pas encore finalisées, nous ne les aborderons que partiellement. Il s'agit donc principalement de définir des procédures et une extension du système d'adressage par chaîne de caractères, comparable à celle que nous venons de présenter avec les points de communication. En effet, le principal changement qu'apporte le contrôle de paramètres au formalisme CORDIS-ANIMA est l'introduction des points d'entrée et de sortie de paramètre. Nous définissons donc des expressions de sélection des points d'entrée et de sortie de paramètres, les ESEP pour Expression de Sélection de points d'Entrée de Paramètre, et les ESSP pour Expression de Sélection de points de Sortie de Paramètre. A l'image des ESP (Expression de Sélection de Points de Communication), les ESEP sont composées de deux parties : une expression de sélection de labels (ESL) et un paramètre. Elles sont en revanche moins complexes puisqu'il n'y pas de distributivité à gérer entre l'ESL et le paramètre. La chaine suivante est un exemple de ESEP : `/chose/interactionRessort1 -> pK`. Elle désigne l'entrée de paramètre pour la raideur de l'interaction désigné par le label `/chose/interactionRessort1`. En ce qui concerne les ESSP, les seuls modules présentant un point de sortie de paramètres sont les <MCP> et ils n'en présentent qu'un seul, une ESSP est donc réduite à une ESL de <MCP>. Nous pouvons donc définir une procédure de connexion d'un point de sortie de paramètres à un point d'entrée de paramètre :

```
Connecter_Param ESSP ESEP
```

On remarque que cette procédure permet également de connecter par liste plutôt qu'individuellement. De la même manière que nous avons proposé deux méthodes de connecter des interactions et des masses (méthode basée point de communication et méthode basée module), nous proposons ici une deuxième méthode pour connecter un module de contrôle de paramètre. Il faut alors prendre en considération d'une part le paramètre contrôlé et d'autre part le module <MAT> effectuant le contrôle. La procédure PNSL correspondant à cette méthode traduit la phrase "contrôler le paramètre *pa* du module *mo* avec la masse *ma* via le module de contrôle de paramètre *mc*", ce qui donne la signature suivante :

```
Contrôler_parametre id_param label_module label_masse1D label_mcp
```

Cette méthode est la plus adaptée pour une traduction graphique. Elle nous permet de définir des conditions de création graphique d'un <MCP> similaire aux conditions de créations d'un <LIA>. En revanche, on déroule la procédure à l'envers : le <MCP>, représenté sous forme d'une flèche (cf. figure 54) doit d'abord être connecté au <MAT> unidimensionnel, puis au module, ce qui déclenche l'apparition d'une liste des paramètres du module, et permet à l'utilisateur de choisir le paramètre contrôlé.

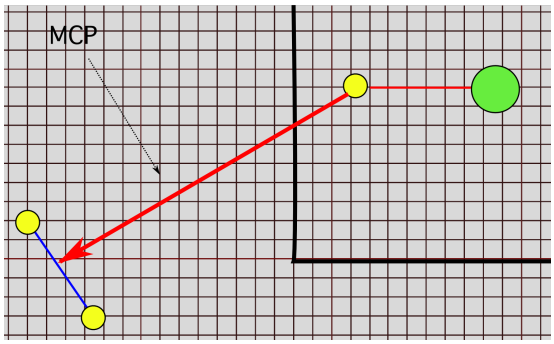


Figure 54 : Représentation graphique d'un <MCP> dans l'espace topologique

Les modules de mesure <MES> s'intègrent directement dans MIMESIS V à l'aide des outils de connexion de points L définis en 2.8. Il ne s'agit en effet que de connecter des points L à des points M. Les représentations et manipulations graphiques des modules de mesure n'ont pas été définies.

2.9. La collaboration entre modalité graphique et PNSL : une nécessité problématique

Certains des outils que nous avons présentés apparaissent exclusivement liés à la modalité script, comme les outils de duplication, de fusion, de filtrage, et les labels de liste de points de communication. Si la modalité script doit être une modalité centrale dans MIMESIS V, nous voulons qu'un maximum d'outils soit accessible via les deux modalités, car l'utilisateur doit pouvoir choisir sa modalité en fonction de sa tâche. En énonçant ce principe d'équivalence, nous commençons à poser le problème plus large de la collaboration entre les deux modalités. En effet, l'équivalence des deux modalités est déjà très problématique car leur nature même les oppose. Davantage que ce premier problème, la "collaboration graphique textuelle" est un enjeu pour la complexité, car ce sont la relation et la combinaison riches entre les deux modalités qui permettront au modélisateur de progresser en complexité.

Le modèle WIMP est insuffisant pour définir MIMESIS V. Pour composer les différents piliers présentés dans cette partie, il faut lui ajouter la collaboration graphique textuelle. Nous devons donc définir cette problématique, chercher à comprendre ce qu'elle signifie, ce qui est en jeu, tant au niveau de l'interface qu'au niveau du processus de modélisation. Ce travail demande une étude théorique préalable, pour interroger les concepts de modalité graphique et modalité script, déterminer leurs spécificités, les

situer par rapport au problème plus général de l'interaction Humain-Machine. Nous pourrons alors définir la collaboration graphique textuelle dans le contexte plus particulier de la modélisation de réseaux masses-interactions pour le mouvement visuel.

3. Cadre d'étude pour la collaboration graphique textuelle

Notre but dans ce chapitre est d'essayer d'établir un cadre d'étude permettant de décrire les interfaces faisant intervenir la collaboration graphique textuelle. Nous devons donc définir avec précision ce que nous entendons par "collaboration graphique textuelle". Il s'agit de faire coexister dans une interface deux modalités : la modalité graphique, qui est une manipulation basée sur des objets graphiques, et la manipulation langagière, c'est à dire une interaction par le biais d'un langage de script. Le cadre que nous allons décrire permettra de préciser ces deux définitions, de les situer par rapport aux différentes définitions d'une modalité. Nous nous attacherons également à préciser ce que nous entendons par "coexister" et "collaborer". Nous avons choisi comme point de départ d'étudier le problème plus large de la multimodalité, largement abordé dans la littérature Interaction Humain-Machine (IHM). Nous aborderons ensuite le thème des métaphores d'interaction avec l'ordinateur avec les quatre métaphores de Hutchins. Cet état de l'Art nous permettra notamment d'introduire la question de l'évaluation d'une interface que nous tenterons de confronter à notre conception de ce qu'est un "bon" outil de création.

3.1. Les cadres d'étude pour la multimodalité

La définition d'une modalité varie selon le point de vue adopté : du point de vue utilisateur la définition la plus largement admise est celle de technique d'interaction, du point de vue système une modalité est un processus analysant et produisant des fragments d'information [Mar98]. Ces deux définitions restent très générales, les différentes contributions que nous présenterons permettront de préciser ces définitions. Comme le soulignent Nigay et Coutaz dans [NC96], « l'important est de choisir la perspective qui convient aux objectifs et de s'y tenir ».

Une modalité est à la fois une représentation, c'est le point de vue de Bernsen sur les modalités de sorties [Ber94], et une technique d'interaction. La technique d'interaction fait intervenir un périphérique en entrée, mais elle s'appuie sur l'aspect représentationnel de la modalité, ainsi Nigay et Coutaz définissent une modalité comme un couple (périphérique, langage d'interaction). Nous avons étudié cinq cadres d'étude pour la multimodalité, nous en présentons ici un bref résumé. Le premier cadre est celui de Bernsen [Ber94], qui fonde une taxonomie pour les modalités de sortie. Celle ci est basée sur 4 propriétés booléennes et orthogonales :

1. Statique/dynamique
2. Linguistique/non linguistique
3. Analogue/non analogue
4. Arbitraire/non arbitraire

À ces 4 propriétés s'ajoute le type de media de représentation :

- Graphique
- Acoustique
- Tactile

Bernsen définit ainsi 48 modalités "pures". La modalité graphique ne peut être ainsi classée, car ce n'est pas une modalité pure, elle mêle notamment des éléments linguistiques et des éléments non linguistiques. Ce cadre ne porte pas sur ce qui fait, pour nous, la spécificité de la modalité graphique, c'est à dire le travail sur l'objet et une manipulation orienté vers l'action.

Les autres cadres étudiés nous permettent de préciser la signification de la collaboration. Laurence Nigay et Joëlle Coutaz proposent dans [NC93] un espace de conception pour les systèmes multimodaux, elles insistent sur deux caractéristiques cruciales de ces systèmes : le traitement concurrent et la fusion des données. Ce travail nous permet d'abord de nous distinguer d'un cas type de multimodalité, la combinaison synergétique (fusion et traitement parallèle), dont l'exemple typique est la désignation à la souris d'un objet et de l'énonciation en langage naturel d'une action impliquant cet objet.

Les propriétés CARE (Complémentarité, Assignation, Redondance et Equivalence), définies dans [CN94] et [CNS+95] par les mêmes auteurs nous permettent de situer plus précisément notre recherche. Nous avons déjà exprimé notre recherche de l'équivalence pour un maximum de tâches, et donc la volonté de réduire l'assignation d'une tâche à une modalité. La complémentarité est recherchée sur des tâches beaucoup plus larges, comme par exemple la construction de la topologie du réseau. Or, le formalisme proposé est basé sur une expression précise des tâches, qui permet au système de mettre en œuvre la propriété spécifiée. Notre perspective est très différente, car c'est l'utilisateur et non le système qui opère cette combinaison. La formalisation ne détaille pas *comment* deux modalités peuvent être combinées de manière complémentaire. La propriété de redondance stipule que deux modalités peuvent être utilisées pour réaliser la même tâche, parallèlement ou séquentiellement.

Les cinq primitives de TYCOON, pour TYpes et buts de COOpératioN entres modalités, définies par Martin dans [Mar98], reprennent les propriétés CARE[4]. Les quatre premières primitives, complémentarité, spécialisation, redondance et équivalence, correspondent aux quatre propriétés CARE. La définition change légèrement car une modalité est définie comme un processus recevant et produisant des fragments d'information. Vient s'y ajouter une cinquième primitive, le transfert. Elle qualifie un fragment d'information traité via une modalité et qui est ensuite traité par une autre modalité. Nous reviendrons sur ces primitives dans la partie suivante, notamment le transfert et la redondance. Cette dernière est définie comme le fait que le même fragment d'information est traité par

[4] Les propriétés CARE étant basées sur un article de Martin datant de 1993

deux modalités. Cette définition, et la définition de la modalité, permettent d'interpréter plus largement les primitives que les propriétés CARE. Notamment, on ne se réfère pas une tâche particulière et le traitement d'un fragment d'information peut être vu aussi bien comme sa représentation, que comme sa création ou son édition.

Vernay et Nigay proposent un cadre d'étude dédié à la combinaison et la caractérisation des modalités de sortie [VN00]. L'espace est organisé selon deux axes. Le premier considère les aspects combinés : temps, espace, langage d'interaction et sémantique. Le second représente le schéma de combinaison, sur le modèle des schémas d'Allen [All83], initialement pensé pour des combinaisons temporelles (cf. figure 55). Nous identifions donc la combinaison recherchée comme spatialement séparée, temporellement séquentielle, syntactiquement différente (le langage de la modalité graphique est différent de PNSL) et sémantiquement complémentaire. Ici encore, nous exprimons notre problème, mais le cadre ne nous donne pas d'indices pour le comment. Nous ne sommes pas dans une situation où, comme les auteurs précités, nous pourrions définir des critères ergonomiques nous permettant de déduire des règles de conception.

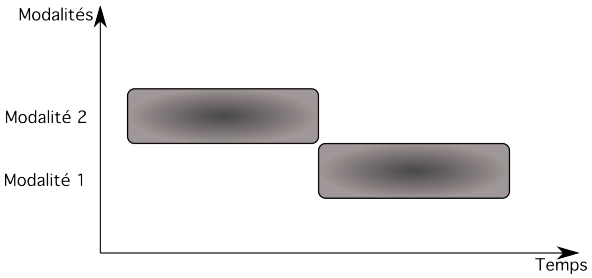


Figure 55 : Schéma pour une combinaison de modalité avec un point de contact. Exemple avec l'aspect Temps.

Cette brève étude de la multimodalité nous montre deux choses. Premièrement, les différentes définitions d'une modalité ne sont pas adaptées à notre problème. En effet nous parlons de modalité graphique comme un concept très général, touchant aussi bien la manipulation directe d'objets graphiques, telle que définie au §2.5, que l'édition par tableau ou boîte de dialogue. A l'inverse, la modalité textuelle est définie de manière très précise, prenant en compte le type de langage (script donc impératif), élément absent de tous les cadres que nous avons étudiés. En second lieu, les propriétés CARE et les primitives TYCOON constituent des clefs d'analyse de notre problème que nous tenterons de mettre en évidence dans la partie suivante, consacrée à la définition de nos objectifs dans le cadre de MIMESIS V.

3.2. Métaphores d'interaction avec l'ordinateur

Qu'est ce qu'un ordinateur pour l'utilisateur ? Dans les premières décennies de l'informatique, c'est à dire jusqu'à 1970, l'acception de l'ordinateur comme outil était courante, et d'ailleurs réduite à sa fonction de calculateur. Dès les débuts de l'interaction homme-machine, cette conception apparaît trop limitée, Alan Kay voit dans l'ordinateur un medium sur lequel on peut lire et écrire en exploitant les 3 modes d'apprentissage

(énactif, iconique et symbolique) [Kay90], conception résumée dans le slogan « Doing with Images make Symbols ». Brenda Laurel favorise une vision anthropomorphique de l'ordinateur [Lau90], arguant que c'est avec d'autres êtres humains que nous communiquons le mieux. En 1984, Shneidermann théorise les interfaces à manipulation directe [Shn84], la même année sort le premier Macintosh, qui popularise ce style d'interface, aujourd'hui très largement dominant. La manipulation directe est définie par trois principes :

1. Représentation constante de l'objet d'intérêt (c'est à dire l'ensemble des données que l'utilisateur veut manipuler)
2. Actions physiques directes sur l'objet d'intérêt, au lieu d'une syntaxe complexe
3. Opérations rapides, incrémentales, réversibles, dont l'impact sur l'objet d'intérêt est immédiatement visible

Hutchins synthétise ces différentes positions en proposant quatre métaphores d'interaction avec l'ordinateur [Hut88]. La métaphore conversationnelle fait intervenir un agent intermédiaire entre l'utilisateur et l'objet d'intérêt : l'utilisateur tient une discussion avec l'agent, qui impacte cette discussion sur l'objet d'intérêt. La métaphore déclarative est aussi une interaction langagière, mais se distingue de la métaphore conversationnelle par son style déclaratif, et non impératif. Ce changement de style a un impact important, car l'utilisateur décrit l'objet plutôt que son processus de construction. La métaphore "model-world" repose sur deux conditions : 1) les expressions de l'interface, ce qu'elle présente, apparaissent comme des actions avec une force causale sur l'objet d'intérêt ; 2) la génération de ces expressions est contrainte de telle manière à ce qu'il ne soit pas possible de composer une expression qui ne peut pas être réalisé sur l'objet d'intérêt. La métaphore collaborative s'inspire de la collaboration entre plusieurs sujets humains. Ici l'utilisateur collabore avec des agents dans une métaphore model-world. Hutchins parle aussi de "mode mixte d'interaction" qui réunit la métaphore conversationnelle et la métaphore model world. Idéalement cette métaphore suggère (traduction libre) : "L'utilisateur doit pouvoir avoir une conversation sur le monde avec l'agent, et l'agent comme l'utilisateur doivent pouvoir manipuler le monde partagé"[5]. La figure 56 résume cette métaphore.

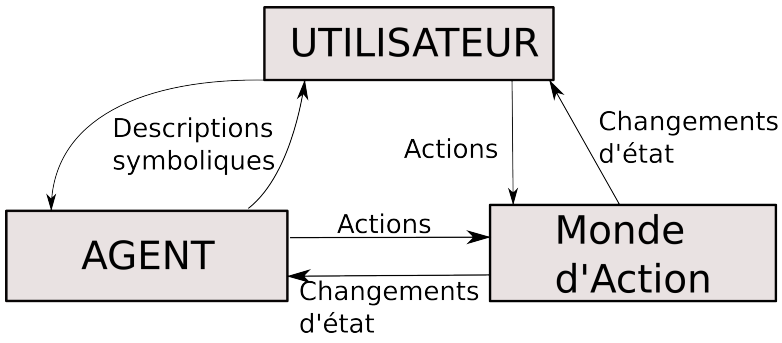


Figure 56 : Schéma général de la métaphore collaborative (d'après [Hut88])

[5] "The user should be able to have a conversation about the world with the agent, and both the user and the agent should be able to manipulate the shared world"

Ces quatre métaphores nous permettent de situer notre problème de manière plus adaptée que la notion de modalité. Ainsi, la métaphore conversationnelle correspond à la modalité script, un script constituant un ensemble d'instructions que va réaliser l'agent. Parallèlement, la métaphore model world correspond à la modalité graphique, centrée sur l'action et sur la représentation graphique de l'objet d'intérêt. Il apparaît donc logique d'associer notre problématique à celle de la métaphore collaborative. Si la citation de Hutchins évoque davantage l'équivalence entre les deux métaphores, on entrevoit toutes les possibilités de combinaison qu'ouvre la métaphore collaborative. Il est intéressant de noter que Hutchins voit d'abord l'agent comme un assistant intelligent, capable d'observer les actions de l'utilisateur et d'en déduire les actions adéquates. Nous sommes dans une perspective assez différente, dans laquelle l'agent n'est qu'un interpréteur du langage du script. Néanmoins, cette métaphore nous paraît toujours valable, d'autant plus qu'elle pose le problème de la collaboration entre deux modes d'interaction très différents, l'un basé sur l'objet et représentant son état et proposant à l'utilisateur d'effectuer directement des actions dessus, l'autre basé sur l'action et représentant le processus de construction de l'objet d'intérêt.

3.3. Evaluation d'une interface

Pour évaluer et comparer les différentes métaphores, Hutchins et al. [HHN85] s'appuient sur la théorie de l'action de Norman : celle-ci oppose les buts de l'utilisateur au système physique (l'ordinateur et ses périphériques), séparés par la chaîne d'exécution et la chaîne d'évaluation. Ces chaînes constituent une distance entre les buts de l'utilisateur et leur réalisation, distance que le concepteur d'interface cherche à minimiser. Cette distance peut être séparée en deux composantes (cf. figure 57) :

- Distance sémantique : dans quelle mesure le langage de l'interface permet à l'utilisateur d'exprimer son problème ? Cette expression est-elle courte ou longue ?
- Distance référentielle : l'expression que l'utilisateur doit formuler est-elle compliquée, requiert-elle un apprentissage préalable ?

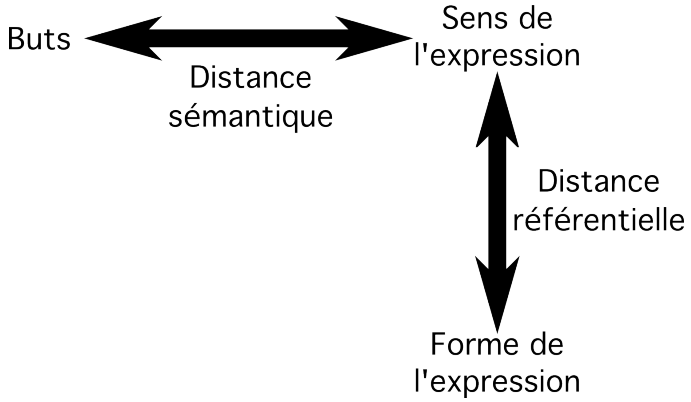


Figure 57 : Distance sémantique et distance référentielle (traduit de [HHN85])

En plus de cette notion de distance, Hutchins et al. , introduisent le critère d'*engagement*. Cette notion est plus difficile à définir formellement. Elle réside dans l'idée "qu'un "bon" engagement donne l'impression d'avoir le contrôle des objets, c'est à dire non pas du programme ou de l'ordinateur, mais des objets sémantiques de nos buts et intentions" (traduction libre). La métaphore conversationnelle est généralement considérée faible par rapport au critère d'engagement. Cette faiblesse est souvent niée par les informaticiens, qui jugent plus fort leur engagement de l'interaction en ligne de commande. Ce renversement n'est pas l'apanage des informaticiens, nous avons constaté que les étudiants en école d'art, après un certain temps d'adaptation, témoignaient également d'un engagement important avec la modalité script.

Hutchins insiste sur le fait que la manipulation directe n'optimise pas ces critères dans tous les cas. En effet toutes les tâches qui pourraient tirer avantage d'un haut degré d'abstraction auront alors une distance sémantique bien plus importante dans une métaphore model world que dans une métaphore conversationnelle. Le choix d'une métaphore dépend donc fortement de la tâche à accomplir, il ne s'agit plus alors de décider *si* la manipulation directe est une meilleure interaction que la métaphore conversationnelle, mais plutôt *quand* et *comment* [Fro96]. Cette dernière assertion nous renforce dans le choix de la collaboration graphique textuelle, l'utilisateur pouvant choisir la modalité *quand* il la juge plus adaptée. Il reste maintenant à détailler le *comment*.

Il existe bien entendu de nombreux autres outils d'évaluation d'une interface, comme la famille GOMS [JK96] ou Keystroke[CMN80]. Cependant ces outils ont l'objectif d'effectuer des mesures très fines par rapport à des tâches bien déterminées. Notre objectif est ici assez différent. D'abord, nous nous intéressons à un outil de création, nous ne cherchons pas à optimiser la performance sur une tâche précise mais plutôt l'expressivité. Les critères d'évaluation nous servent à illustrer les différences entre manipulation directe et manipulation par script, pour montrer en quoi leur combinaison peut être synonyme de puissance d'expression. Dans cet objectif, les distances référentielles et sémantique sont suffisantes, elles permettent, comme le rappelle Frohlich dans [Fro96], d'insister sur l'avantage du script sur la manipulation directe pour les tâches très répétitives, pour lesquelles l'utilisateur est limité justement par le caractère direct de l'interaction. La distance sémantique est alors très importante, ce qui sanctionne la manipulation directe, même avec une distance référentielle faible.

3.4. Conclusion

L'étude des différents cadres pour la multimodalité, ainsi que les métaphores d'interaction, ont montré que le problème de la collaboration graphique/textuel n'avait jamais été abordé avec notre objectif. En tant que multimodalité, les conditions et les implications de la combinaison des deux modalités ne sont jamais traitées. Du point de vue de la métaphore collaborative, son utilité est affirmée mais elle peut revêtir des formes très diverses, et ici encore la forme qui nous intéresse n'est pas abordée.

Les conditions, enjeux et objectifs de la collaboration entre modalité graphique et modalité script restent donc à préciser, dans le cadre bien particulier de la modélisation physique. Nous avons déjà évoqué la simple possibilité de concevoir une partie du réseau par script et une autre partie graphiquement, ce qui correspond à la primitive de

complémentarité dans [Mar99]. Notre ambition va au-delà, cette collaboration doit être un outil supplémentaire pour l'accès à la complexité et l'augmentation des possibilités créatives de MIMESIS V. L'utilisateur doit pouvoir maitriser la portée de chacune des modalités et les relations qu'elles entretiennent.

4. Conditions et enjeux de la collaboration entre modalité graphique et modalité script

4.1. Champ de la modalité graphique dans l'interface MIMESIS

La modalité graphique est un concept difficile à définir comme nous l'avons vu dans la partie précédente. Le terme modalité déjà est lui même polysémique, selon qu'on traite d'une modalité comme représentation [Ber94], ou comme "un type de canal de communication utilisé pour transporter ou acquérir de l'information " [NC94]. Ensuite le terme graphique est lui aussi ambigu, car on peut aussi bien parler du canal sensoriel visuel que de représentations graphiques non alphanumériques. Pour notre sujet, nous statuons donc temporairement sur ce problème en définissant le champ de la modalité graphique par la négative : "Tout ce qui n'est pas la modalité script". La différence fondamentale entre ces deux modalités, en dehors du type de représentation (graphique vs textuelle) est que la modalité graphique permet de travailler uniquement sur l'objet d'intérêt, c'est le principe même de la manipulation directe de Shneiderman, elle ne permet pas de travailler sur son processus de construction.

La partie graphique de MIMESIS V est basée sur le modèle WIMP (Window Icon Menu Pointer) et consiste en trois types de tâches :

- l'utilisateur sélectionne une sous-partie de l'objet d'intérêt (un groupe de modules, des labels) dans un espace graphique et opère une action sur cette sous-partie (suppression, connexion, paramétrage physique, changement des conditions initiales) soit à travers une action graphique (glisser/déplacer), soit en choisissant une action dans un menu ou parmi les icônes disponibles, ou encore en fixant une valeur numérique dans une boîte de dialogue.
- Choix d'un outil dans une palette, travail avec cet outil dans un espace graphique (espace topologique, espace 3D, espace simulation). L'outil le plus commun est l'outil de création de module, la palette proposant plusieurs types de modules.
- Travail sur un tableau.

Les actions possibles dans l'espace graphique ne visent pas toutes à modifier l'objet d'intérêt, nombreuses sont celles qui portent sur les différents outils autour de l'objet d'intérêt :

- Le système de labellisation : la vue arborescente de celui-ci permet de modifier directement des labels.
- L'espace topologique : le placement des modules sur cet espace et le choix du point de vue sur cet espace.
- ...

Le champ de la modalité graphique est large mais il est caractérisé par l'action directe sur l'objet d'intérêt et sa représentation continue. On retrouve ici deux caractéristiques constitutives d'une interface à manipulation directe telle que définie par Shneiderman [Shn84]. La largeur de ce champ est une difficulté de taille pour la collaboration, car chaque élément de la modalité graphique (tableaux, boîtes de dialogue, espaces graphiques) doit être pris en compte.

4.2. Travailler sur le processus comme si on travaillait sur l'état

Comme nous venons de le rappeler, la modalité graphique permet de travailler sur l'état : on agit directement sur l'état actuel de l'objet d'intérêt. A l'inverse, la modalité script permet de travailler sur le processus de construction des objets d'intérêts de MIMESIS V. MIMESIS 4 proposait la modalité script comme unique modalité d'action sur la topologie du réseau CORDIS-ANIMA, ce qui permettait à la fois une puissance d'expression importante mais aussi un certain recul par rapport la construction de la topologie, impossible si seule une représentation de l'état était disponible. Cette monomodalité est certes restrictive mais elle donne à l'utilisateur la possibilité de travailler sur le processus comme s'il travaillait sur l'état, il peut "sculpter", modifier à sa guise l'unique script qui décrit la construction de la topologie du réseau. L'utilisation de MIMESIS 4 en situation de création a montré l'intérêt de cette posture, les possibilités qu'elle procure au modélisateur, qui apprécie, après certes un temps d'adaptation, la complexité que lui permet d'exprimer le langage de script. MIMESIS V se doit de proposer un travail sur le processus au moins comparable combiné à une modalité graphique permettant d'effectuer toutes les tâches de base, y compris l'édition de la topologie. La représentation du processus de construction, le ou les scripts, doit donc coexister harmonieusement avec les représentations graphiques de l'état du réseau et des autres objets d'intérêt.

Par rapport aux propriétés CARE, l'équivalence est ici réaffirmée mais avec une condition supplémentaire, relative à la cohérence de l'ensemble. On remarque notamment que la redondance est probable et que le système doit alors assurer que les deux représentations, de l'état et du processus, sont compatibles.

Concrètement, une telle proposition implique que le ou les scripts sont liés aux objets qu'ils créent, aux propriétés qu'ils définissent ou modifient, sinon l'interface n'a aucun moyen d'assurer la compatibilité entre les deux représentations (du processus et de l'état). Cette relation entre scripts et objets d'intérêt est inédite au regard des interfaces mixant les modalités script et graphique. En effet, ces interfaces sont basées sur la notion de macro, c'est à dire un script qui s'exécute sans conserver de lien avec ce qu'il a généré, il n'a pas valeur de représentation. C'est le cas notamment dans le duo

Simulink/Matlab [Sim] et dans Maya [May] avec le langage de script MEL. Notons que certaines interfaces offrent ce lien, de manière plus ou moins prolongée, mais le script n'est accessible qu'à partir d'un ensemble prédéterminé de paramètres via une boîte de dialogue. Le logiciel de modélisation géométrique procédurale Groboto [Gro] et ses objets AutoBots fonctionnent sur ce modèle, de même que la collection de scripts du logiciel de dessin vectoriel Inkscape [Ink] en mode prévisualisation.

4.3. Un objectif simple : la collaboration horizontale.

L'un des objectifs de la collaboration provient du constat d'hétérogénéité des réseaux CORDIS. Plus précisément, certains modèles déjà existants peuvent être vus comme la composition d'un réseau de grande taille exhibant une certaine régularité topologique et d'un réseau de petite taille montrant peu de symétries de répétitions. Il apparaît immédiatement que la modalité script est adaptée à la conception du premier réseau et la modalité graphique à la conception du deuxième. Un troisième élément apparaît ensuite dans le processus, un groupe d'interactions connectées aux deux premiers réseaux. Ici encore, la taille de ce groupe ainsi que la complexité des connexions pourront amener le modélisateur à préférer l'une ou l'autre modalité. Il doit donc être possible d'opérer graphiquement une connexion sur un élément créé textuellement, ce qui en suppose une représentation graphique des points de communication de cet élément. Symétriquement, il doit être possible d'opérer textuellement une connexion sur un élément créé graphiquement, ce qui suppose d'en connaître un label.

Plus largement cet objectif aboutit à un premier critère d'interopérabilité : "Tous les éléments créés avec la modalité graphique doivent être accessibles avec la modalité script, et vice versa". Le terme "accessible" signifie que l'on peut agir sur ces éléments et les référencer : en connecter un point M, en connecter un point L "pendant" (non connecté), le dupliquer. Cette interopérabilité ne s'applique pas qu'aux éléments du réseau, mais aussi à tous ses outils périphériques et en premier lieu les labels, qui doivent pouvoir être utilisés par les deux modalités.

4.4. Un objectif complexe : la collaboration verticale

Cette première interopérabilité donne lieu à de nouveaux types de processus de modélisation, que nous détaillerons ultérieurement. Cependant, nous pouvons pousser plus loin l'interopérabilité avec un second critère : "Tous les éléments créés avec la modalité graphique doivent être éditables avec la modalité script, et vice et versa". Le terme "éditable" va donc plus loin que la simple accessibilité décrite plus haut, les éléments peuvent être supprimés, leurs points L déconnectés; les labels peuvent être supprimés, renommés : une modalité A peut donc modifier un état défini ou généré par une modalité B. Il faut considérer cette interopérabilité étendue avec l'impératif du "travail sur le processus comme sur l'état". En effet, si un script a produit un certain résultat (créé certains modules, effectué certaines connexions) et que l'utilisateur vient

éditer ce résultat, alors le script n'est plus représentatif de ce résultat. L'utilisateur ne peut donc plus travailler sur ce script, car il n'est plus représentatif non seulement du résultat mais surtout de son processus. Le script doit donc être augmenté de la suite de ce processus : il faut transformer les actions graphiques en instructions PNSL. Ainsi énoncée, cette opération ne semble pas poser de problème particulier, en effet, toutes les actions possibles dans l'espace graphique ont leur équivalent en PNSL. Mais revenons à l'un des éléments centraux de la modalité graphique, la tâche de sélection. Nous avons vu que celle-ci faisait sens par rapport à l'organisation de l'espace topologique, ou bien par rapport aux conditions initiales si la sélection s'est faite dans l'espace géométrique. Ce sens doit être conservé à la transformation de cette sélection graphique en expression de sélection de labels (ESL), sinon l'instruction correspondante est illisible, l'utilisateur ne peut pas travailler dessus dans de bonnes conditions. Nous avons déjà spécifié un moyen pour la collaboration verticale : toute sélection dans la vue arborescente du système de labellisation est accompagnée par l'affichage de l'ESL correspondante. Ainsi cette tâche graphique est immédiatement accompagnée par son équivalent textuel, ce qui permet une traduction utile et compréhensible en PNSL.

De manière générale, nous opposons deux types de scripts. Le premier type correspond à une description "à plat" du réseau ou d'une partie de son processus de construction. On peut l'assimiler à un langage déclaratif avec un style impératif. Il se caractérise par l'absence de structures de contrôle et l'utilisation exclusive des labels automatiques pour référencer les modules. Nous voulons le plus possible éviter une collaboration verticale basée sur ce type : elle est facile à implanter mais pratiquement inutilisable. Le second type est structuré, il correspond au processus de construction tel que pensé par le modélisateur, c'est le style qu'il utilise naturellement lorsqu'il écrit un script. Il se caractérise par l'usage extensif de structures de contrôle et de labels utilisateurs. Nous estimons que la collaboration verticale doit le plus possible être réalisée avec le style structuré car il est signifiant, il est seul qualifié à représenter correctement le processus de construction du réseau. Mais il est évidemment beaucoup plus difficile à implanter puisqu'il demande une certaine intelligence de la part de l'interface, qui doit comprendre la logique du modélisateur pour pouvoir synthétiser un script structuré à partir des seules manipulations graphiques. L'utilisation des labels est exemplaire de cette difficulté : sachant la multiplicité des labels, l'interface n'a aucun moyen systématique de deviner quelle ESL correspond à une sélection graphique donnée. La difficulté devient alors impossibilité, la transformation d'une sélection graphique en ESL ne peut être faite sans l'intervention de l'utilisateur, l'interface ne peut qu'assister le modélisateur dans cette tâche de transformation. Du point de vue pédagogique, cette tâche, qu'on pourrait pourtant considérer comme inutile, est en fait très intéressante car elle oblige le modélisateur à définir précisément la signification des tâches qu'il accomplit graphiquement : quels sont les critères topologiques d'une sélection ? A-t-on sélectionné un module au hasard ? etc... Cette précision est indispensable pour faire émerger des structures de contrôle, principalement les boucles, car pour répéter une action, celle-ci doit être décrite de la manière adéquate. C'est ce qui permettra à l'interface de détecter un invariant dans une séquence d'actions et de proposer la structure de contrôle adaptée.

Nous voulons donc, autant que faire ce peut, intégrer des ESL dans les espaces graphiques pour donner à l'utilisateur un retour bimodal sur ses actions de sélection : à la fois visuel par le surlignage de la sélection, et textuel par l'affichage d'une liste d'ESL

correspondant à la sélection. Le caractère ordonné d'une ESL, qui correspond à une liste de labels, ne doit pas être occulté car l'ordre d'une sélection est essentiel dans certaines tâches, comme la connexion. C'est là une difficulté réelle dans la mise en œuvre de cette bimodalité, car il faut faire intervenir cette notion d'ordre dans des espaces 2D et 3D, qui eux ne sont pas ordonnés. Nous illustrerons cet aspect dans la mise en œuvre d'opérations de fusion, qui requièrent une collaboration verticale de qualité pour être compréhensible et donc efficaces.

4.5. Enjeux liés à la représentation graphique du résultat d'un script

Les deux objectifs que nous venons d'énoncer démontrent la nécessité d'une représentation graphique des effets d'un script. Quelle doit être la nature de cette représentation ? Nous nous focalisons sur les représentations de l'espace topologique, espace réellement dédié à la représentation du réseau (et non pas des conditions initiales des <MAT> comme l'est l'espace géométrique). Le choix de cette représentation est guidé par des besoins précis : connecter un nombre petit ou grand de points de communication, visualiser une topologie pour vérifier son script. Il est également guidé par la nature des effets d'un script : s'agit-il de la création d'un sous-réseau complet, d'un groupe de modules, d'opérations de labellisation, de connexions, de duplications ?

Nous distinguons deux alternatives de représentation : 1) la représentation atomique, telle que nous l'avons déjà décrite, une icône pour un module 2) La représentation macro, un ensemble de modules, formant ou non un réseau, est représenté par une icône unique. La deuxième alternative, qui peut être vue comme une généralisation des macro-modules, est réservée à la représentation des scripts qui créent un groupe de modules et qui se cantonnent à agir sur ce groupe de modules. On peut qualifier ces scripts d' "autiste", ils ne connaissent pas le reste du réseau. Cette représentation présente deux avantages :

- Elle est condensée, elle permet donc une économie de représentation, particulièrement appréciable pour les grands réseaux ou groupes de modules.
- Elle ouvre la voie à une représentation graphique des LPC (labels de liste de points de communication). Ceci permet la représentation d'un grand nombre de connexions, mais aussi d'établir ces connexions via une unique action graphique.

Ces avantages ne sont pas sans contrepartie : cette représentation condensée a tendance à montrer le réseau comme un assemblage d'objets, à présenter une partition du réseau comme étant sa structure. Cependant un réseau masses-interactions n'a pas de structure unique, la structuration représentée n'est qu'une parmi une infinité et ne correspond qu'à la vision du modélisateur à un moment donné de son processus de modélisation. Ce problème de la représentation graphique est exactement le même que le problème de labellisation : ces regroupements, qu'ils soient syntaxiques ou spatiaux, ne sont qu'une vue du réseau, un outil pour sa manipulation correspondant à un instant du processus de modélisation. Par ailleurs, la représentation condensée ne permet pas de visualiser le groupe de modules contenus et par là même de vérifier que la topologie est bien celle désirée.

La représentation atomique résout en partie ces problèmes, en partie seulement car le placement des modules sur l'espace topologique suggère également une partition, mais la possibilité de modifier facilement ce placement amoindrit considérablement cet effet. De plus, ce placement des modules ne peut être automatique, il est donc à la charge du modélisateur, qui devra le prendre en compte dans son script pour justement lui permettre de visualiser au mieux la topologie qu'il a créée. En revanche, la représentation atomique reste inadaptée à la représentation d'un grand nombre de modules par la surcharge visuelle qu'elle génère.

4.6. Conclusion

Il est intéressant d'analyser les différents points que nous avons abordés dans cette partie à l'aune des cinq primitives de Martin [Mar99]. Les collaborations horizontale et verticale constituent ainsi deux types de complémentarité. Le fait de travailler sur le processus comme sur l'état concerne principalement la modalité script, mais permet d'insister sur l'équivalence entre les deux modalités. La collaboration verticale montre aussi la nécessité d'une redondance entre les deux modalités, comme l'illustre bien la représentation graphique de l'arbre des labels. La redondance est également en question dans les représentations graphiques du résultat d'un script, même si l'on peut s'interroger sur le fait qu'une représentation de l'état soit redondante par rapport à une représentation du processus. Le transfert est un moyen pour la complémentarité, on l'observe bien pour la collaboration horizontale, où ce transfert s'opère via les labels ou via une représentation graphique redondante.

Nous avons également lancé les bases d'une collaboration avec toujours ce même objectif de complexité en ligne de mire. Les outils que nous allons proposer se justifient donc par rapport à cet objectif, ils tenteront de réaliser la collaboration entre les deux modalités telle que nous l'avons décrite dans cette partie et de répondre aux problèmes de fond qu'elle pose.

5. Outils pour la collaboration entre modalité graphique et modalité script

5.1. Introduction

Nous disposons de deux familles d'outils, la première est celle du champ graphique :

- Un espace topologique et sa palette de modules permettant de créer graphiquement un réseau
- Différentes boîtes de dialogue/widgets permettant de fixer les paramètres des modules sélectionnés
- Un espace géométrique permettant de déterminer graphiquement les valeurs initiales des modules

La deuxième est le langage PNSL et toutes ses fonctionnalités (labelliser, dupliquer, fusionner). La première famille constitue le cadre de travail initial de l'interface MIMESIS, qui reste une interface basée sur le modèle WIMP. Notre problématique dans cette partie, qui constitue le cœur de notre contribution, est de proposer une intégration de PNSL dans cette interface qui satisfasse les conditions que nous avons présentés dans la partie précédente : "Travailler sur le processus comme si on travaillait sur l'état", partager le processus de création du réseau entre modalité graphique et modalité script, travailler sur des représentations du réseau qui respectent sa nature systémique.

Pour tenter de répondre à cette problématique, nous avons basé l'intégration du script sur un concept fort : sa réification au sein de l'interface. Ce choix est très rarement fait dans les interfaces WIMP en général, les concepteurs préfèrent le concept de macro, objet qui reste extérieur au projet et qui n'a pas d'autre lien avec lui autre que son propre contenu (i.e. le code). Cette réification va permettre à l'utilisateur de travailler sur le script, c'est à dire sur le processus même, et au delà de définir plusieurs objets scripts, et donc plusieurs processus, chacun ayant son champ d'activité, de même que l'espace topologique, ainsi que l'ensemble des modalités graphiques, définissent également des champs d'activité. Le fait de réifier va nous permettre de définir des relations entre ces différents champs d'activités : par exemple, peut-on influer sur le champ d'activité du script A avec des manipulations dans l'espace topologique ? Nous aurons donc à gérer un compromis entre deux orientations : définir des relations très ouvertes compromet la "représentativité" d'un objet script, mais fermer ces relations

réduit les possibilités créatives et tend à associer un script à une sous-partie de réseau, favorisant ainsi une vision objet du réseau. En effet, les deux solutions que nous proposons se situent chacune d'un côté de la barrière entre ces deux extrêmes, une barrière relativement floue que ces deux outils proposeront justement de traverser. En revanche, l'interface propose exclusivement ces deux solutions pour la modalité script, l'exécution d'un script en mode macro n'est pas possible, le script est obligatoirement réifié dans MIMESIS, ce qui nous permet, en tant que concepteurs de l'interface, de maîtriser l'usage de cette modalité. Le concept de réification provient de l'interaction instrumentale de Michel Beaudoin-Lafon [BL00]. Nous décrirons donc dans un premier temps cette théorie sur les applications graphiques pour montrer comment elle s'articule avec notre problématique. Nous détaillerons ensuite chacune des deux solutions que nous proposons, la *fonction* et la *capsule*.

5.2. L'interaction instrumentale

L'interaction instrumentale, telle que défini par Michel Beaudoin-Lafon dans [BL00], part d'un constat que nous partageons très largement : les interfaces graphiques actuelles font intervenir toute une collection d'objets "qui ne font pas directement partie du domaine de l'application, mais sont indispensables à sa mise en œuvre". Or, le premier principe de la manipulation directe d'après Ben Shneiderman ne fait référence qu'aux objets d'intérêt de l'application, ainsi les applications graphiques ne traitent pas ces *objets auxiliaires* comme des objets d'intérêt. Ces objets sont par exemple les styles dans Microsoft Word ou les filtres dans Adobe Photoshop. Michel Beaudoin-Lafon introduit donc la notion d'instrument d'interaction comme "médiateur entre les actions de l'utilisateur et les objets de l'application". Cette notion lui permet de définir le processus de réification qui permet de transformer un instrument en un objet d'intérêt à part entière, et donc éditable, duplicable, enregistrable.

L'introduction du concept d'interaction instrumentale a pour objectif principal de définir un paradigme pour les interfaces dites "post-WIMP". La définition d'un instrument en deux parties, une logique et une physique, permet d'introduire des techniques d'interactions basées sur d'autres périphériques que la souris et d'autres activités que le pointage, notamment toutes les interactions bimanuelles et aujourd'hui les gestes basés sur une interface multi-points. Cependant les concepts mis au point par Beaudoin-Lafon pour une interaction graphique peuvent également être mis en œuvre pour une interaction langagière. La réification est ainsi un concept adapté à la mise en œuvre de la collaboration entre modalité script et modalité graphique. En effet, donner à un script le statut d'objet de l'interface permet de le manipuler en tant que tel et de définir des relations avec les autres objets, et donc avec leurs représentations graphiques. Ce sont ces relations qui permettent de définir la collaboration. Les deux solutions que nous proposons sont donc définies sur cette base, proposer le script comme un instrument et un objet d'intérêt. Nous proposons également en annexe 4 une brève analyse du logiciel KpovModeler [Kpo], qui permet une collaboration entre les deux modalités en réifiant le script.

5.3. Fonction

5.3.1. Présentation

La fonction se situe du côté ouvert de la collaboration graphique/textuel, elle laisse une plus grande liberté à l'utilisateur, qui en contrepartie est responsable de la cohérence entre ses scripts et l'état du système. Concrètement, nous proposons de réifier le script comme un objet d'un nouveau type, nommé fonction. Cet objet présente un attribut principal, le script, dont le champ d'action s'étend à tout le système : le réseau, ses paramètres, le système de labellisation, et les différentes représentations graphiques. Cette largeur du champ d'action ouvre des possibilités nombreuses mais présente deux inconvénients, conséquences directes de cette largeur. D'abord, une représentation iconique d'une fonction n'est pas envisageable, en effet, le script ne construit pas forcément un assemblage de modules, il peut par exemple se limiter à l'interconnexion de deux groupes de modules. La représentation atomique sur l'espace topologique s'impose donc, favorisant une vision systémique du réseau mais obligeant l'utilisateur à définir des positions dans l'espace topologique pour les modules <MAT>. Par ailleurs, la représentation atomique demeure inadaptée pour les modèles de très grande taille, ainsi un agglomérat de 1000 masses nécessiterait l'affichage d'un million de modules <LIA> ! Deuxième inconvénient consécutif à la liberté permise par la fonction, le script est fragilisé, un grand nombre de manipulations dans le domaine graphique, ou bien dans une autre fonction, peuvent remettre en question la validité du script comme représentation correcte d'un fragment du processus de construction du réseau. Autrement dit, la modification d'une entité utilisée ou créée par le script détruit la représentativité du script. La fonction est donc ouverte, flexible, mais en contrepartie fragile.

5.3.2. Etat d'une fonction et état du système

Pour pallier à cette fragilité du script, nous définissons des relations entre la fonction, qui est en quelque sorte l'interface du script, et le système dans son ensemble, c'est à dire tout ce qui peut être référencé, modifié, créé par le script. C'est ici que la réification du script prend tout son sens : le fait de faire correspondre un script à un objet du noyau, un objet fonction, permet de créer des relations entre cet objet et les autres entités du système (système de labellisation, modules, représentations graphiques des modules), ainsi qu'avec certaines de leurs propriétés (paramètres physiques et initiaux des modules, connexions des modules). Ces relations sont créées à l'exécution du script et ont deux rôles. Premièrement, un script est exécuté et modifié plusieurs fois au cours du processus global de modélisation, chaque nouvelle exécution signifie une modification de l'existant et non un ajout à l'existant. En conséquence, lorsqu'une fonction exécute son script, elle doit annuler tout ce qu'elle avait produit à l'exécution précédente, sous peine de créer involontairement de nouveaux modules, voire provoquer une erreur en créant une entité déjà existante (par exemple un label). Deuxièmement, toutes les modifications réalisées à l'extérieur de la fonction, mais qui modifient le résultat de son exécution, peuvent être traitées de plusieurs manières : 1) Ces modifications sont bloquées, par exemple si une fonction effectue une connexion entre un point L et un point M, il n'y a aucun autre moyen de défaire cette connexion que de modifier le script 2) Ces modifications signalées à l'utilisateur, libre à lui d'exécuter une nouvelle fois le script ou de le modifier en conséquence 3) Ces modifications sont traduites sous forme

d'instructions PNSL insérées en fin du script concerné. Cette troisième voie peut paraître intéressante, mais elle ne rétablit pas forcément la cohérence du script avec l'état du système. Par exemple, si la modification concerne un label utilisé dans le script, ajouter la suppression ou la redéfinition de ce label ne rétablira pas la cohérence, qui ne pourra d'ailleurs pas s'exécuter correctement (puisqu'il appelle un label qui n'existe plus).

Nous avons choisi la deuxième voie, qui offre le plus de flexibilité à l'utilisateur mais l'oblige à une certaine responsabilité. L'interface présente ces notifications en trois parties :

- Modifications du résultat du script (suppression d'un module créé, modification d'une connexion, ...)
- Modifications des objets référencés par le script (renommage ou destruction d'un label, édition d'un LPC, ...)
- Modifications des paramètres de représentation (positions des modules sur l'espace topologique, point de vue sur l'espace topologique)

La sensibilité d'une fonction à des modifications de labels mérite un approfondissement. Un script référence des labels par le moyen d'expressions de sélection de labels (ESL). La fonction porteuse de ce script doit donc être invalidé si la signification de cette ESL est modifiée, c'est à dire si une manipulation quelconque change la liste de modules désignés par cette ESL. Une fonction n'est donc pas liée à des labels mais à des familles de labels. Prenons l'exemple d'un script faisant appel à l'ESL `/chose/>>`, qui référencie tous les labels ayant pour préfixe `/chose/`. Si l'utilisateur crée un nouveau label ayant ce préfixe, la fonction portant le script doit être invalidée. La fonction est donc liée au conteneur nomme /chose, et doit donc être invalidée si ce conteneur est renommé, ou si l'arbre issu de ce conteneur est modifié d'une quelconque manière.

L'utilisation d'une fonction est donc très simple, l'utilisateur commence par une première phase de création, dans laquelle il peut donner un nom à sa fonction, puis enchaîne des cycles d'édition/exécution du script, tout en surveillant la validité de la fonction tout au long du processus de modélisation. Nous insistons sur le fait que la re-exécution d'une fonction sans réédition n'est pas équivalente à un retour en arrière : l'ensemble du contexte a changé, ce qui peut affecter l'exécution de la fonction, jusqu'à provoquer une erreur (un appel à un label supprimé par exemple). Cette extrême flexibilité de la fonction est très intéressante, elle permet de concevoir des scripts qui se réadaptent en fonction du contexte, mais elle requiert une attention particulière à l'évolution de ce contexte.

5.3.3. Intégration à l'interface

Les fonctions sont accessibles via un tableau dédié (cf. figure 58).

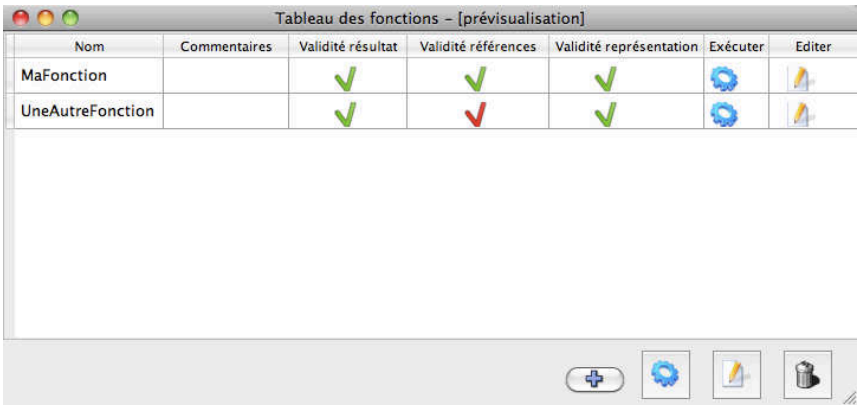


Figure 58 : Le tableau des fonctions

Il est ainsi possible de créer et supprimer des fonctions, de consulter leurs validités, de leur donner un nom et des commentaires, d'éditer et d'exécuter son script. L'édition du script se fait dans une fenêtre d'édition dédiée. Cette présentation est un premier essai destiné à évoluer pour permettre une organisation plus complexe des fonctions et une interaction plus riche avec les autres éléments de l'interface.

5.4. Encapsulation

5.4.1. Présentation

L'encapsulation propose de réifier le script dans une classe d'objet déjà existante : les modules. Ce nouveau type de module est nommé capsule. La capsule est donc un module dont les éléments (algorithme et points de communication) sont définis par un assemblage de modules <MAT> et <LIA>, ne formant pas nécessairement un réseau bien formé (i.e. les <LIA> ne sont pas forcément connectés). Cet assemblage de modules est défini par un unique script, moyen d'interaction principal de l'utilisateur.

La capsule se pose clairement du côté fermé, protégé, de la collaboration, c'est dans cette logique qu'elle dispose de son propre projet, contenant l'assemblage de modules, un système de labellisation et sa représentation graphique, ainsi qu'un espace topologique. Ce projet est isolé du projet principal, leurs systèmes de labellisation s'ignorent respectivement, prévenant ainsi toute modification de la capsule depuis l'extérieur. Cette protection permet de travailler sur le script en garantissant qu'il représente correctement à tout moment l'assemblage de modules et le système de labellisation.

5.4.2. Conception de l'assemblage de modules

L'utilisateur définit l'assemblage de modules selon une bimodalité très simple, pouvant rappeler le mode d'édition des éditeurs html en WYSIWYG[6] : l'utilisateur alterne entre vue graphique et vue textuelle. A la différence qu'ici PNSL est un langage impératif et non

[6] What You See Is What You Get

déclaratif comme le html. Chaque manipulation dans l'espace graphique se traduit donc par l'adjonction de commandes en fin de script. Cette bimodalité alternée permet de garantir à tout moment l'adéquation entre le script et l'état du système (assemblage de modules et système de labellisation principalement). La conception se fait donc selon une collaboration verticale relativement simple, qui n'est pas l'intérêt principal de la capsule, d'abord conçue pour être édité textuellement.

5.4.3. La capsule et ses moyens de connexion

Les points M de la capsule sont définis automatiquement : un point M pour chaque <MAT>. Les points L de la capsule sont également automatiquement définis : tout point L non connecté est défini comme un point L de la capsule. L'interface structurelle, de la capsule, c'est à dire ses points de communication et leurs moyens d'accès, est donc définie de manière très rudimentaire : une liste de points M et une liste de points L accessibles via leur indice dans la liste. Ces moyens apparaissent comme insuffisants. D'abord, sémantiquement les indices sont difficiles à manipuler, l'utilisateur a du mal à percevoir à quels modules internes ils correspondent. Ensuite graphiquement, vu le nombre potentiellement très important des points de communication, une représentation atomique des points à la périphérie de l'icône représentant une capsule est définitivement inadaptée. C'est ici que les labels de liste de points de communication (LPC, présentés en §5.3) prennent tout leur intérêt. Ils peuvent être définis depuis le projet de la capsule, non pas à partir des indices des listes de points M et L, mais directement à partir des modules qui en sont à l'origine. La procédure suivante, propre aux projets d'une capsule permet cette définition :

```
définir_LPC_surface nom_LPC ESP_points
```

Avec les LPC, l'interface d'une capsule devient beaucoup plus manipulable à l'égard des deux modalités. Au niveau du langage, une LPC est une expression compacte et signifiante pour une liste de points de communication. Au niveau graphique, la représentation des LPC est une option plus adaptée que l'option atomique, en revanche elle modifie la représentation graphique classique des connexions sur l'espace topologique : une connexion à une capsule ne peut plus se représenter comme la fusion d'un module <BLC> (représentant un point L) et d'un module <MAT> (représentant un point M) comme indiqué en figure 59, il faut représenter les "câbles" connectant une liste de points à une autre (cf. figure 60).

Figure 59 : Représentation fusion

Figure 60 : Représentation câblée

Cette représentation permet de connecter graphiquement une liste de points L à une liste de points M de même taille ou bien à un unique point M, comme l'illustre la figure 61.

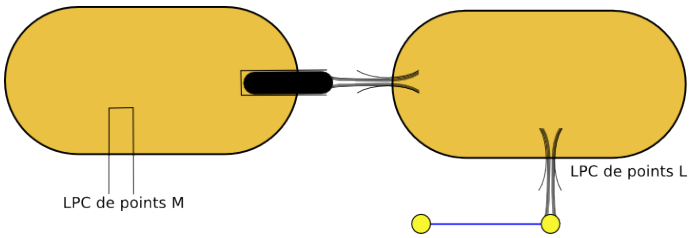


Figure 61 : Deux capsules et leurs connexions

Cette figure montre les représentations graphiques des LPC de points M et de points L. Ces deux représentations peuvent être vues comme des métaphores de fourreaux : l'utilisateur tire des câbles depuis un fourreau, ces câbles sont issus d'une liste de points L. Il tire donc ces câbles jusqu'à les "brancher" en les introduisant dans un autre fourreau conduisant à un point M ou a une liste de points M.

Le placement des représentations graphiques des LPC à la surface de la capsule est automatique mais personnalisable. Cette personnalisation peut être effectué textuellement dans une boîte de dialogue, ou graphiquement. L'utilisateur peut également désactiver la représentation graphique d'une LPC. Toutes ces personnalisations mettent l'accent sur les LPC comme principal moyen de connexion d'une capsule, mais nous avons voulu accentuer cet aspect en faisant des LPC le moyen exclusif de connexion d'une capsule. En effet, une connexion via les indices implique une création de LPC relative à ces indices mais aussi éventuellement d'une LPC relative au complémentaire de ces indices. En conséquence, pour assurer la compatibilité entre la méthode de connexion via les indices de points de communication et la méthode de connexion via les LPC, il faut accepter une surcharge importante de la surface de la capsule, surcharge qui n'est ni souhaitable ni aisément compréhensible pour l'utilisateur.

Nous avons spécifié un critère imposant qu'une tâche soit réalisable dans les deux modalités, ce critère est respecté puisque la connexion d'un LPC peut être faite dans une fonction. Nous voulons aussi favoriser la collaboration verticale et encadrer la connexion par script des LPC, que nous considérons comme une tâche bien spécifique dans la construction d'une topologie. Dans cette optique, nous proposons des fonctions spécialisées dans la connexion de LPC. Ainsi connecter une LPC textuellement se fait obligatoirement par le biais d'une telle fonction, symétriquement une connexion effectuée graphiquement génère automatiquement une fonction de ce type. L'utilisateur dispose ainsi d'une sorte d'historique dédié aux connexions des capsules et des macro-modules. A la différence d'un historique classique, celui-ci n'est pas linéaire et il est éditable. Il n'est pas linéaire car on peut revenir sur une action passée sans défaire toutes les actions qui lui ont succédé. Il est éditable car une action étant représentée par une fonction, on peut éditer le script de cette fonction. Cet historique est aussi sensible au contexte, en effet, si une capsule est éditée, toutes les fonctions relatives aux connexions de cette capsule sont invalidées car après édition d'une capsule toutes ses LPC sont déconnectées.

Le dernier point concernant la connexion des capsules concerne les points L non connectés. Contrairement à la représentation atomique, l'utilisateur n'identifie pas clairement les points L non connectés d'une capsule, il est sensé maîtriser la définition de sa surface paramétrique pour pouvoir déduire si oui ou non tous ses points L sont connectés. Il nous paraît intéressant de signaler graphiquement qu'une capsule n'est pas

entièrement connecté sans toutefois préciser quels points ne sont pas connectés, encore une fois pour ne pas alourdir la représentation graphique et de manière plus générale ne pas surcharger l'utilisateur d'informations.

5.4.4. Capsule et contrôle de paramètres

Nous avons montré l'intérêt du contrôle de paramètres dans le chapitre II. Nous l'avons démontré notamment sur des modèles de petite taille, mais il est nécessaire de permettre le contrôle de paramètres sur un grand nombre de modules, par exemple pour un modèle de changement d'état. Ce changement d'échelle nécessite l'adaptation des procédures de manipulation du contrôle de paramètres aux capsules.

Tous les paramètres contrôlables sont contrôlables via une capsule, c'est à dire les paramètres de tous les modules à l'exception des LLM génériques. Dans ce cadre, il s'agit de rendre accessibles à la surface de la capsule les entrées de paramètre des modules internes. Nous cherchons donc à définir une opération semblable à la déclaration des labels de listes de points de communication (LPC). Nous avons défini les expressions de sélection de points d'entrée de paramètre (ESEP) au §2.8. Pour définir un élément sur la surface de la capsule, il est nécessaire d'objectiver cette expression, de la même manière qu'une LPC était l'objectivation d'une expression de sélection de points de communication. Nous définissons donc les labels de liste d'entrée de paramètres (LEP), qui permettent de nommer une ESEP. Nous pouvons à présent étendre la définition d'une ESEP, qui est composée d'une expression de sélection de labels et d'un paramètre ou bien d'une expression de sélection de labels et d'une ESEP. La procédure PNSL permettant de déclarer un label de point d'entrée de paramètre à la surface de la capsule est analogue à la procédure dédiée aux LPC de surface. Une fois un LEP déclaré sur une capsule, le contrôle des paramètres mis en surface peut se faire comme pour les autres modules avec la modalité de son choix.

5.5. Conclusion

La fonction et la capsule sont deux outils adaptés pour la collaboration entre modalité graphique et modalité script telle que nous l'avons spécifiée pour MIMESIS V. Ils permettent notamment le travail sur le processus ainsi que les collaborations horizontale et verticale. Cette dernière est certes rendue possible par ces deux outils mais dans des conditions encore peu satisfaisantes, car nous n'avons pas approfondi la problématique de la traduction d'actions graphiques en script dit structuré (comportant des structures de contrôle et des labels utilisateurs). Ce sujet est assez proche de la programmation par démonstration [Hal84], où l'on retrouve d'ailleurs l'idée originale de la métaphore collaborative de Hutchins.

Cependant, aucun de ces deux outils ne respecte complètement les conditions que nous avons fixées dans la partie précédente. Mais aucun outil ne pourrait remplir à lui seul l'ensemble de ces conditions, car certaines sont intrinsèquement incompatibles, comme la vision objet du réseau et la manipulation de réseaux de très grande taille. Nous estimons en revanche que ces deux outils sont complémentaires et que c'est dans leur association que l'utilisateur pourra trouver les conditions adéquates pour son activité de modélisation. Leur complémentarité s'illustre dans le fait que l'on peut transformer une

capsule en fonction, à condition de définir des positions sur l'espace topologique, et qu'inversement on peut transformer une fonction en capsule, si la fonction construit un sous-réseau bien formé. Ces opérations de transformation ne sont pas directement disponibles dans l'interface, mais il suffit de conserver le script de l'une pour l'intégrer dans l'autre, moyennant un petit nombre de modifications (déclaration des LPC pour les capsules). Ces deux outils sont ainsi compatibles parce qu'ils sont tous les deux des objets de l'interface et qu'ils partagent les mêmes liens avec le résultat de leur script. Cette complémentarité permet, comme nous le verrons dans la partie consacrée aux exemples, de bénéficier des avantages respectifs des deux outils. Par exemple, la topologie d'un réseau peut être conçue dans un premier temps à petite échelle à l'aide de fonctions et de manipulations graphiques, puis à plus grande échelle à l'aide de capsules

6. Organisation et usage des outils dans MIMESIS V

6.1. Organisation de MIMESIS en 5 temps

Le processus de modélisation de MIMESIS V est composé de cinq phases :

1. Construction et édition de la topologie du réseau (PSQL : Pré-Structuration QuaLitative)
2. Edition des paramètres physiques des modules (PSQN : Pré-Structuration QuaNtitative)
3. Edition des valeurs initiales des modules <MAT> (CI : Conditions initiales)
4. Gestion de la simulation, projection en film (SIMU)
5. Habillage des trajectoires (HABI)

Le processus est cyclique, chaque phase est donc revisitée plusieurs fois pendant le processus. La première passe se fait toujours rigoureusement dans l'ordre indiqué (comme préconisé dans [Evr09]). Les retours sur les autres phases se font ensuite dans un ordre non déterminé, selon les besoins du modélisateur, mais qui termine systématiquement sur la phase de simulation, qui est la phase d'observation du résultat permettant d'obtenir le résultat final. C'est la stricte séparation de ces cinq phases qui est cruciale pour ouvrir les possibilités au modélisateur dans son processus de modélisation, notamment la séparation des phases PSQL et PSQN, comme démontré dans [Evr09]. Cette séparation se traduit spatialement dans MIMESIS 4, qui propose une interface figée exposant directement les 5 phases du processus de modélisation. Cette organisation avait un intérêt pédagogique car elle montrait un parcours de création, en revanche elle était peu efficace au niveau ergonomique car à chaque phase une grande partie de l'écran était inutilisée. Nous proposons d'organiser l'interface en plusieurs temps, un pour chaque phase, chaque temps correspondant à une organisation de l'espace de travail adaptée à chaque phase. L'espace est pré-organisé mais pas figé, l'utilisateur a la possibilité de réagencer les différents éléments, d'en ajouter certains, mais il reste dans le contexte de la phase. Ainsi, s'il se trouve dans le temps PSQL, il ne pourra pas modifier les paramètres physiques ni les conditions initiales, quelle que soit la modalité (graphique ou script). Par ailleurs, tout retour à la phase PSQL oblige à repasser par les phases PSQN et CI, et éventuellement HABI, avant de passer à la phase de simulation. Ainsi, les changements topologiques peuvent être pris en compte pour définir tous les paramètres et conditions initiales des modules nouvellement créés. En effet,

MIMESIS V ne définit pas automatiquement des paramètres et conditions initiales par défaut. En revanche, au premier passage sur ces phases, des fonctions permettant d'effectuer cette définition sont rendues disponibles à l'utilisateur. Il prend ainsi lui même la responsabilité de les exécuter.

6.2. Fonctions, capsules et séparation des phases

Les fonctions et les capsules ont d'abord été conçues pour PSQL. La séparation des phases nous oblige cependant à définir leur statut dans les autres phases. En ce qui concerne les fonctions, leur flexibilité leur permet d'intégrer simplement les autres phases. En effet, chaque phase possède son ensemble de fonctions et son contexte d'exécution. Le contexte d'exécution définit quelles sont les procédures accessibles pour chaque phase, ainsi le contexte de PSQN interdira l'utilisation des procédures de création de modules, de suppression et de connexion. Les outils avancés, tels que la duplication sont par ailleurs strictement réduits à PSQL, ainsi les modules d'un duplicata présenteront toujours des paramètres par défaut, même si les modules dupliqués ont déjà été paramétrés. En revanche, la fusion, qui est également circonscrite à PSQL, vérifie quand même l'identité des paramètres des modules qu'elles fusionnent, car cette opération doit conserver son caractère symétrique.

La capsule pose davantage de problèmes pour intégrer PSQN à cause de son caractère fermé. En effet, si ce caractère est déjà dommageable pour PSQL, il est rédhibitoire pour PSQN et les autres phases, car il faut pouvoir établir des groupements de modules qui font éclater le cadre de la capsule. Nous proposons donc d'ouvrir la capsule dans le contexte des phases PSQN, CI, SIMU et HABI, c'est-à-dire de rendre accessibles et modifiables ses modules internes. Concrètement, ceci se traduit par la notion d'espace de noms (*namespace*) propre à un projet. Chaque projet capsule possède son espace de noms, dans lequel prend place le système de labellisation. Il est donc possible d'accéder à ces labels depuis le projet principal en préfixant les expressions de sélection de labels par `//nom_du_namespace`. Cependant le système de labellisation reste protégé en écriture pour garantir l'intégrité de la capsule. Cette possibilité est surtout intéressante pour la phase PSQN, en effet les phases CI et HABI n'ont besoin que des références vers les points M, déjà accessibles par les labels de liste de points de communication. Toujours pour préserver l'intégrité de la capsule, le contexte de son script est réduit à la définition de la topologie et du système de labellisation.

6.3. Outils auxiliaires

Nous avons mis l'accent sur l'utilisation de listes, listes de labels, listes de points de communication. Nous devons donc proposer des outils auxiliaires pour manipuler de manière souple et puissante les listes. PNSL est un langage de script construit sur Tcl, langage qui donne déjà un certain nombre d'outils de manipulation de listes. Nous nous sommes donc appuyés sur l'existant pour proposer une bibliothèque de fonctions dont nous donnons ici un extrait :

- Construction de listes d'indices suivant des règles mathématiques (par exemple les entiers pairs entre 4 et 54).
- Répétition de listes
- Filtres basés sur des règles mathématiques
- Permutation circulaire des modules désignés par une liste de labels
- Opérateurs de filtrage pour expressions de sélection de labels pour avoir une seule occurrence d'un module dans une liste.
- Expressions de sélection aléatoire de labels : pour piocher au hasard un certain nombre de labels à partir d'une expression de sélection de labels.

Toutes ces fonctions peuvent être implantés par un utilisateur avancé en utilisant le langage Tcl. C'est donc davantage une bibliothèque extensible que nous proposons qu'un ensemble réduit de fonctionnalités avancées. PNSL est donc un langage extensible qui pourra évoluer grâce à l'apport d'utilisateurs avancés, sous couvert d'une validation, indispensable pour veiller au respect des principes de modélisation.

6.4. Pédagogie

La pédagogie de MIMESIS V ne diffère pas fondamentalement de celle mise en place pour MIMESIS 4 et décrite par Matthieu Evrard dans [Evr09]. Nous procédons ainsi d'abord par une première étape directive, montrant les différentes phases du processus de modélisation, ainsi que les outils de base de l'interface. Cette étape directive est illustrée par la conception d'un modèle très simple, le rebond d'une bille sur un sol, modélisé par un <SOL> et une <MAS> en interaction de butée. Dans les outils de base exposés, PNSL est immédiatement mis en avant à travers l'utilisation de l'outil fonction. Même si la topologie très simple du modèle oriente vers une conception en manipulation directe, le formateur montre les deux modalités. Le concept de label est ainsi exposé très rapidement. Tous les outils plus évolués que nous avons présenté ne sont pas exposés dans cette première étape destiné aux débutants : capsule, LPC, filtres, duplication et fusion.

Après cette première étape, les apprenants sont invités à explorer par eux même ce modèle simple. Nous guidons cette exploration en adoptant la démarche inverse de la conception, depuis la phase HABI vers la phase PSQL, en observant à chaque étape l'effet des changements qu'ils opèrent. Cette deuxième étape permet à l'apprenant de se familiariser avec l'interface mais aussi avec la philosophie qu'elle propose. En effet, commencer par manipuler l'habillage permet de faire immédiatement le décadrage qui met la forme après le mouvement, souvent surprenant pour des utilisateurs habitués aux logiciels plus classiques de synthèse d'image. Cette étape est aussi l'occasion de mettre l'accent sur PSQN, de montrer comment sur un modèle minimal, un spectre non négligeable de comportements peut être obtenu en jouant sur un nombre réduit de paramètres.

L'apprenant peut ensuite explorer dans une troisième étape d'autres modèles, plus complexes, dans la bibliothèque que nous lui proposons. Selon sa sensibilité, cette étape peut être l'occasion de découvrir de nouveaux outils, comme la duplication ou les

capsules, pour aller vers plus de complexité topologique. Le formateur laisse l'apprenant assez libre, et profite de cette étape pour approfondir le concept de label, insister sur son aspect non structurant. Pour les utilisateurs de MIMESIS 4, le changement le plus marquant par rapport aux labels est sans aucun doute la distinction entre label identifiant (label auto) et label utilisateur. L'utilisateur prend conscience d'un besoin du noyau, l'identifiant unique, et perçoit mieux ainsi l'arbitraire que représentent les différents labels qu'il peut créer pour un même module.

La quatrième et dernière étape consiste enfin à laisser l'apprenant partir de la page blanche, en suivant "l'observation abstractisante" décrite au chapitre I §3.1. L'apprenant quitte alors peu à peu son statut d'étudiant pour devenir un praticien de l'art du mouvement.

6.5. Conclusion

L'interface ne peut pas contraindre l'utilisateur à penser "modélisation physique particulaire", c'est à dire à penser systémique plutôt que structuraliste, à voir tous les modules en relation plutôt que d'identifier des objets, à mettre la géométrie avant la physique. L'interface ne contraint pas l'utilisateur, elle l'encourage, et c'est toute une pédagogie qui doit venir compléter l'étant donné de l'interface pour parfaire cet encouragement. La pédagogie consiste, dans un premier temps, à développer la philosophie de l'Art du Mouvement et de la modélisation physique particulaire *en dehors* de l'interface. Mais parallèlement, cette philosophie se doit d'être illustrée par des exemples concrets d'utilisation de l'interface, montrant comment les outils permettent une activité de modélisation en accord avec la philosophie.

7. Scenarii

7.1. Introduction

Les différents scenarii exposés dans cette partie ont un double rôle. Ils sont d'abord illustratifs des possibilités permises par MIMESIS V, et en premier lieu les outils pour la collaboration. Les scenarii, consacrés exclusivement à PSQL, montrent différentes stratégies de construction de topologies. Ces stratégies ne sont pas un catalogue dans lequel viendrait puiser l'utilisateur, elles sont des exemples d'utilisation des outils présentés dans la deuxième partie de ce chapitre. Les topologies présentées ne sont pas pensées dans un processus de modélisation complet, mais dans l'objectif d'exposer leur diversité. Nous tentons donc de démontrer deux choses : la diversité des topologies et la qualification des outils pour atteindre cette diversité, qui fait, en partie, la complexité des modèles masses-interactions.

7.2. Agglomérat et oscillateur

7.2.1. Description

Le modèle est constitué d'un agglomérat de 300 masses en interaction de cohésion, les 300 masses de l'agglomérat sont en interaction de ressort frottement avec la masse libre d'un pendule. Nous étudions ici la construction de ce modèle uniquement au niveau PSQL et différentes modifications de sa topologie : augmentation du nombre de masses de l'agglomérat, substitution des interactions de l'agglomérat, changement du nombre de masses de l'agglomérat en interaction avec la masse libre du pendule.

7.2.2. Scénario fonction

Construction du modèle

La première étape consiste à construire l'agglomérat de 300 masses en interaction de cohésion. L'utilisateur crée une fonction nommée mon_agglo dans laquelle il va saisir le code suivant :

```
%création et labellisation des 300 masses

Labelliser [créer_module 300 MAS_3D] /agglo/masse1..

%création et labellisation des n*(n-1)/2 interactions nécessaires
```

```
Labelliser [créer_module [expr 300*299/2] COH3_3D] /agglo/cohesion1..
%connexion des interactions aux masses
affecter cur %indice de l'interaction à connecter
pour i de 1 a 299 faire{
      pour j=i+1 a 300 faire{
      connecter_LIA /agglo/cohésion$cur /agglo/masse$i
                  /agglo/masse$j
            incr cur
      }
}
%disposer les modules en cercle sur l'espace topologique
placer_cercle /agglo/mas1..300 –centre 0 0 –rayon 5
```

L'utilisateur exécute le script et observe le résultat décrit en figures 62 et 63 sur l'espace topologique :

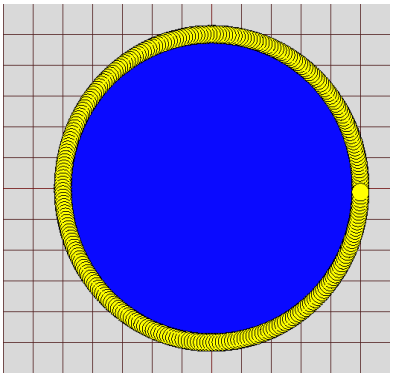


Figure 62 : Représentation graphique d'un agglomérat de 300 masses

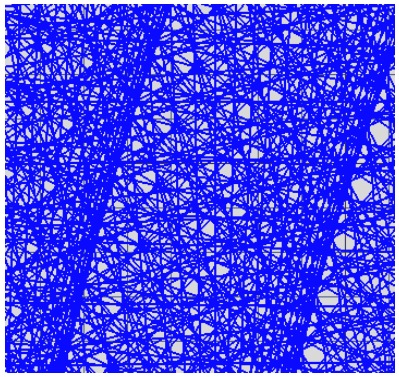


Figure 63 : Zoom sur l'agglomérat

Il crée ensuite le pendule à l'aide de la palette. Pour créer et connecter des interactions au pendule il faut rendre celui-ci accessible à la modalité textuelle, l'utilisateur doit donc labelliser la masse libre du pendule ou bien récupérer son nom automatique. C'est ce dernier choix qui est retenu dans le script suivant :

```
%Création et labellisation des interactions
labelliser [créer_module 300 REF_3D] /inter/ressort1..
pour i de 1 a 300 faire{
connecter_LIA /inter/ressort$i /agglo/masse$i @45154
}
```

L'utilisateur exécute le script et observe le résultat décrit en figures 64 et 65 sur l'espace topologique.

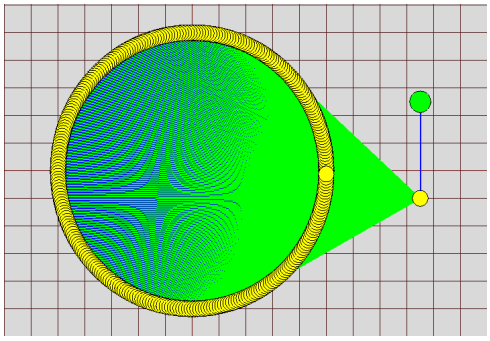


Figure 64 : Agglomérat en interaction avec un pendule

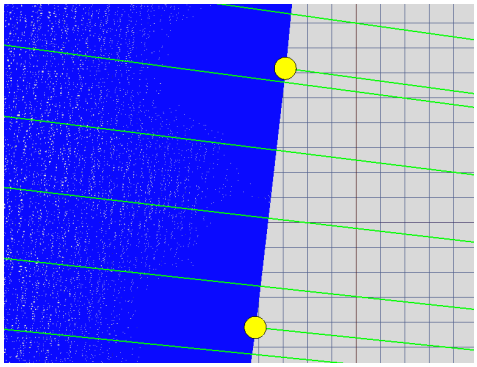


Figure 65 : Zoom sur l'agglomérat en interaction avec le pendule

Changement du nombre de masses de l'agglomérat

L'utilisateur doit éditer les scripts des deux fonctions pour remplacer toutes les occurrences de 300 et 299 par les nombres appropriés. S'il désire faire varier plusieurs fois le nombre de masses, cette procédure est peu adaptée, il décide donc d'utiliser une unique variable pour gérer ce qu'on pourrait appeler un paramètre topologique. Le script de la fonction "agglo" est donc modifié de la manière suivante :

```
%Déclaration du nombre de masses dans l'agglomérat
Affecter nb_masses 200
%création et labellisation des nb_masses masses
Labelliser [créer_module $nb_masses MAS_3D] /agglo/masse1..
%création et labellisation des nb_masses*(nb_masses-1)/2 interactions
nécessaires
Labelliser [créer_module [expr $nb_masses*($nb_masses-1)/2] COH3_3D]
/agglo/cohesion1..
%connexion des interactions aux masses
affecter cur %indice de l'interaction à connecter
pour i de 1 a [expr $nb_masses – 1] faire{
      pour j=i+1 a $nb_masses faire{
            connecter_LIA /agglo/cohésion$cur /agglo/masse$i
                        /agglo/masse$j
            incr cur
      }
}
%disposer les <MAT> en cercle sur l'espace topologique
placer_cercle /agglo/mas1..$nb_masses –centre 0 0 –rayon 5
```

Le script de la fonction "inter" est également modifié pour se connecter à toutes les masses de l'agglomérat, quelqu'en soit le nombre :

```
%compter le nombre de masses de l'agglomérat

Affecter nb_inter [compter_label MAS3D:/agglo/>]

%Création et labellisation des interactions

labelliser [créer_module $nb_inter REF_3D] /inter/ressort1..

pour i de 1 a $nb_inter faire{

      connecter_LIA /inter/ressort$i /agglo/masse$i @45154

}
```

L'utilisateur remarque ainsi qu'à chaque modification du nombre de masses de l'agglomérat, après exécution de la fonction "agglo", la fonction "inter" est invalidée en connexion, car les masses de l'agglomérat ont été supprimées pour être recréées. La simple exécution de la fonction "inter" suffit à la revalider et à obtenir la topologie attendue.

Changement du nombre de masses en interaction avec le pendule

Cette tâche peut se faire graphiquement en supprimant manuellement des interactions entre l'oscillateur et l'agglomérat. La modalité graphique est adaptée pour un faible nombre de suppressions sur des critères arbitraires ou spatiaux. Agir via la modalité graphique invalide la fonction "inter", il peut être intéressant pour l'utilisateur de relever les labels des interactions qu'il supprime pour insérer une commande du type suivant à la fin du script :

```
Supprimer_modules {/inter/ressort43 /inter/ressort67 /inter/ressort24}
```

Il peut alors exécuter de nouveau la fonction en obtenant le résultat voulu, tout en ayant des fonctions valides. Nous remarquons ici la difficulté de mise en œuvre de la collaboration verticale.

Pour supprimer des interactions de manière plus systématique, il peut aussi modifier la fonction "inter", soit en ajoutant des commandes de suppression, soit en créant moins d'interactions. C'est cette dernière option qui est réalisée dans le script ci-dessous, où seules les masses d'indice multiple de 3 sont mises en interaction avec le pendule.

```
%compter le nombre de masses de l'agglomérat

Affecter nb_inter [expr [compter_label MAS3D:/agglo/>]/3]

%Création et labellisation des interactions

labelliser [créer_module $nb_inter REF_3D] /inter/ressort1..

pour i de 1 a $nb_inter faire{

      connecter_LIA /inter/ressort$i /agglo/masse[expr $i*3] @45154

}
```

Substitution du type d'interaction dans l'agglomérat

L'utilisateur édite le script de la fonction "agglo" et change uniquement le type d'interaction. L'exécution de la fonction permet l'observation graphique du résultat, l'utilisateur reconnaît le code couleur du type d'interaction qu'il vient de définir. En revanche, les interactions entre le pendule et l'agglomérat semblent toujours connectés à l'agglomérat, ce n'est pas le cas, d'ailleurs la fonction "inter" indique que certaines de ses opérations de connexions ne sont plus valides. Ici encore, il suffit d'exécuter une nouvelle fois la fonction "inter" pour achever d'obtenir la topologie recherchée.

7.2.3. Scénario capsule

Construction du modèle

L'utilisateur crée une capsule sur l'espace topologique, ce type de module est accessible depuis la palette de modules. Il édite le script de la capsule, il est très semblable à celui de la fonction "agglo" vu précédemment :

```
Affecter nb_masses 200
%création et labellisation des nb_masses masses
Labelliser [créer_module $nb_masses MAS_3D] /agglo/masse1..
%création et labellisation des nb_masses*(nb_masses-1)/2 interactions
nécessaires
Labelliser [créer_module [expr $nb_masses*($nb_masses-1)/2] COH3_3D]
/agglo/cohesion1..
%connexion des interactions aux masses
affecter cur %indice de l'interaction à connecter
pour i de 1 a [expr $nb_masses – 1] faire{
      pour j=i+1 a $nb_masses faire{
            connecter_LIA /agglo/cohésion$cur /agglo/masse$i
/agglo/masse$j
            incr cur
      }
}
%Déclaration des LPC de surface
définir_LPC_surface tous MAS3D:/agglo/>
```

Après exécution de ce script, de retour sur l'espace topologique l'utilisateur peut observer la capsule et son label de liste de points de communication (LPC). On remarque notamment la représentation graphique de la LPC de surface sur la figure 66.

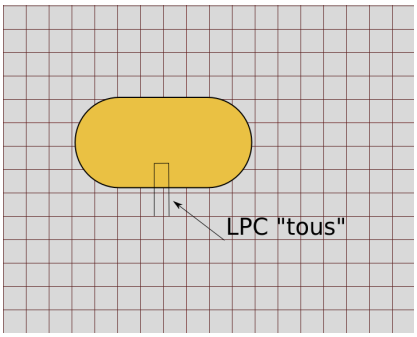


Figure 66 : La capsule et sa LPC sur l'espace topologique

Il crée l'oscillateur graphiquement et une deuxième capsule dont le script est le suivant:

```
Affecter nb_inter 200

%Création et labellisation des interactions

labelliser [créer_module $nb_inter REF_3D] /inter/ressort1..

%Définition des LPC de surface

définir_LPC_surface gauche /inter/ressort1..200 -> 1

définir_LPC_surface droite /inter/ressort1..200 -> 2
```

Après exécution de ce script, de retour sur l'espace topologique, l'utilisateur peut observer la deuxième capsule et ses deux LPC (cf. figure 67)

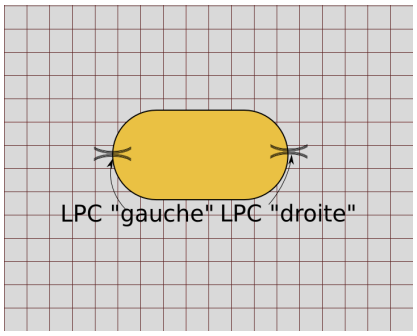


Figure 67 : Capsule des interactions entre l'agglomérat et le pendule

Il ne reste plus qu'à "tirer les câbles" depuis les LPC de la capsule d'interactions vers l'agglomérat d'une part et vers la masse libre de l'oscillateur d'autre part, comme illustré en figure 68. Ces deux actions graphiques provoquent la génération de deux fonctions dans l'espace dédié aux "fonctions de connexion de LPC". Comme aucun label n'a été défini, ni pour les capsules, ni pour le pendule, ces fonctions utilisent les labels auto[7].

[7] Nom absolu objectif d'un module (cf. §2.3)

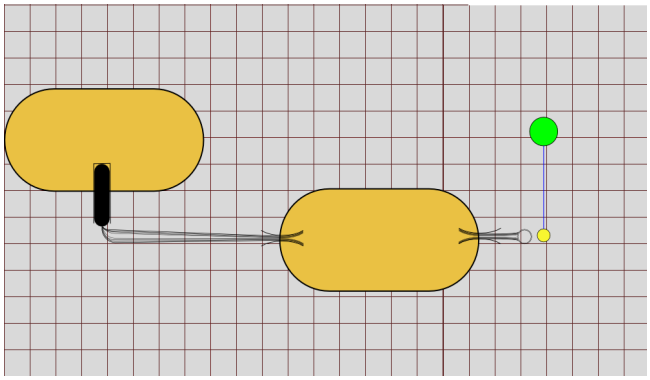


Figure 68 : Connexion des capsules

Changement du nombre de masses de l'agglomérat

L'utilisateur change la valeur de la variable `nb_masses` et exécute à nouveau le script de la capsule de l'agglomérat. La fonction dédiée à la connexion entre l'agglomérat et les interactions de ressort frottement est invalidée. Avant de l'exécuter à nouveau, il faut d'abord modifier la capsule contenant les interactions pour rectifier le nombre d'interactions. On ne peut pas, comme avec la fonction, mettre à jour automatiquement ce nombre car la capsule ne peut pas interroger l'extérieur. A la fin de l'opération, le résultat visuel sur l'espace topologique est le même quel que soit le nombre de masses de l'agglomérat.

Changement du nombre de masses en interaction avec le pendule

L'utilisateur édite le script de l'agglomérat et y ajoute les définitions de LPC suivantes:

```
Définir_LPC_surface partiel {/agglo/mas4 /agglo/mas12..20}
%Création d'une liste de labels contenant tous les labels d'indice pair
pour les masses de l'agglomérat
Affecter liste {}
Pour i=0 a [expr $nb_masses/2] faire{
      lappend liste /agglo/mas[expr $i*2]
}
Définir_LPC_surface pair $liste
```

Selon la LPC à laquelle il souhaite se connecter, l'utilisateur change le nombre d'interactions de la capsule d'interactions. Il achève son changement de topologie en "tirant les câbles" sur la LPC de son choix ou bien en éditant la fonction de connexion entre les interactions et l'agglomérat.

Changement du type d'interaction de l'agglomérat

La procédure est très similaire au scénario de la fonction. L'utilisateur édite le script de l'agglomérat, change le type de l'interaction et exécute le script. La fonction dédiée à la connexion entre l'agglomérat et les interactions est invalidée, car la capsule de l'agglomérat a été modifiée, il suffit de l'exécuter une nouvelle fois pour obtenir la topologie attendue.

7.2.4. Scénario macro-modules

Construction du modèle

A partir de la palette, l'utilisateur crée deux macro-modules : un macro-MAS et un macro-COH3. Avant de les disposer sur l'espace topologique, l'interface demande à l'utilisateur via une boîte de dialogue de préciser leur cardinalité –i.e. combien contiennent-ils respectivement de <MAS> et de <COH3>. Le champ de saisie accepte les commandes PNSL, l'utilisateur saisit donc 300 pour le macro-MAS et `[expr 300*299/2]` pour le macro-COH3.

Les deux macro-modules se présentent sous la forme suivante sur l'espace topologique (cf. figure 69), on remarque les LPC "tous", "gauche" et "droite" présents par défaut.

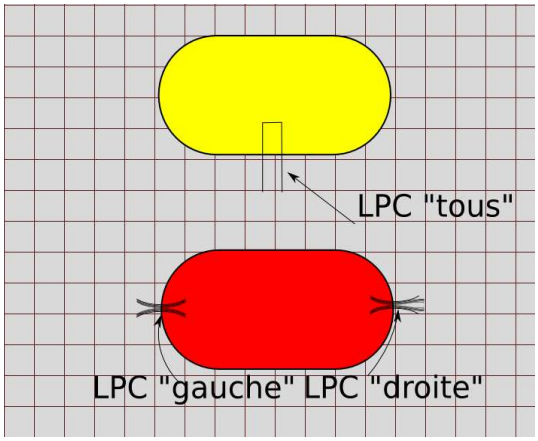


Figure 69 : Un macro-MAS (en haut) et un macro-COH3 (en bas) sur l'espace topologique

Pour définir un agglomérat, les LPC "gauche", "droite" et "tout" sont insuffisantes. L'utilisateur crée donc une fonction "lpc_agglo" pour définir les LPC nécessaires. On remarque dans le script suivant les difficultés importantes dans la création de cette LPC, qui requièrent des compétences en langage Tcl.

```
%Récupérer le nombre de points M du Macro-MAS

Affecter nbPtM [expr compter_ptM @1]

%créer une liste d'indice {2..nbPtM, 3..nbPtsM, …,NbPtM}

Affecter liste_gauche {}

Pour i=2 a $nbPtM faire{

lappend liste_gauche $i..$nbPtM

}

Définir_LPC agglo_gauche @1 $liste_gauche

%créer une liste

Affecter liste_droite {}

Pour i=1 a [expr $nbPtM -1] faire{
```

```
      lappend liste_droite [lrepeat [expr $nbPtM -$i] $i]

      }
Définir_LPC agglo_droite @1 $liste_droite
```

Après exécution de ce script, les deux nouveaux LPC apparaissent à la surface du macro-MAT, il suffit de tirer les câbles depuis les LPC "gauche" et "droite" pour achever la construction de l'agglomérat.

L'utilisateur construit graphiquement le pendule et crée une macro-REF de taille 300 pour le mettre en interaction avec l'agglomérat. Il achève sa construction en tirant les câbles du macro-REF vers les LPC "tous" du macro-MAS et vers la masse libre du pendule. Le résultat final est décrit en figure 70.

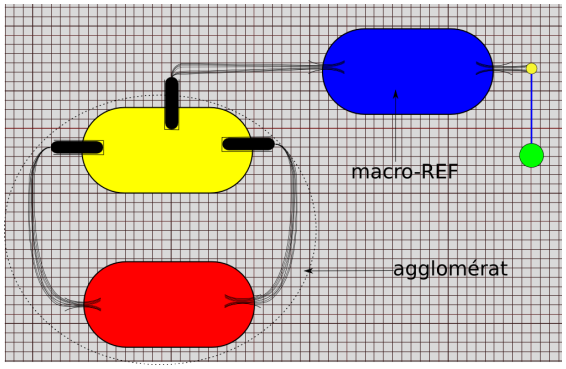


Figure 70 : Agglomérat en interaction avec un pendule, version macro-modules

Changement du nombre de masses de l'agglomérat

L'utilisateur accède aux propriétés du macro-MAS et change sa cardinalité. Il fait de même avec le macro-COH3, pour lequel il utilise une expression PNSL, comme pour sa création. La fonction de définition des LPC du macro-MAS est invalidée, il l'exécute pour mettre à jour les LPC `agglo_gauche` et `agglo_droite`. Il rétablit graphiquement les connexions entre le macro-MAS et le macro-COH3 d'une part et le macro-MAS et le macro-REF d'autre part.

Changement du type d'interaction de l'agglomérat

Il suffit de substituer le macro-COH3 par une macro-interaction du type voulu. Cette opération n'affecte pas les connexions car elle ne modifie pas la surface structurelle du macro-module. L'opération de substitution se fait graphiquement avec l'outil de substitution, qui permet de convertir la sélection vers le type de module désiré.

7.2.5. Analyse

Dans ce scénario exemplaire de collaboration horizontale, la fonction montre son intérêt en terme de simplicité d'utilisation et de souplesse. Il est aisé de faire des modifications. En revanche elle apparaît beaucoup moins adaptée pour la gestion de modèles de grande taille (le modèle présenté comptait environ 50 000 modules). Elle pâtit en effet d'une lourdeur au niveau graphique qui de plus n'est justifiée par aucun usage. Elle pâtit également d'une lourdeur au niveau du noyau : temps de chargement du projet

important, exécution des script relativement lente, d'où des modifications certes aisées mais peu rapides. Il faut aussi prendre en compte les conséquences du marquage des fonctions, c'est à dire la "surveillance" des modifications du réseau susceptibles d'invalider une fonction. Dernier inconvénient, les connexions des interactions de ressort frottements à l'agglomérat et au pendule ne peuvent être faites facilement par voie graphique.

La capsule résout le problème de la lourdeur graphique et donne la possibilité d'une connexion graphique, mais la surcharge du noyau demeure problématique, même si elle est relayée à un sous-projet. La capsule se révèle également relativement compliquée à mettre en œuvre au regard de la simplicité de son contenu. L'utilisation de macro-modules associée à l'utilisation de fonction nous paraît la solution optimale pour ce scénario précis. En effet, les macro-modules sont tout à fait adaptés à la construction d'un agglomérat, topologie dans laquelle tous les modules sont indifférenciés, ils permettent de décrire cette topologie sans surcharger le noyau, aussi bien au niveau de la gestion des modules qu'au niveau du système de labellisation. La mise au point des LPC nécessaires peut paraître un exercice compliqué, mais il est pédagogique, de même que l'écriture de l'algorithme de connexion de l'agglomérat dans les cas précédents. Par ailleurs, vue l'utilisation très fréquente de la topologie en agglomérat, l'utilisateur peut définir une procédure dédiée à la définition des LPC du macro-MAS, lui permettant ainsi de réutiliser son code dans d'autres projets.

7.3. Conception d'une structure complexe de grande taille

7.3.1. Description

Le modèle consiste en une topologie type drapeau dont on modifiera ensuite le motif de base, la manière de le répéter, la dimension du réseau final et enfin la topologie globale (plaque, tore, sphère). On s'attachera enfin à mettre ce sous-réseau en interaction avec le mât et le vent.

7.3.2. Etape 1 : obtenir différentes topologies avec peu de masses

L'utilisateur commence par construire graphiquement sa maille et définit de nombreux labels (figure 71). Ces labels permettent de lister les modules par groupe de trois selon leur orientation sur l'espace topologique, d'où l'utilisation des points cardinaux. Par exemple, le label /maille/EstNS/1 désigne le premier module de la liste de modules située à droite et parcourue de haut en bas.

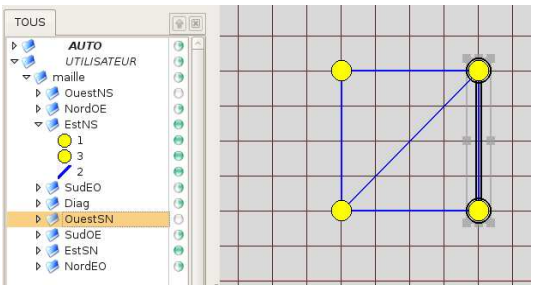


Figure 71 : Une maille et l'arbre de ses labels

Il crée ensuite une fonction "drapeau_replication_horizontale" avec laquelle il crée la première "ligne" du drapeau. On se réduit à une ligne de deux mailles pour tester sans être surchargé graphiquement, d'autant plus que le script ne gère pas le placement des modules.

```
Affecter nb_colonnes 2

%duplication

Pour i de 1 a $nb_colonnes faire{

      Dupliquer /maille/>> avec /maille dans /drapeau/ligne1/maille$i

}

%fusion

Pour i de 1 a [expr $nb_colonnes -1] faire{

      Fusionner /drapeau/ligne1/maille$i/EstSN/1..3
/drapeau/ligne1/maille[expr $i+1]/OuestSN/1..3

}
```

La figure 72 illustre un résultat intermédiaire, après l'opération de duplication et avant la fusion, les deux listes de modules entourées vont être fusionnées à l'étape suivante (figure 73).

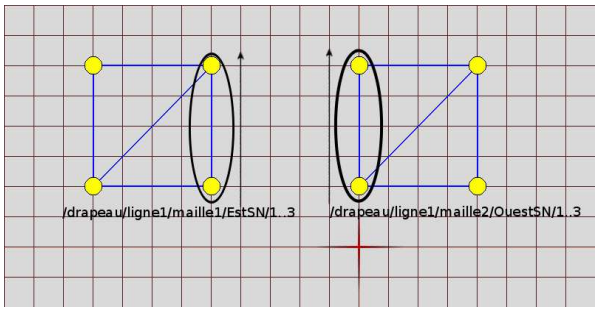


Figure 72 : Deux mailles avant fusion

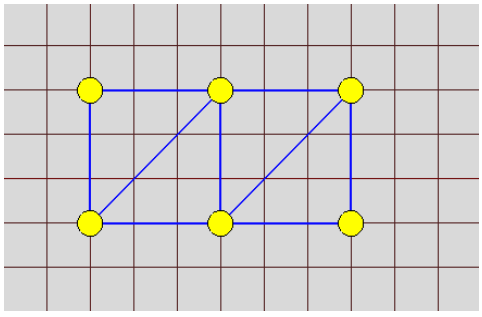


Figure 73 : Deux mailles fusionnées

Pour obtenir un drapeau de deux mailles sur deux, l'utilisateur crée une nouvelle fonction "drapeau_replication_verticale" avec le script suivant :

```
Affecter nb_colonnes 2
```

```
Affecter nb_lignes 2
%duplication
Pour i de 1 a $nb_lignes faire{
      Dupliquer /drapeau/ligne1/>> avec /drapeau/ligne1 dans
/drapeau/ligne$i
}
%fusion
Pour i de 1 a [expr $nb_lignes -1] faire{
Fusionner /drapeau/ligne$i/maille1..$nb_colonnes/SudOE/1..3
/drapeau/ligne[expr $i+1]/maille1..$nb_colonnes/NordOE/1..3
}
```

La figure 74 illustre un résultat intermédiaire, après l'opération de duplication et avant la fusion, les deux listes de modules entourées vont être fusionnées à l'étape suivante (figure 75).

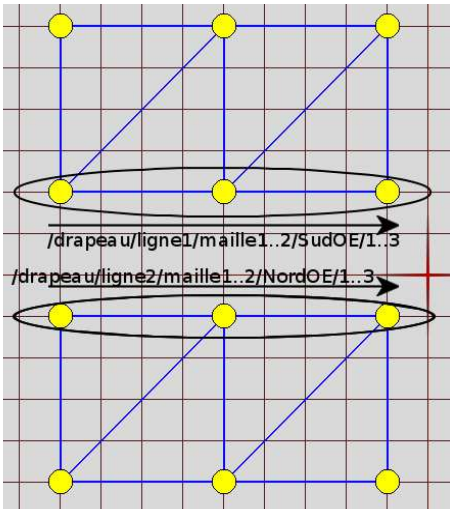


Figure 74 : Deux lignes avant fusion

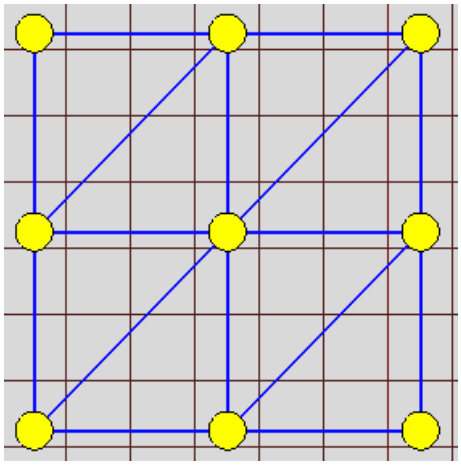


Figure 75 : Deux lignes fusionnées

Nous remarquons que les opérations de fusion prennent ici en argument 2 listes de labels qui comportent des doublons, c'est à dire que deux labels dans la même liste pointent vers le même module. Ces doublons ne sont pas gênants car ils sont situés aux mêmes endroits dans les deux listes, la deuxième occurrence est donc ignorée car elle sera interprétée comme la fusion du même module.

Cette manière de construire une topologie de drapeau permet d'imaginer de nombreuses variantes, soit en faisant varier le motif de départ, soit en faisant varier les processus de réplication horizontale et/ou verticale. L'exemple suivant montre une variante sur la réplication horizontale :

```
Affecter nb_colonnes 3
```

```
%duplication
Pour i de 1 a $nb_colonnes faire{
      Dupliquer /maille/>> avec /maille dans /drapeau/ligne1/maille$i
}
%fusion
Pour i de 1 a [expr $nb_colonnes -1] faire{
      Fusionner /drapeau/ligne1/maille$i/EstSN/1..3
/drapeau/ligne1/maille[expr $i+1]/OuestNS/1..3
}
```

La figure 76 montre le résultat intermédiaire après duplication, on remarque que la deuxième maille a subi une symétrie par rapport à son axe horizontal. Cette symétrie est la représentation graphique de la fusion de l'étape suivante entre deux listes de modules "orientés" en sens inverse.

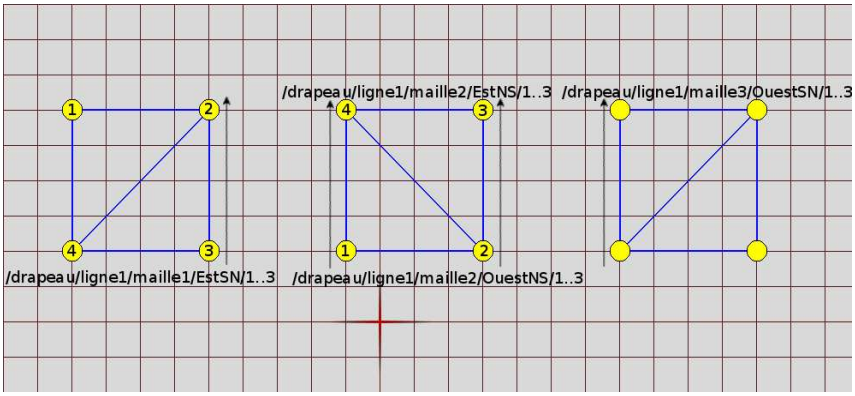


Figure 76 : Trois mailles du drapeau avant fusion

Pour obtenir un drapeau de trois mailles sur deux, l'utilisateur crée une nouvelle fonction "drapeau_replication_verticale2" avec le script suivant :

```
Affecter nb_colonnes 2
Affecter nb_lignes 3
%duplication
Pour i de 1 a $nb_lignes faire{
      Dupliquer /drapeau/ligne1/>> avec /drapeau/ligne1 dans
/drapeau/ligne$i
}
%fusion
Pour i de 1 a [expr $nb_lignes -1] faire{
            Pour j=1 a $nb_colonnes faire{
```

```
                    Si j%2 alors faire{
                    Fusionner /drapeau/ligne$i/maille$j/SudOE/1..3
/drapeau/ligne[expr $i+1]/maille$j/NordOE/1..3
                    }
              Sinon faire{
                    Fusionner /drapeau/ligne$i/maille$j/NordOE/1..3
/drapeau/ligne[expr $i+1]/maille$j/SudOE/1..3
                    }
      }
}
```

En effet la fonction "drapeau_replication_verticale" n'est plus valable, car une maille sur deux a subi une symétrie par rapport à son axe horizontal, interchangeant ainsi le "Nord" et le "Sud". Une autre solution aurait été de réorganiser les labels avant réplication verticale pour qu'ils correspondent à la représentation graphique des mailles.

7.3.3. Etape 2 : passer à des dimensions supérieures

Cette première étape a permis d'expérimenter des topologies diverses en travaillant à la fois sur le script et sur la représentation graphique. L'utilisateur peut à présent, maintenant qu'il a choisi le motif et son processus de réplication, chercher à obtenir un réseau de plus grande taille. Il peut continuer à travailler avec des fonctions et simplement augmenter le nombre de lignes et de colonnes, mais conserver une représentation graphique certainement plus lourde en raison du grand nombre de modules ne lui est plus forcément utile, de faibles dimensions ont suffi pour travailler sur la topologie. Il peut donc être intéressant d'encapsuler le drapeau en vue de l'étape 4. Le script de la capsule se construit en trois étapes : 1) copier la maille de départ depuis l'espace topologique, puis la coller dans le script. Cette opération génère un script reproduisant la maille et ses labels dans le projet de la capsule. 2) Copier le script de réplication horizontale de son choix 3) Copier le script de réplication verticale de son choix. L'exécution du script ainsi obtenu permet d'aboutir un drapeau encapsulé, dont il suffit de changer le nombre de colonnes et de lignes pour changer la dimension.

7.3.4. Etape 3 : mettre le drapeau en interaction avec le reste du réseau

Le drapeau est en interaction avec un mât fixe modélisé par un macro-SOL de cardinalité égale au nombre de lignes du drapeau plus un. Les interactions entre le mât et le drapeau sont des interactions de ressort-frottement. Différentes configurations sont possibles pour connecter ces interactions au mât et au drapeau. Nous en présentons deux, la première dite "directe" avec autant d'interactions que de <SOL> dans le mât (cf. figure 77), la deuxième dite "croisée" avec deux fois plus d'interactions que de lignes dans le drapeau (cf. figure 78). Chacune nécessite la définition de LPC adaptées pour le macro-SOL et pour le drapeau encapsulé.

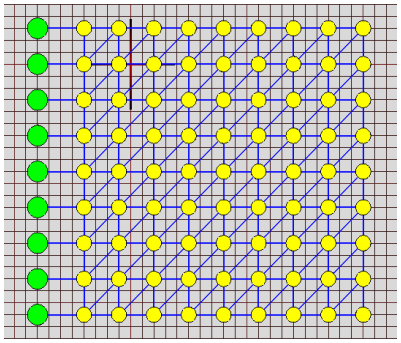


Figure 77 : Connexion drapeau mât "directe"

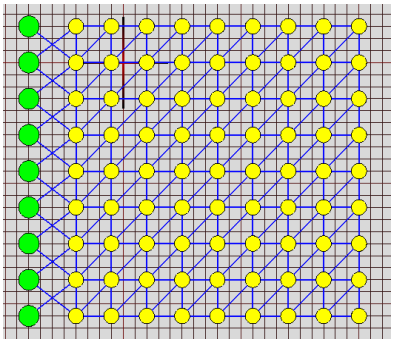


Figure 78 : Connexion drapeau mât "croisée"

Un macro-REF de cardinalité identique connecte le macro-SOL au drapeau. Il faut donc déclarer une LPC adaptée, selon le type de connexion que l'on souhaite établir avec le mât. Le script de la capsule s'achève donc avec la déclaration de deux LPC pour deux types de connexion différents :

```
Définir_LPC_surface Ouest {/drapeau/ligne1..$nb_lignes/maille1/OuestNS/1
/drapeau/ligne$nb_lignes/maille1/OuestNS/3}
Définir_LPC_surface OuestCroise
<MAT>:/drapeau/ligne1..$nb_lignes/maille1/OuestSN/1..3
```

Pour la configuration "directe", la LPC par défaut du mât convient, en revanche pour la croisée, il faut la modifier puisque tous les <SOL> "centraux" sont connecté à deux points L :

```
Affecter NbSol [compter_PtM /mat]
Affecter liste {1}
Pour i de 2 a [expr $NbSol-1] faire{
      lappend liste $i $i
}
lappend liste $NbSol
Définir_LPC /mat croise $liste
```

L'utilisateur connecte graphiquement la "gauche" du macro-REF à "tous" le macro-SOL, puis la "droite" du macro-REF à "l'Ouest" ou bien à "l'OuestCroisé" du drapeau.

7.3.5. Etape 4 : changer la topologie globale du drapeau

Nous allons à présent montrer comment il est possible de passer d'une topologie "plan" à des topologies "cylindre", "tore" et "sphère". Notons que ces noms réfèrent à des topologies spatiales mais que cette terminologie n'est pas rigoureusement correcte pour qualifier des réseaux masses-interactions. En effet ceux-ci doivent être considérés

comme des topologies discrètes, hors espace, nous faisons l'analogie avec des topologies spatiales car elle est aisément compréhensible et qu'il n'y pas de mots équivalents, à notre connaissance, pour décrire des topologies discrètes.

Passage du "plan" au "cylindre", puis du "cylindre" au "tore"

Pour passer d'un plan à un cylindre, il faut joindre deux côtés du plan, comme lorsqu'on joint les deux côtés d'une feuille de papier. Ceci revient donc, avec nos outils et en considérant une topologie réseau et non plus spatiale, à fusionner les "côtés" du drapeau. L'utilisateur va donc modifier le script de la capsule "drapeau" en ajoutant à la fin la commande de fusion suivante, qui fusionne le "côté Ouest" avec le "côté Est" :

```
Fusionner /drapeau/ligne1..$nb_lignes/maille1/OuestNS/1..3
/drapeau/ligne1..$nb_lignes/maille$nb_colonnes/EstNS/1..3
```

Le problème qui se pose alors est le suivant : comment vérifier visuellement que la topologie obtenue est bien celle recherchée ? La capsule permet de visualiser la topologie dans son espace topologique dédié mais il reste à placer les modules <MAT> de manière judicieuse sur cet espace. La bidimensionnalité de cet espace semble alors poser problème, en effet la topologie cylindrique est une topologie tridimensionnelle. Cependant, nous cherchons à visualiser une topologie de *réseau* cylindrique, on cherche donc uniquement à vérifier que les "côtés" de la plaque ont été correctement fusionnés. Le placement illustré en figure 79 et 80 permet d'effectuer cette vérification.

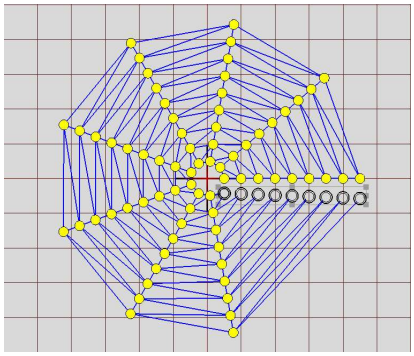


Figure 79 : Drapeau pendant l'opération de fusion

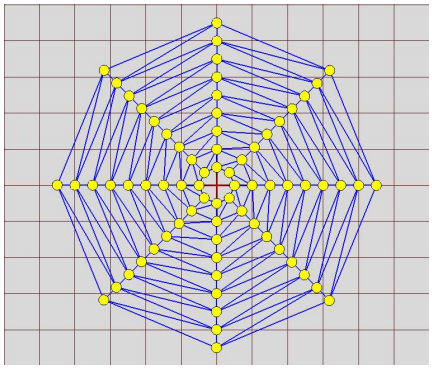


Figure 80 : Topologie cylindrique après fusion

Le passage du cylindre au tore se fait de manière similaire mais cette fois une unique visualisation 2D est difficilement envisageable, cependant l'objectif étant toujours le même (vérifier la fusion de deux côtés), il peut être atteint en utilisant deux visualisations correspondant à deux placements différents des modules <MAT>, comme illustré en figures 81 et 82. Dans la première vue, les "colonnes" sont disposées en cercles concentriques, alors que dans la deuxième vue, ce sont les lignes qui sont disposées en cercles concentriques. Dans les deux cas, on observe que les deux "extrémités" sont correctement fusionnées.

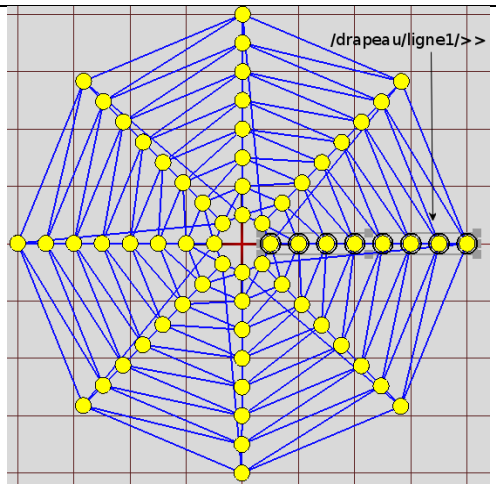


Figure 81 : Vue n°1 de la topologie tore

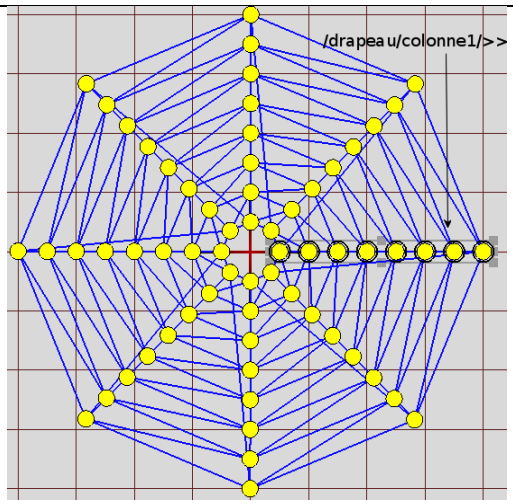


Figure 82 : Vue n°2 de la topologie tore

Nous avons montré avec cet exemple que la vérification visuelle d'une topologie devait être guidée par l'objectif et non par l'analogie avec la topologie spatiale. En effet, ne pas se détacher de l'analogie spatiale amènerait l'utilisateur à exiger un espace topologique tridimensionnel, non seulement inutile, mais trompeur quant à la signification de la topologie du réseau.

Passage du "cylindre" à la "sphère"

Du point de vue spatial, passer d'un cylindre à une sphère peut être vu comme la réduction de chacune des deux sections du cylindre à un unique point. Du point de vue de la topologie réseau, cette opération est donc équivalente à fusionner successivement toutes les paires de modules <MAT> des deux "sections" du "cylindre" jusqu'à obtenir un unique <MAT> sur chaque "section". La portion de script suivante réalise cette opération, elle est inséré à la fin du script du "drapeau cylindre".

```
Pour i de 1 a $nb_colonnes faire{

     Fusionner /drapeau/ligne1/maille$i/1 /drapeau/ligne1/maille$i/3

Fusionner /drapeau/ligne$nb_lignes/maille$i/1

/drapeau/ligne$nb_lignes/maille$i/3

}
```

La visualisation de la topologie pose le même problème que pour le tore et peut trouver la même solution, en utilisant deux visualisations complémentaires.

7.4. Modèle à deux couches : topologie agglomérat sur topologie graphe de Heawood

Ce dernier modèle est totalement abstrait, ce n'est pas un modèle d'un phénomène particulier mais davantage un essai de topologie complexe avec un grand nombre de modules. Le modèle de véhicule sur sol meuble était un premier exemple mais il est construit et pensé comme des objets en interaction. Or nous voulons montrer qu'il est possible de construire des topologies en dehors d'une pensée objet. Notre modèle est donc un modèle par couche : la première couche est constituée d'un agglomérat, la deuxième couche est constituée des masses de l'agglomérat regroupée en 14 paquets de même taille reliés par des interactions afin de former un graphe de Heawood (voir figure 83).

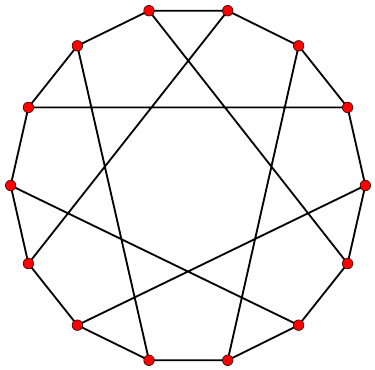


Figure 83 : Graphe de Heawood

Nous avons choisi de créer cette topologie à l'aide d'une unique fonction, décrite ci dessous.

```
%Création et labellisation de l'agglomérat

Labelliser [créer_macro_module 280 MAS_3D] /agglo/masses

Labelliser [créer_macro_module [expr 280*279/2] COH3_3D]
/agglo/cohésions

Connecter_PtL /agglo/masses -> aggloGauche /agglo/cohésions ->gauche

Connecter_Ptl /agglo/masses -> aggloDroite /agglo/cohésions ->droite

%Création de la topologie en graphe de Heawood

%1)Calcul du nombre d'interactions entre chaque paquet de masse

Affecter nbCons [expr (280/14)*(280/14)]

%2)Création et labellisation des interactions

Labelliser [Créer_macro_module [expr 14*$nbCons] REP4_3D
/Heawood/InterAnneau
```

```
Labelliser [Créer_macro_module [expr 7*$nbCons] REP4_3D
/Heawood/InterCroise
%3)Deuxième labellisation du macro-MAS
Labelliser /agglo/masses /Heawood/masses
%4)Déclaration des LPC pour préparer les connexions
Pour i de 1 a 14 faire {
Definir_LPC /Heawood/masses Heawood$i [lrepeat [expr ($i-1)*20 +
1]..[expr $i*20] 20]
Definir_LPC /Heawood/InterAnneau gauche$i [pair [expr ($i – 1)*$nbCons
+ 1]..[expr $i*$nbCons]]
Definir_LPC /Heawood/InterAnneau droite$i [impair [expr ($i –
1)*$nbCons + 1]..[expr $i*$nbCons]]
}
Pour i de 1 a 7 faire {
Definir_LPC /Heawood/InterCroise gauche$i [pair [expr ($i – 1)*$nbCons
+ 1]..[expr $i*$nbCons]]
Definir_LPC /Heawood/InterCroise droite$i [impair [expr ($i –
1)*$nbCons + 1]..[expr $i*$nbCons]]
}
%5)Connexions "anneau"
Pour i de 1 a 13 faire {
    Connecter_ptL /Heawood/InterAnneau -> gauche$i /Heawood/masses ->
Heawood$i
    Connecter_ptL /Heawood/InterAnneau -> droite$i /Heawood/masses ->
Heawood[expr ($i + 1)]
}
Connecter_ptL /Heawood/InterAnneau -> gauche$i /Heawood/masses ->
Heawood14
Connecter_ptL /Heawood/InterAnneau -> droite$i /Heawood/masses ->
Heawood1
%6)Connexions "croisées"
Pour i de 1 a 3 faire {
    %Les indices pairs sont connectés " 5 en avant"
    Connecter_ptL /Heawood/InterCroise -> gauche[expr 2*$i]
/Heawood/masses -> Heawood[expr 2*$i]
```

```
     Connecter_ptL /Heawood/InterCroise -> droite[expr 2*$i]
/Heawood/masses -> Heawood[expr 2*$i + 5]
     %Les indices impairs sont connectés "5 en arrière"
     Connecter_ptL /Heawood/InterCroise -> gauche[expr 2*$i – 1]
/Heawood/masses -> Heawood[expr 2*$i – 1]
     Connecter_ptL /Heawood/InterCroise -> droite[expr 2*$i + 1]
/Heawood/masses -> Heawood[expr 10 + 2*($i-1)] %(2*$i – 1 – 5)%14
}
 Connecter_ptL /Heawood/InterCroise -> gauche7 /Heawood/masses ->
Heawood8
 Connecter_ptL /Heawood/InterCroise -> droite7 /Heawood/masses ->
Heawood13
```

Nous constatons que ce script PNSL est long et complexe. Il ne remplit manifestement pas les critères d'accessibilité requis. La principale raison à ce défaut est que la moitié du code est consacré à l'établissement des connexions pour la topologie graphe de Heawood, alors que déjà un quart du code était consacré à la préparation de ces connexions. Pourtant ces connexions sont en nombre relativement réduit (21 groupes d'interactions donc 42 listes de points L à connecter), et il serait plus "naturel" d'effectuer ces connexions graphiquement. On se heurte alors à un double problème de représentation que nous n'avions jamais rencontré auparavant : 1) la représentation adéquate pour connecter la topologie graphe de Heawood (cf. figure 79) est incompatible avec la représentation monobloc des macro-modules 2) la représentation de l'agglomérat rentre en conflit avec la représentation nécessaire pour le graphe de Heawood. On pourrait objecter au premier problème que le modélisateur a mal préparé son modèle et aurait dû créer plusieurs macro-MAS et plusieurs <N-REP4>, mais ce serait à tort déplacer le problème de la représentation vers la modélisation. En effet, les modules <MAS> et <REP4> encapsulés dans ces macro-modules sont tous identiques du point de vue topologique. Il n'y a pas de raison, du point de vue de la modélisation, de les différencier. De plus, la définition de l'agglomérat s'en retrouverait considérablement complexifiée.

Nous n'apportons pas de solution clairement spécifiée à ce problème. En revanche nous mettons l'accent sur sa spécificité, plus particulièrement sur sa nature : un problème de représentation. Nous donnons cependant deux pistes concernant les deux aspects de ce problème. La première partie du problème peut trouver une solution dans la notion d' "alias graphique", c'est-à-dire qu'un même module peut avoir plusieurs représentations graphiques distinctes sur l'espace topologique. Dans notre cas, le <N-MAS> aurait 14 alias graphiques, chacun exhibant une LPC différente, de même pour les <N-REP4>. La deuxième partie du problème est un problème de surcharge graphique, l'utilisateur est submergé par des représentations qui lui sont inutiles et gênantes par rapport à la tâche qu'il a accomplir. Ce problème est bien connu dans les logiciels de dessin vectoriel et de traitement d'images. La solution est elle aussi largement répandue : c'est le système des calques, qui permet de rendre visible uniquement les objets sur lesquels on travaille.

Dans MIMESIS V, ce concept se traduirait par la superposition de plusieurs espaces topologiques, chacun ayant sa représentation, partielle ou non, du réseau.

7.5. Conclusion

Nous avons développé trois exemples mettant en jeu des topologies diverses, chacune exposant ses problèmes propres. Nous nous sommes efforcés de mettre en œuvre différentes solutions, en particulier dans le premier exemple, pour mettre en évidence leurs avantages et inconvénients. Nous avons ainsi pu vérifier la facilité d'utilisation des fonctions mais leur lourdeur pour les modèles de grande taille. D'autres exemples, basés sur des modèles existants, auraient pu être détaillés dans cette partie, notamment le véhicule sur sol meuble, qui met en jeu une topologie hétérogène et complexe. Mais notre second objectif, après la validation des différents outils proposés dans les parties 2 et 5, était de mettre en avant différents éléments méthodologiques. Nous avons ainsi pu voir l'intérêt d'introduire des "paramètres topologiques", c'est à dire des variables d'un script PNSL contrôlant la topologie, où encore la définition de label pour rendre accessible des modules à la modalité textuelle. Pour ce dernier point, on retrouve d'ailleurs la primitive de transfert (une des cinq primitives TYCOON, cf §3.1), le transfert étant rendu possible par la labellisation. Nous notons également l'importance d'un minimum de maitrise du langage Tcl : combinaison de procédures avec l'opérateur [], création de listes complexes. Ce point est à prendre en considération dans la pédagogie, et il sera certainement nécessaire de définir des procédures plus accessibles pour le traitement de liste, comme préconisé au §6.3.

Le dernier exemple est assez atypique, car ce genre de topologie n'a jamais été mis en œuvre. Il montre les limites des outils que nous proposons, notamment au niveau des représentations graphiques. MIMESIS V devra donc encore évoluer à l'avenir pour gérer de la manière la plus adéquate des topologies toujours plus complexes. Cependant, ces évolutions ne doivent pas remettre en cause les bases solides que nous avons proposées, ni surcharger inutilement l'interface. Le bilan des complexités topologiques atteignables avec MIMESIS V est déjà très satisfaisant, de nombreuses stratégies sont imaginables. Les quelques unes exposées dans cette partie ne sont certainement qu'un échantillon. Il appartient aux utilisateurs de créer les leurs et de réadapter celles que nous avons proposées à leur mode de pensée.

8. Conclusion

Le chapitre II définissait une extension de CORDIS-ANIMA pour aller vers plus de complexités en introduisant des non-linéarités. Pour tirer parti de la puissance de CORDIS-ANIMA, avec ou sans contrôle en ligne de paramètres, un environnement interactif complet est indispensable. Cette interface, MIMESIS V, doit satisfaire un certain nombre de contraintes pour donner à l'utilisateur les moyens d'atteindre des niveaux importants de complexité. La première des conditions est de disposer d'un langage de script puissant, permettant d'agir sur toutes les phases du processus de modélisation. PNSL, Physical Network Scripting Language, a donc été conçu dans cet objectif et dans le cadre plus général de la suite logicielle pour la multisensorialité. Nous avons donc été confrontés à la problématique de la collaboration entre cette modalité, le script, et la modalité graphique. En effet le script n'est pas optionnel comme dans la plupart des interfaces WIMP, c'est une modalité incontournable, particulièrement dans les modèles de grande taille. Nous avons donc proposé une panoplie d'outils orientés vers l'exploration de la complexité, à même de gérer un très grand nombre de modules et/ou un très grand nombre de points de communication. Il fallait définir un ciment entre ces outils, les uns purement graphiques, d'autres définis pour le langage de script, pour mettre en œuvre dans les meilleures conditions la collaboration entres les deux modalités. Nous avons défini rigoureusement ces conditions et proposé deux outils complémentaires, la fonction et la capsule. Nous avons organisé MIMESIS en cinq temps, pour chacune des cinq phases. La phase de conditions initiales est une phase très sensible dans le processus de modélisation. Les outils que nous avons définis sont une bonne base pour aborder ce problème complexe dans lequel intervient la relation entre physique et géométrie, parmi ces outils PNSL bien sûr, mais aussi les objets spatiaux, qui comptent parmi les abstractions principales de MIMESIS V, et dont le statut reste à préciser. Pour donner au modélisateur la possibilité de travailler finement sur cette phase, il est nécessaire d'apporter une réflexion supplémentaire sur les liens entre physique et géométrie, et d'étudier quels ponts sont possibles entre les deux univers. C'est le sujet du prochain chapitre.

Chapitre IV. Conditions initiales et spatialité dans la modélisation physique : un enjeu pour les modèles complexes

1. Enjeux liés aux conditions initiales

1.1. Enoncé du problème

Les systèmes dynamiques tels qu'ils sont modélisables par les réseaux CORDIS masses – interactions, sont par principe des systèmes sensibles aux conditions initiales. C'est pourquoi dans les logiciels de modélisation tels que MIMESIS, développés par le laboratoire ICA, leurs données et leur manipulation par l'utilisateur qui effectue le modèle fait l'objet d'une phase dédiée dans le processus de modélisation.

Rappelons que le processus de modélisation donné à pratiquer à l'utilisateur est structuré en 5 phases bien clairement définies : PSQL pour la topologie du réseau, PSQN pour ses paramètres, CI pour les conditions initiales, et enfin les phases d'habillage et de simulation. Matthieu Evrard s'est attaché dans sa thèse plus spécifiquement à certaines questions de la phase PSQL, en particulier celles liées à la l'introduction de non-linéarités dans les interactions physiques. Nous avons abordé dans le Chapitre II, l'augmentation de la complexité en nous intéressant à la modification en ligne des paramètres physiques. Cette modification introduisant un second niveau de non-linéarité dans les comportements simulés.

Nous abordons dans ce chapitre la question critique des conditions initiales. Jusqu'à ce jour, les valeurs initiales manipulées étaient sommaires d'une part parce qu'elles étaient en assez petit nombre et d'autre part parce que leur manipulation s'effectuait à la main une à une pour chacun des modules, hormis quelques fonctionnalités plus globales mais néanmoins triviales comme le positionnement sur des lignes ou dans des boites englobantes, dans leur manipulation.

Le passage à des modèles de grandes tailles et pouvant présenter des niveaux de structures très différents (des réseaux très homogènes et de très grande taille à des réseaux plus hétérogènes ou homogènes par parties) conduit à examiner spécifiquement la question de la modélisation des conditions initiales.

Usuellement, les modèles physiques masses-interactions étant des combinaisons de systèmes différentiels du second ordre, les conditions initiales sont de deux types, telles que par exemple les positions et les vitesses à l'instant t=0.

Nous discuterons très brièvement de la question des vitesses en conclusion de ce chapitre. Nous nous sommes essentiellement focalisés sur la question des positions initiales.

Le passage à la complexité suppose de pouvoir donner de manière aussi élégante et manipulable que possible et au grain souhaité par l'utilisateur, au plus simple la totalité des positions des modules de type <MAT>. Ce procédé, n'est simple qu'en apparence puisqu'il soulève plusieurs questions difficiles et couplées :

1. La question du grand nombre de masses couplées entre elles par des interactions non nécessairement linéaires ou à plusieurs états d'équilibre,

2. La question de la relation entre la phase "Conditions initiales" et la phase PSQN "préstructuration quantitative", question d'autant plus difficile que le nombre de <MAT> ou de <LIA> est grand et que les interactions sont complexes.

3. La question de la précision des valeurs.

Ainsi, un procédé autre que de donner les valeurs initiales de tous les <MAT>, et qui peut avoir son intérêt également, serait de pouvoir donner certaines longueurs au repos à des modules <LIA>. Celles-ci s'effectuent dans la phase PSQN. Mais surtout, le problème de la détermination univoque des positions initiales des masses qu'ils relient reste entier. Par conséquent, la donnée des positions initiales de tous les modules <MAT> reste, en tout cas pour le moment, le moyen le plus simple.

Néanmoins, l'utilisateur ne peut connaître véritablement l'état initial de son système qu'en comparant les longueurs initiales des modules <LIA> telles qu'elles se déduisent des positions des <MAT> qu'ils relient avec les longueurs au repos de ces modules <LIA>. Ainsi, maîtriser l'état initial du système dynamique requiert d'avoir la possibilité d'effectuer des allers-retours entre les phases "Conditions Initiales" et "Préstructuration Quantitative" et que le logiciel permette aisément de pouvoir comparer ces valeurs, une à une ou par partie de réseau. Dans MIMESIS 4, la seule information dont dispose l'utilisateur est une information locale dans les modules <LIA> où apparaît à titre informatif, la valeur de la longueur initiale aux côtés des valeurs des paramètres (élasticités, viscosités, seuils) du module.

D'une manière plus générale, la spécification de l'état initial du système dynamique représenté par un réseau CORDIS-ANIMA, suppose d'agir simultanément sur les positions initiales et sur les paramètres spatiaux des modules d'interaction. En effet, les modules <MAT> sont nécessairement connectés entre eux par des interactions et par conséquent, les positions initiales doivent être en cohérence avec les paramètres spatiaux de ces interactions pour obtenir l'état initial voulu du système. Or les paramètres spatiaux des modules <LIA> se définissent de manière relative puisqu'il s'agit usuellement de valeurs de distance alors que les positions initiales des modules <MAT> effectuent un ancrage spatial absolu. Il serait par ailleurs tout à fait utopique et impossible pour des réseaux complexes et de topologie physique quelconque de travailler les positions des modules <MAT> et les paramètres spatiaux des modules <LIA> de manière totalement relative.

Ce problème n'apparaît pas tant que le nombre de masses manipulées est assez petit pour le résoudre à la main. Il n'apparaît pas non plus lorsque les interactions sont simples et linéaires et qu'elles définissent des topologies régulières comme dans le cas des modèles dits masses-ressorts. Dans ces modèles en effet, les interactions sont usuellement des interactions élastiques linéaires à un seul état de repos ; la topologie physique suit une structure de maillage, les interactions principales se situant

majoritairement sur les arêtes d'un maillage d'un objet géométrique ; l'état initial recherché est la plupart du temps un état d'équilibre. Cet état d'équilibre se déduit assez facilement de la géométrie de l'objet. Dans des cas de maillages un peu complexes, une présimulation physique, comme proposée par Jimenez [Jim93] permet d'obtenir les positions et longueurs initiales correspondant à l'état d'équilibre.

Dans notre cas où les topologies physiques peuvent être quelconques et ne représentent que rarement des structures du type maillages, des proximités ou des contigüités, où les interactions peuvent être non linéaires en distance (ainsi qu'en vitesse, mais la non linéarité en vitesse n'est pas concernée ici) et où elles peuvent présenter plusieurs état d'équilibre, la question n'est plus triviale.

Ces points sont d'autant plus critiques qu'un état dynamique peut être très différent d'un autre à des valeurs initiales extrêmement proches. Une différence de la résolution du codage des nombres entre une longueur au repos d'une interaction et sa longueur initiale peut faire bifurquer le système dynamique d'un état dans un autre, d'un état d'immobilité à un mouvement de déformation ou de déplacement, d'un état d'équilibre à un état chaotique, chacun devant pouvoir être choisi par l'utilisateur. La finesse d'ajustement est donc un point essentiel pour le modélisateur dans la définition de l'état initial de son modèle.

Le problème fondamental qui réapparaît derrière une question qui aurait pu sembler simplement technique est celui du statut de l'espace et de la géométrie dans les réseaux masses-interactions. Le paragraphe suivant « Relation forme –Mouvement » revient sur une base fondamentale pour ces réseaux :

La spatialité physique se définit dans les interactions de manière relative et à l'inverse de manière absolue dans les éléments matériels.

1.2. Relation forme-mouvement

Nous avons abordé dans le chapitre introductif la triple nature de l'image, morphologique, optique et dynamique. La distinction phénoménologique entre ces trois composantes d'une image sous-tend une distinction également au niveau de la cause du phénomène. Nous ne disserterons pas sur les causes "réelles" d'une image, et de leur partition également en trois composantes. En revanche, si nous revenons plus simplement au domaine de la synthèse d'image, nous pouvons constater, comme nous l'avons déjà fait dans ce manuscrit, l'existence de deux approches. La première approche, classique en animation, établit des modèles géométriques sur lesquelles elle va appliquer un mouvement, à l'aide d'un modèle descriptif ou générateur. La deuxième approche, initiée par Annie Luciani depuis bientôt 30 ans, consiste à inverser cette préséance de la forme et à concevoir le mouvement pour lui même en s'abstrayant de toute vision figurative pour privilégier une vision dynamique. Dans cette approche, la forme arrive après le mouvement, pour le révéler, pour l'explorer de différentes manières. La forme est donc au centre des attentions du modélisateur dans la dernière étape du processus de modélisation, l'habillage. Cependant, la forme, et la géométrie de manière plus générale, interviennent avant dans le processus de modélisation. Les conditions initiales obligent à définir une spatialité absolue, alors que les paramètres spatiaux des interactions obligent à définir une spatialité relative. Ces données quantitatives ne sont

bien sûr pas sans rapport avec la géométrie, qui intervient donc au cœur du processus de modélisation, et pas seulement à la fin. Il existe donc un réel problème dans la définition de ces données, et ce problème de fond a trait à la relation complexe qu'entretiennent forme et mouvement. Comment gérer l'intervention de la géométrie dans le processus de modélisation sans s'orienter vers une vision figurative du réseau, comme c'est le cas dans les systèmes masses-ressorts ? C'est à cette question complexe que nous allons tenter de répondre, qui ne constitue qu'une partie de la problématique de la relation forme-mouvement.

1.3. Ciblage de l'objet d'étude

Partant de la question de définir l'état initial d'un modèle physique, nous avons abouti à la question centrale de l'introduction de la spatialité dans les réseaux masses-interactions et de la définition nécessairement ambiguë des lieux où cette spatialité s'introduit : positions, distances phénoménologiques (distance entre deux positions initiales par exemple), distances multiples prédéterminées dans les fonctions d'interaction.

On retrouve ici la dualité entre la démarche partant de critères spatiaux définissant les formes des objets, i.e. dans laquelle la forme est la propriété des objets, et une démarche partant d'inerties ponctuelles, donc amorphes, en interaction dans laquelle la spatialité est implicitement codée dans l'interaction, i.e. dans laquelle la forme émerge de propriétés de fonctions d'interaction.

La première est usuelle en synthèse d'image où ce sont les objets qui portent des formes et où les interactions sont majoritairement des interactions de collisions portées par des procédés qui ne contiennent pas de notions spatiales, telles que la méthode de pénalité ou le multiplicateur de Lagrange. La deuxième est structurellement inhérente à la philosophie particulaire dans laquelle la représentation de toute scène en points a fait disparaître toute forme et toute structure strictement et statiquement spatiale entre les points.

Le problème de la spécification de l'état initial d'un système dynamique représenté par les paramètres physiques et variables initiales d'état d'un réseau masses-interactions est donc complexe.

Notre travail a d'abord consisté à essayer de reposer clairement quelques prérequis théoriques dans les paragraphes précédents, dans le contexte d'une augmentation de la complexité des modèles, en taille et en fonctions. Nous ne pouvons néanmoins avoir la prétention d'avoir posé d'un bloc tous les tenants et aboutissants théoriques liés à la difficulté de l'introduction de la géométrie dans les réseaux masses–interactions. Certains réapparaîtront parfois au fil des paragraphes suivants.

De manière à avancer dans la résolution à plus long terme de cette question, nous avons été amenés à restreindre nos ambitions. Pour cela, nous avons élaboré une typologie permettant de catégoriser quelques situations importantes de manière opérationnelle. On peut remarquer en effet que trois cas importants peuvent se présenter de manière assez fréquente :

Cas 1 : La scène physique, ou des parties de la scène, peut être constituée d'éléments matériels <MAS> ou <SOL>, qui peuvent présenter une organisation spatiale initiale

assez simple comme d'être positionnés sur une ligne ou sur un plan. Il serait alors intéressant d'introduire au sein même de MIMESIS, des outils de modélisation de certaines de ces organisations spatiales, parmi les plus fréquentes, sachant que le cas général est complexe. Deux remarques importantes :

- Ces placements sont suffisamment simples, non seulement parce qu'ils convoquent des organisations spatiales de base mais également car la relation avec les paramètres physiques des interactions qui concernent ces <MAS> et ces <SOL> pourra être laissé à la charge manuelle du modélisateur.

- Nous attirons ici l'attention sur le fait qu'il ne faut pas confondre les placements initiaux et la modélisation de contraintes géométriques qui devraient être satisfaites au cours de la simulation, comme « rester sur une ligne ou dans un plan ». Les organisations spatiales ainsi définies comme conditions initiales, comme « être alignés » par exemple, n'ont pas vocation à être maintenues au cours de la simulation dynamique (voir plus loin le cas 3).

Cas 2 : La scène physique, ou des parties de la scène, peut présenter une organisation initiale spatiale complexe, qui nécessite des outils de placement trop complexes pour être intégrés dans MIMESIS, tels que ceux disponibles par nature dans d'autres types de logiciels, en particulier des logiciels de modélisation géométrique. Dans ce cas, la démarche optimale consiste à importer des données géométriques, valeurs et/ou organisation, de modèles externes et de voir comment les transformer en positions initiales d'éléments matériels <MAS> ou de <SOL> ainsi qu'en données intervenant dans les paramètres des interactions sur lesquelles sont connectées ces <MAT> et <LIA>.

Ici encore, il ne s'agit que de fonctionnalités permettant de spécifier des conditions initiales et non des contraintes géométriques dans le modèle lui-même.

Le cas général étant assez complexe, un cas particulier mais très fréquent est celui où la scène peut se structurer en deux parties fonctionnelles : un ensemble d'éléments fixes <SOL> présentant des organisations spatiales complexes en interaction avec un ensemble d'éléments mobiles de type <MAS> en interaction physique avec ces <SOL>. Les éléments peuvent être alors modélisés par des ensembles de <SOL>, entre lesquels il n'y a de ce fait pas d'interaction, pouvant présenter une organisation spatiale aussi bien en termes topologiques que géométriques. Cette situation est plus simple que la situation générale qui concernerait tous types de modules matériels <SOL> et <MAS> dans la mesure où il n'y a pas de fonction d'interaction entre les <SOL>. La donnée des valeurs géométriques doit se faire de concert avec la détermination des paramètres spatiaux des interactions de ces <SOL> avec les éléments mobiles <MAS> de la scène avec lesquels ils interagissent. Nous appellerons ces ensembles de <SOL> spatialement structurés des "Obstacles". Ce vocable se justifie par le fait que les obstacles ne sont pas influencés par les mouvements des <MAS>. Ils se comportent comme des masses de valeur infinie. Un cas particulier est celui où ces <SOL> sont mobiles car en provenance d'une simulation amont via un fichier d'évolution de positions, d'une entrée gestuelle par capteur ou tout autre processus purement cinématique, via des modules <GP> pour "Générateurs de positions". Ces modules <GP> sont équivalents à des <SOL> mobiles. Néanmoins leur mouvement est fixé et n'est pas modifié par le comportement des autres éléments physiques du modèle. Dans ce cas, si organisation spatiale il y a, alors elle est prédéterminée en amont de la scène ou de la simulation en cours et inscrite dans la

cinématique résultante introduite par les <GP> dans la simulation courante. En dehors de ce cas très particulier, et compte tenu du fait que certaines contraintes spatiales peuvent également être codées dans les interactions, comme nous l'avons montré dans le paragraphe « Relation forme-mouvement », deux cas de figures se présentent ici selon que :

a. Les organisations topologiques et géométriques soient codées dans les positions initiales des <SOL>

b. ou qu'elles puissent être codées dans les interactions.

Cas 3 : La scène physique peut présenter un ensemble d'éléments mobiles pouvant présenter des organisations spatiales complexes qui ont vocation de maintenir au cours du temps, par exemple « rester dans un plan ou sur une ligne ». A priori, ces éléments peuvent être déformables ou non. Deux possibilités se présentent ici : soit de satisfaire cette contrainte portant sur le maintien de cette organisation spatiale au cours du temps via la définition d'interactions physiques adaptées ; soit définir des contraintes géométriques que la simulation sera chargée de satisfaire. Dans ce dernier cas, il s'agira d'analyser la faisabilité d'une telle démarche, eu égard en particulier à sa généralité ainsi que ses avantages et inconvénients, en particulier dans des cas où elle pourrait s'avérer positivement concurrente à la modélisation physique. Un cas plus simple pourrait être en effet de considérer cette possibilité pour des parties mobiles rigides. Leur description géométrique conduit alors à la modélisation de parties totalement indéformables, puisque modélisées géométriquement, mais qui peuvent être en interaction physique avec les autres éléments de la scène. Ce cas consiste à introduire des contraintes géométriques plus ou moins généralisées ou généralisables dans les réseaux masses-interactions.

Ce dernier cas ne concerne pas typiquement la spécification de l'état initial du système dynamique. Une partie de ce cas concerne les modules <MAT> 1D ou 2D dans un espace 3D, puisque ces modules peuvent être utilisés pour contraindre certains degrés de liberté dans le modèle. Nous introduirons et rappellerons ici dans la conclusion de ce chapitre quelques éléments de discussions concernant cette question.

Dans ce qui suit, nous examinons successivement :

1. Le cas 1 qui consiste en l'introduction, au sein de MIMESIS, d'outils permettant de spécifier des organisations spatiales initiales simples pour des modules matériels <MAS> ou <SOL>, organisations définies par leur topologie et/ou leur géométrie,

2. Le cas 2a qui consiste en l'importation d'organisations spatiales initiales (géométrie et/ou topologie) des modules matériels <MAS> ou <SOL>, dès lors que des ensembles de modules présentent des organisations spatiales complexes ainsi que la spécification des fonctions d'interaction physique, en nature et en paramètres, sur lesquelles ils sont connectés. Nous verrons que, dans le cas général, les outils de modélisation géométrique ne sont pas adaptés à ce problème. Par conséquent, nous restreindrons notre étude de leur apport aux organisations de <SOL> fixes, que nous avons appelées "Obstacles".

3. Le cas 2b qui consiste dans le codage implicite des formes des obstacles fixes dans des interactions physiques non isotropes.

Nous montrerons ainsi comment la diversité des outils que nous proposons permet progressivement et surtout de manière adaptée, de faire le pont entre des données spatiales, issues du domaine géométrique, et le modèle physique.

2. Introduction de structures spatiales élémentaires au sein de MIMESIS

2.1. Introduction

Le placement à la main des positions initiales de chaque module matériel est une opération qui devient vite impossible malgré les fonctionnalités de labellisation de parties de réseau ou déplacement global de sous-parties. Les procédures de base de PNSL pour la définition des positions initiales (`définir_PI x y z`) restent également insuffisantes malgré la puissance du langage. Des outils et structures périphériques manquent, que ce soit pour la modalité script ou la modalité graphique. Par conséquent, l'intervention d'objets et procédures géométriques dans le processus de modélisation physique au sein de MIMESIS est incontournable.

Le besoin d'une organisation des positions initiales des <MAT> dans l'espace intervient forcément dans le processus de modélisation. Il peut s'agir d'une disposition grossière pour préparer un placement plus complexe à l'aide de la simulation, d'une disposition plus précise suivant une structure géométrique, pouvant correspondre à un état d'équilibre. Il peut également s'agir de modifications très fines respectant une organisation préexistante (un infime déplacement d'un grand nombre de masse pour obtenir un équilibre juste instable). Les possibilités d'édition de nature géométrique doivent donc exister, mais elles doivent être minimales, c'est une édition *pour* la modélisation physique.

Il s'agit donc d'introduire au sein de MIMESIS des outils de base qui permettent de créer et de manipuler des structures spatiales dans la phase "Conditions initiales", simples au sens que :

- Les structures créées ainsi que leur manipulation correspondent à des règles géométriques de base,
- Les interactions physiques auxquelles sont connectés les modules ainsi organisés seront exclusivement à la charge de l'utilisateur. Leurs paramètres ne sont pas déterminés automatiquement en fonction des structures spatiales.

Ces structures sont des objets auxiliaires pour les tâches de placement. Elles permettent de définir des contraintes géométriques (un placement sur une ligne) et des transformations géométriques (rotation par rapport à un axe). Ces objets auxiliaires font partie des abstractions de MIMESIS V. Ce sont des objets spatiaux (cf. Chapitre II §2.1) qui permettent de définir un grand nombre de points dans l'espace, structurés ou non

(un objet spatial peut être un simple nuage de points) et de paramétrer des transformations géométriques.

2.2. Objets spatiaux : quelle séparation entre topologie spatiale et plongement géométrique ?

L'introduction de structures spatiales dans MIMESIS pose la question de la dissociation du qualitatif et du quantitatif. Nous avons vu comment cette dissociation se manifeste pour la modélisation physique particulaire par deux phases de conception séparées, PSQL pour la topologie du réseau et PSQN pour les données quantitatives de ses paramètres. Pour les objets spatiaux, cela revient à parler d'une part de topologie, au sens spatial, et de plongement géométrique. Il est donc naturel de poser la question de la séparation de ces deux concepts dans l'interface. Faut-il regrouper toutes les manipulations de type qualitatif et de type quantitatif dans deux phases distinctes de modélisation, quel que soit le type d'objet manipulé ? Au delà de la question de l'organisation de l'interface, le problème est celui du lien entre les deux univers, physiques et géométriques. Ce problème se retrouve à plusieurs moments du processus de modélisation, y compris bien entendu pour les conditions initiales. En prenant ce moment du processus de modélisation, on peut s'interroger sur les liens avec entre topologie spatiale et topologie du réseau. En effet les conditions initiales sont dépendantes de la topologie du réseau dans le sens où une certaine topologie suppose une certaine organisation de l'espace si on cherche un état initial proche de l'équilibre. L'exemple des modèles de tissu (illustrés au chapitre III §7.3) est à ce titre très éclairant. La topologie suggère – et suggère seulement – dans ce cas une répartition régulière sur un rectangle, avec un voisinage spatial homéomorphe au voisinage topologique du réseau. A l'inverse, pour une topologie d'agglomérat, qui n'est pas du tout structurée (toutes les masses sont équivalentes), la topologie n'a aucune influence dans la détermination des conditions initiales.

La question de la séparation entre topologie et plongement géométrique pour les objets spatiaux est donc encore ouverte. Dans un premier temps, cette séparation ne se justifie pas, car elle complique la tâche sans apporter un plus intéressant pour la complexité. De plus, cette question n'est pas seulement valable pour la phase CI, elle l'est aussi, et même davantage, pour la phase d'habillage. Cette phase, que nous n'avons pas abordée en profondeur dans nos travaux, est l'objet d'un projet ANR en cours qui explore notamment la relation entre topologie spatiale et modèle physique.

2.3. Les objets spatiaux élémentaires dans MIMESIS

Les objets spatiaux que nous présentons dans cette partie étendent les objets spatiaux primaires que l'on trouve comme attributs des modules. Ils font partie de ce que nous nommons les objets d'intérêt auxiliaires de MIMESIS V, le réseau masses-interactions constituant l'objet d'intérêt principal. En tant qu'objet d'intérêt, le réseau dispose de

modalités de création, d'édition et d'adressage. Pour les modules ce sont les labels qui tenaient principalement le rôle d'adressage. Pour les objets spatiaux, la modalité d'adressage principale est alphanumérique comme pour les modules. Mais la chaîne de caractères désignant un objet spatial n'est pas un label, ce terme étant réservé aux modules. Nous la nommons simplement "os-label", pour la distinguer des labels. La distinction n'est pas seulement lexicale et conceptuelle, elle se traduit au niveau du noyau par un arbre distinct, autorisant ainsi d'avoir deux chaînes de caractères identiques désignant à la fois un module et un objet spatial.

Nous proposons une collection réduite d'objets spatiaux. Cette collection est conçue pour effectuer des placements structurés simples et des opérations géométriques simples (que nous détaillerons dans le paragraphe suivant). Les objets spatiaux peuvent aussi accueillir des données importées d'autres logiciels, nous aborderons plus en détail ce point dans la partie 3. La collection d'objets spatiaux est formée par trois types d'éléments :

- Les droites, axes et segments (primitives 1D)
- Les plans, rectangles et les cercles (primitives 2D)
- Les sphères, boites, nuages de points et champs de hauteurs (primitives 3D)

Certains des objets évolués sont déjà présents dans MIMESIS 4, ils sont utilisés comme commodités pour déterminer les positions initiales des <MAT> : placer aléatoirement dans une boîte, régulièrement sur un segment, régulièrement sur un cercle. Dans MIMESIS 4, ces objets ont une durée de vie limitée, ils n'existent que pendant la procédure de placement. Les objets spatiaux ne sont donc pas réellement réifiés dans MIMESIS 4, ils ne peuvent être liés durablement à une procédure de placement. Une même contrainte géométrique n'est pas modifiable au cours du processus de modélisation, elle doit être recréée à chaque modification. L'utilisateur est ainsi contraint de redéfinir la relation entre les modules (plus exactement leurs conditions initiales) et l'objet spatial définissant la contrainte. L'objet spatial lui-même doit être redéfini et non simplement modifié. Ces limitations expliquent pourquoi nous avons fait des objets spatiaux de vrais objets d'intérêt : pérennes, éditables tout au long du processus de modélisation, et adressables.

2.4. Les opérateurs spatiaux : définir et manipuler des objets spatiaux.

La création d'objets spatiaux est une opération de nature géométrique. Toutefois, nous tenons à attirer l'attention du lecteur sur le fait que la création de tels objets contient aussi une part topologique de par le simple fait de le désigner comme une chose. En effet, mis à part le cas particulier du nuage de points, tous les objets spatiaux sont topologiquement semblables car ils définissent des ensembles de points continus. Ainsi, un objet spatial se constitue en tant qu'objet, avec son unité, avant même d'avoir un plongement géométrique. Dans la mesure où ces objets vont être utilisés pour déterminer des conditions initiales de groupes de <MAT>, la question de l'identification d'un sous-réseau à un objet via l'objet spatial se pose. Or cette identification est erronée en raison de la nature systémique des réseaux masses-interactions et peut donc

conduire à des erreurs de modélisation. Il est donc important de ne pas favoriser une telle confusion dans l'interface. Nous ne pensons pas être en mesure d'apporter une réponse définitive à ce problème complexe, nous faisons le choix temporaire de restreindre la création et l'édition d'objets spatiaux à la phase CI. MIMESIS reste une plateforme d'expérimentation, nombre de choix que nous avons faits pendant nos travaux sont amenés à évoluer en fonction des usages et de la pédagogie.

Un objet spatial peut être créé de quatre manières :

1. à partir d'objets spatiaux primaires (conditions initiales de modules, paramètres d'autres objets spatiaux)
2. à partir de fichiers (importation),
3. à partir de données de la simulation.
4. Directement, c'est à dire que les paramètres sont saisis à la création par l'utilisateur, soit graphiquement, soit alphanumériquement.

Nous avons donc fait le choix de confondre création et paramétrage des objets spatiaux dans la même opération. Cependant, les paramètres d'un objet spatial peuvent être modifiés après leur création, soit directement, soit via des opérateurs spatiaux. Ces opérateurs correspondent aux applications affines les plus élémentaires. Ils s'appliquent à des points 3D et à tous les objets spatiaux sauf les vecteurs. Les paramètres de ces opérateurs sont naturellement des objets spatiaux :

1. Translation : un vecteur
2. Rotation : un axe et un angle
3. Homothétie : un point 3D et un coefficient scalaire
4. Dilatation : un vecteur et un coefficient scalaire
5. Symétrie : un plan

Nous venons de traiter de la création et de l'édition d'objets spatiaux en tant qu'objets d'intérêt auxiliaires. Le cœur de la question des conditions initiales dans MIMESIS V réside à présent dans le lien qu'entretiennent ces objets spatiaux avec les conditions initiales des modules <MAT>.

2.5. Enjeux du lien entre objets spatiaux et conditions initiales

Comment manipuler des conditions initiales ? Comment définir des valeurs par défaut à la première itération de la phase CI ? Quelle articulation relie les conditions des modules <MAT> aux objets spatiaux ? Ces questions difficiles ne sont pas seulement des questions techniques ou même ergonomiques. Elles ont trait à la difficulté du processus de modélisation, au problème du rapport ambigu/complexe/ambivalent entre géométrie et physique. Nous devons trouver un compromis entre un lien suffisamment fort pour permettre des manipulations complexes à moindre effort cognitif et une indépendance nécessaire du modèle physique vis-à-vis des différentes contraintes géométriques.

La question de cette dépendance se traduit dans la relation objets spatiaux / CI. Nous avons le choix entre établir une contrainte directe entre les deux, ou bien d'expliciter l'application de la contrainte. Concrètement, ces deux choix signifient 1) pour la contrainte directe, de définir un lien entre un objet spatial et un ensemble de CI 2) pour l'explicitation, de disposer d'une procédure de placement sur un objet. Dans le premier cas, toute modification d'un objet spatial serait immédiatement répercuté sur les CI qui lui sont liées. Dans le deuxième cas, toute modification d'un objet spatial devra être complétée par une exécution de la procédure de placement pour que les modifications soient répercutées.

Ces deux choix ne peuvent être considérés de manière exclusive. Il est manifeste que le choix exclusif de la contrainte directe serait dommageable pour l'indépendance du physique vis-à-vis du géométrique. Ce choix remet notamment en cause la possibilité de modifier individuellement les conditions initiales d'un module. Le deuxième choix suppose dans un premier temps une distance syntactique importante, c'est à dire plus de manipulations pour réaliser une même tâche, qu'avec le premier choix. Mais associé à l'outil fonction, présenté au chapitre III §5.3, cette distance peut être largement réduite avec un bénéfice intéressant pour la maitrise du processus de modélisation. En effet, l'objet fonction peut jouer le rôle de liaison entre les objets spatiaux et les CI, une liaison beaucoup plus souple que la contrainte directe. Nous pouvons ici revenir sur les notions de collaboration horizontale et verticale (chapitre II §4.3 et §4.4). En effet, si nous basons la phase CI sur une articulation bien choisie entre fonctions, objets spatiaux et conditions initiales des modules, nous pouvons introduire des synergies très intéressantes entre modalités graphique et textuelle. La collaboration horizontale se manifesterait, par exemple, par la manipulation graphique d'un objet spatial et le placement sur cet objet via un objet fonction. Nous montrerons un exemple d'une telle synergie dans la partie dédiée aux exemples.

Les options que nous venons de présenter se prêtent bien à la manipulation par lot des CI des modules. Mais dans bien des cas, que ce soit dans les modèles de grande taille ou de petite taille, la manipulation individuelle des CI est la manière la plus adéquate de régler le problème. Dans ce cas, les objets spatiaux ne sont plus l'outil pertinent pour manipuler les CI. On retrouve alors les outils traditionnels de MIMESIS 4 : tableau, boîtes de dialogue et espace graphique 3D.

2.6. Espace graphique pour la manipulation des conditions initiales

La phase CI propose un espace graphique 3D représentant les objets spatiaux du projet. L'utilisateur visualise donc positions et vitesses initiales des modules <MAT> et les différents objets spatiaux auxiliaires. Cet espace permet également l'édition graphique de ces objets spatiaux. La surcharge visuelle doit être rigoureusement maitrisée dans cet espace, surtout dans le cas des modèles de grande taille qui exhibent un grand nombre de modules <MAT>. Tous ces objets spatiaux peuvent donc être cachés pour permettre à l'utilisateur de se concentrer sur les objets concernés par la tâche en cours.

L'espace graphique doit également présenter un certain nombre de repères, absolus et relatifs, pour une bonne appréhension de l'espace par l'utilisateur. Nous reprenons

l'échelle et la grille adaptable en résolution de l'espace de modélisation 3D de MIMESIS 4 (cf. figure 84). Des repères absolus chiffrés manquaient à cet espace. Nous avons donc ajouté des réglettes gradués sur les bords, actives uniquement lorsque la caméra est dirigée selon les axes Ox, Oy ou Oz. À cet outil s'ajoute l'affichage des coordonnées du point matériel survolé par la souris.

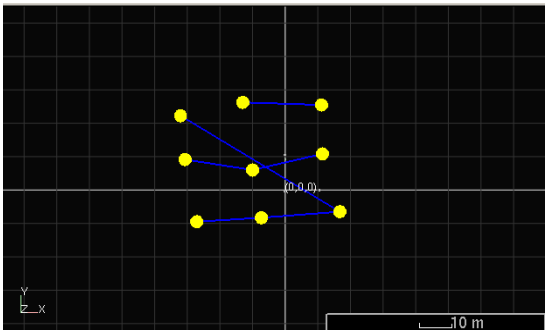


Figure 84 : Grille et échelle sur l'espace 3D

L'utilisation d'un espace 3D est problématique du fait de sa troisième dimension. La profondeur entraine des difficultés dans les tâches de sélection, de navigation et de perception des structures spatiales. Les outils de repérage sont une première aide, particulièrement pour la perception, mais restent insuffisants. Un paramètre important dans la visualisation d'espaces tridimensionnels est le type de projection. Pour la modélisation géométrique 3D, une projection en perspective est adaptée, car la perception de la profondeur sur les volumes est primordiale. En revanche, pour des tâches relatives à la modélisation physique, notamment la tâche de sélection de <MAT>, usuellement représentés par des sphères, l'information de profondeur est souvent beaucoup moins pertinente. C'est pourquoi nous proposons d'abord à l'utilisateur une projection orthographique. La projection en perspective reste cependant disponible, dans le cas où la situation – objets spatiaux tridimensionnels complexes par exemple – le requerrait. La tâche de sélection de <MAT> est particulièrement problématique dans une scène 3D. Le lasso ou l'outil de sélection rectangulaire sont inadaptés car ils ne prennent pas en compte la profondeur. Nous proposons donc d'utiliser les objets spatiaux 3D, sphère et boite, comme outils de sélection. L'utilisateur peut ainsi sélectionner les <MAT> dont les positions initiales sont inscrites dans une sphère ou une boite, en profitant des modalités de manipulation interactive de ces objets dans l'espace 3D. Ces modalités sont semblables à celles que l'on trouve dans les logiciels de modélisation géométriques : poignets, points de contrôle.

Nous avons décrit les propriétés de l'espace 3D et les différents outils de manipulation et de repérage qu'il présente. Ces éléments sont également disponibles dans l'espace de simulation, également tridimensionnel. Rappelons que cet espace est aussi disponible dans la phase CI, permettant à l'utilisateur de sélectionner des modules sur des critères ayant trait à une simulation.

2.7. Conclusion

Nous avons introduit des outils permettant de spécifier des organisations initiales simples pour les modules <MAT>. Les objets spatiaux et les procédures assorties s'intègrent dans MIMESIS V, au côté des outils décrits dans le chapitre précédent, notamment l'espace 3D que nous venons de détailler. Nous disposons ainsi d'un socle solide pour la manipulation des conditions initiales. Le placement sur des structures géométriques simples et interactivement manipulables est ainsi accessible, mais aussi extensible grâce à l'utilisation de PNSL. Rappelons que les scripts PNSL sont facilement combinables avec des manipulations graphiques grâce à l'outil fonction défini au chapitre II §5.3.

Cependant, des organisations spatiales plus complexes peuvent intervenir dans le processus de modélisation. Les structures spatiales disponibles sont alors insuffisantes, mais ce n'est pas réellement le problème puisque des modèles géométriques complexes sont nécessairement conçus en dehors de MIMESIS. Le problème de leur importation se pose alors, et de toutes les conséquences qu'un tel import peut avoir sur l'ensemble des modèles, et pas uniquement au niveau des conditions initiales.

3. Importation de structures spatiales dans MIMESIS : positions initiales et interactions physiques

3.1. Introduction

Il existe des outils dédiés et particulièrement bien adaptés à la modélisation de structures et données spatiales. Il est donc naturel de vouloir les utiliser pour proposer des organisations spatiales initiales dans MIMESIS. Rappelons que le problème est double :

1. disposer d'un placement initial spatialement structuré d'ensemble de points, qui seront dans MIMESIS les valeurs initiales des positions des modules matériels <MAS> et <SOL>,

2. disposer d'éléments permettant d'être utilisés dans la détermination des paramètres spatiaux des fonctions d'interaction auxquels sont connectés ces <MAS> et ces <SOL>.

Il y a là un changement de paradigme important à réaliser, le passage d'une conception géométrique à une conception physique. Les formats de données autant que la manière de penser sont radicalement différentes. C'est pourquoi ce passage ne peut se faire ni par l'intégration d'un logiciel d'import dans MIMESIS, ni par l'intégration d'un greffon d'export dans un logiciel de modélisation géométrique. Penser ce passage, cette transformation, requiert un logiciel tiers, permettant de réaliser un changement de système de représentation, depuis le géométrique vers le physique. La difficulté de ce changement de représentation réside dans un compromis entre l'importation de l'organisation des données spatiales et une structuration trop importante des données importées dans MIMESIS. En effet, MIMESIS importe des positions et des données intervenant dans les paramètres des interactions, principalement des distances. MIMESIS n'importe donc que des valeurs, pas des structures. La corrélation entre ces différents types de données, par exemple une position et une distance définissant une sphère, ne doit pas être encapsulée car elle est alors imposée au lieu d'être proposée. L'association doit donc être suggérée, et c'est justement le logiciel tiers qui est en charge de désempaqueter tout en conservant de manière plus souple une association.

Dans le problème précis que nous cherchons à traiter dans cette partie, c'est à dire une organisation de <SOL> fixes et d'interactions définissant des "Obstacles", le changement de système de représentation se fait en deux étapes. Nous venons de décrire la

deuxième, qui consiste à désempaqueter les données spatiales, de manière à séparer organisation et valeurs. La première étape est une transformation géométrique qui vise à transformer une forme quelconque en une suite de sphères. Cette transformation aboutit ainsi à une structure géométrique beaucoup plus proche du paradigme masses-interactions que des polyèdres quelconques.

3.2. Etat de l'Art

3.2.1. Approximation d'un maillage par des sphères pour la détection de collisions

La gestion de collisions dans une scène 3D est un domaine critique en informatique graphique. Il est connexe à notre problématique, mais s'en démarque par la préséance de la géométrie et la focalisation sur l'interaction de collision. En conséquence, le problème est essentiellement traité comme un problème géométrique, l'interpénétration de deux volumes. Néanmoins, l'éventail des techniques utilisées est très large, depuis les boîtes englobantes [Zac02] jusqu'aux méthodes basée image, en passant par les méthodes stochastiques. Les différentes variantes des boîtes englobantes sont intéressantes, particulièrement les méthodes basées sur des boîtes englobantes sphériques. En effet, comme nous l'avons déjà mentionné, nous recherchons une transformation géométrique permettant de passer d'une modèle polygonal à une collection de sphères.

L'approximation d'un maillage par des sphères a été proposé à plusieurs reprises ([Hub96], [BO02]) comme méthode de détection de collision efficace. Hubbard est le premier, en 1996, à s'inspirer du Medial Axis Transform (MAT) ou squelette pondéré. Cette méthode mathématique, issue de la recherche en biologie ([Blu67], permet d'approximer un volume par un ensemble de sphères en calculant son axe médian. L'axe médian d'une forme est le lieu des centres des boules maximales inscrites dans la forme (cf figure 85). A partir de cet objet mathématique, Hubbard définit une structure hiérarchique, sorte d'octree de sphères s'appuyant sur l'axe médian (cf. figure 86).

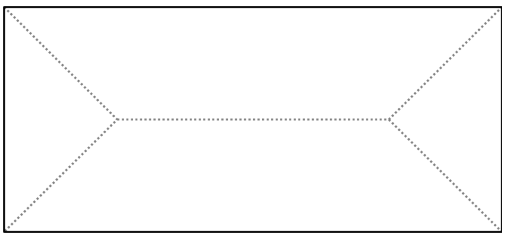


Figure 85 : Un rectangle et son axe médian en pointillés

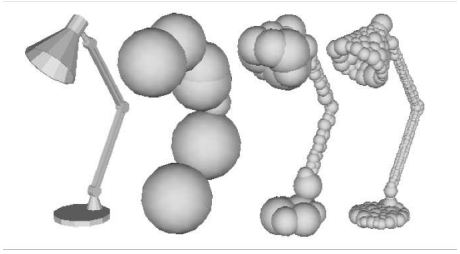


Figure 86 : Différentes collections de sphères approchant un modèle 3D de lampe (tiré de [Hub96])

Pour calculer l'axe médian, Hubbard distribue uniformément des points à la surface du maillage et calcule le diagramme de Voronoi associé à ces points. Les sommets du diagramme constituent une approximation, satisfaisante selon l'auteur, de l'axe médian. Cette méthode est améliorée par Bradshaw et O'Sullivan [BO02]. Ils raffinent l'approximation en contrôlant la distribution des points sur le maillage de manière à réduire l'erreur (cf. figure 87)

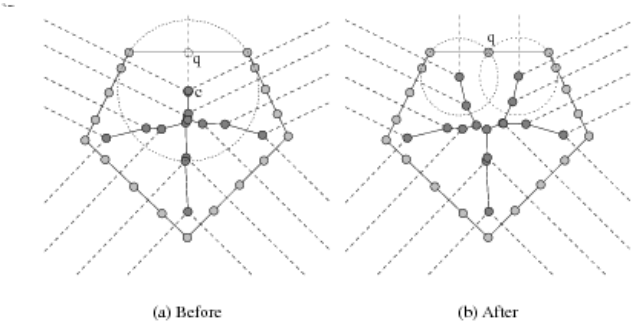


Figure 87 : Un raffinement de l'axe médian par adaptation de l'échantillonnage (repris de [BO02])

Ces deux exemples montrent bien comment la physique est assujettie à la géométrie. En effet, même si le but est la détection de collision dans une simulation physique, cet objectif n'impacte pas sur la manière de calculer l'arbre de sphères. Le seul objectif affiché est le meilleur compromis entre précision géométrique et efficacité de calcul.

Les travaux de Benoit Chanclou sur la modélisation d'un véhicule sur un terrain [Cha96] l'ont amené à modéliser des cailloux et des rugosités avec une approche semblable au squelette pondéré. Son approche se distingue des précédentes car la transformation en sphères est faite entièrement manuellement, l'axe médian n'est pas mentionné.

3.2.2. Autres applications et algorithmes pour le squelette pondéré

L'axe médian a été énormément développé en dehors du domaine de la détection de collisions. On trouve notamment beaucoup de travaux dans le domaine de la reconstruction de surfaces à partir de nuages de points. Ces nuages de points proviennent, pour la plupart des applications, d'un scanner laser. La méthode la plus reconnue dans ce domaine est l'algorithme dit du "Power Crust" [ACK01], "Power" pour

Power Diagram, un diagramme proche du diagramme de Voronoï, et "Crust" pour croûte. Cette méthode a l'intérêt de résoudre le défaut du diagramme de Voronoï en 3D. En effet, si 4 points sont cocirculaires, ce qui semble arriver usuellement d'après [ACK01], un sommet de Voronoï apparaît, proche du contour, mais éloigné de l'axe médian. Dans un domaine plus proche des mathématiques pures, on note les travaux de Culver [CKM99]. Il propose une méthode analytique exacte pour tous les polyèdres. Lorsque le polyèdre est convexe, l'axe médian est une surface linéaire par morceaux. Lorsqu'il est concave, l'axe médian est une surface quadratique par morceaux.

3.2.3. CSG et interactions

Dans le cadre de ses travaux sur la plateforme de simulation synchrone multisensorielle interactive [Cas10], Julien Castet s'est intéressé à la modélisation 3D d'une poêle pour l'interaction avec un matériau de type pâte. La plateforme étant basée sur le formalisme CORDIS-ANIMA, l'enjeu était de trouver un moyen de modéliser une poêle manipulable avec six degrés de liberté en interaction rigide avec des particules. Il propose de composer les interactions de butée et de bulle pour faire émerger des formes autres que la sphère, creuse (bulle) ou pleine (butée). La figure 88, tirée de [Cas10], résume les possibilités de cette technique.

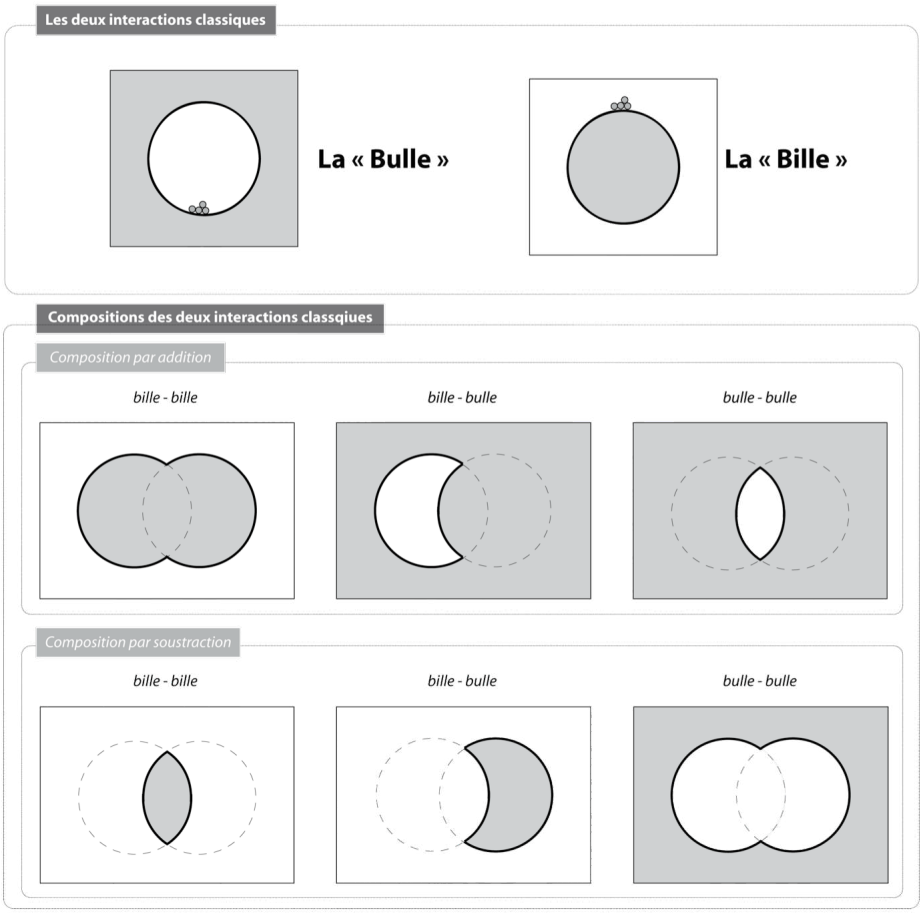


Figure 88 : Compositions des interactions <BUT> et <BUL> (tiré de [Cas10])

Cette méthode de modélisation avait déjà été utilisée dans un modèle 2D qui mettait en scène un gobelet contenant des billes et manipulé également en temps réel [FRL+86]. Notons qu'il apparaît difficile d'importer des géométries complexes avec cette méthode. Elle est davantage un moyen original de modéliser des objets géométriques rigides avec CORDIS-ANIMA.

3.2.4. Importation dans MIMESIS 4

Les processus d'importation de positions initiales sont assez limités dans MIMESIS 4. Il n'est possible d'importer que des fichiers au format MIMESIS, nommé .init, décrivant les positions et vitesses initiales pour chaque <MAT>, référencé par son nom unique dans le modèle. Un tel format signifie d'ailleurs que l'import de conditions initiales n'est prévu que pour des données provenant d'un modèle MIMESIS et partageant les mêmes noms uniques de <MAT>. Ce format n'était donc pas utilisé pour importer des données spatiales depuis un logiciel de modélisation géométrique, mais pour réaliser des placements par simulation. Cette technique consiste à créer un modèle physique pour trouver un état d'équilibre à partir de positions initiales aléatoires. L'état d'équilibre pouvait ensuite être injecté dans un deuxième modèle physique.

Nous avons participé avec Matthieu Evrard à une extension de MIMESIS 4 permettant d'importer des données depuis un modeleur géométrique, en utilisant un format proche du format `.init`, mais ne présentant que les positions initiales. Ce format a été nommé `.pos`. Cette extension consiste en une interface de "mapping" (mise en correspondance), comme décrite dans [Evr09]. L'importation d'un fichier au format .pos aboutit à la représentation graphique d'un nuage de points. L'interface (cf. figure 89) permet de faire correspondre une position dans le nuage de points à un module <MAT> sélectionné dans un espace topologique ou dans l'arbre des labels.

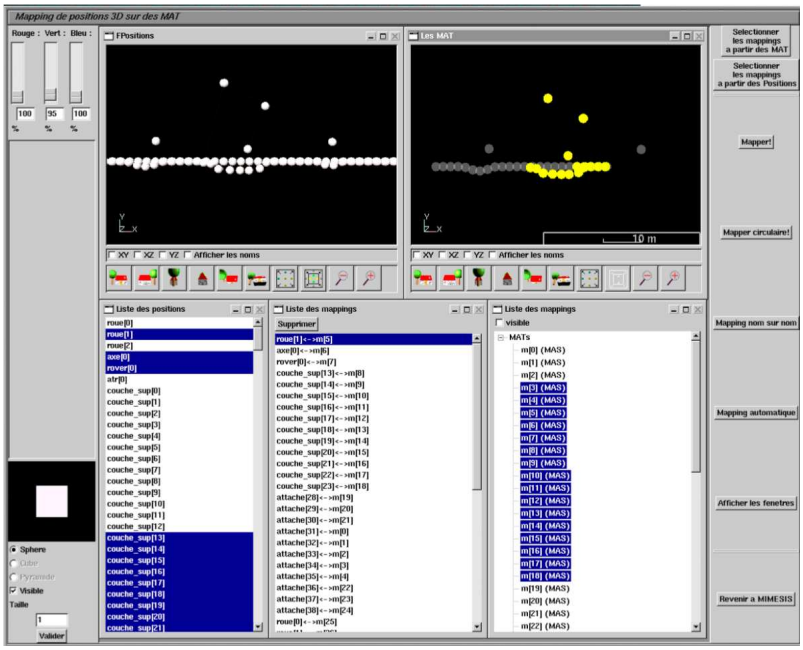


Figure 89 : Interface de mapping

Pour être complet, ce processus d'importation nécessite encore la conception de programme de conversion de fichiers depuis des formats géométriques standards (.mesh, collada, .obj, ...) vers le format .pos. Cette étape est particulièrement problématique, car c'est là que se situe le pont entre l'univers géométrique et l'univers physique.

3.3. Méthode d'importation de données géométriques dans MIMESIS V

L'introduction de données spatiales dans le modèle physique doit s'intégrer au processus de modélisation, car elle impacte fortement sur les phases PSQN et Conditions Initiales. Le processus de modélisation demeure un processus cyclique basé sur l'expérimentation et l'observation. Les transformations géométriques que nous proposons s'intègrent donc dans ce cycle. En effet, ces transformations ont une influence sur le mouvement final, et le modélisateur expérimente l'influence que peut avoir une transformation particulière. Les données spatiales importées, c'est à dire les centres et rayons des sphères, ne suffisent pas à déterminer entièrement les paramètres cibles dans le modèle physique, c'est à dire les positions initiales des modules <SOL> et les seuils des interactions. En effet ce dernier élément est partiellement déterminé par le rayon des sphères. Il est aussi grandement influencé par le phénomène en interaction avec l'obstacle. Ces seuils sont donc réglés en fonction de l'ensemble des paramètres physiques du modèle.

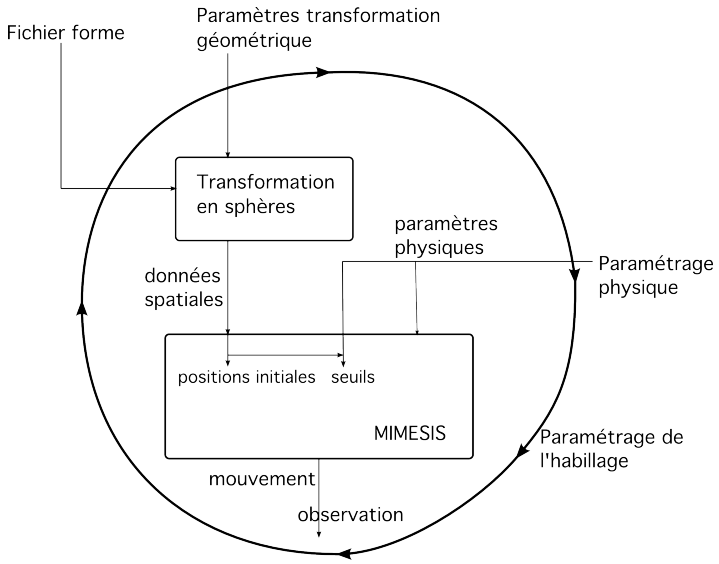


Figure 90 : Processus de modélisation et importation de données géométriques

La figure 90 résume cette méthode. On remarque notamment que la transformation des sphères se fait en dehors de MIMESIS. Comme nous le recommandions en introduction, un logiciel tiers intervient dans la boucle, faisant le pont entre un modeleur géométrique (Maya, Autocad, Blender...) et le modeleur physique qu'est MIMESIS. Cette application accepte les formats de description de forme les plus courants et permet de paramétrer l'algorithme de transformation en sphères. Les données spatiales que l'application fournit

en sortie sont labellisées pour préparer l'importation dans MIMESIS. Le couple (centre, rayon) définissant chaque sphère est découplé, le logiciel exporte une liste de points et une liste de distances. L'association entre les deux listes ne se fait que par les noms associés aux points et aux rayons. Chaque élément des deux listes est ainsi identifié par rapport à un obstacle et un numéro (ex : `nomObstacle_rayon_n`). Ce découplage est fondamental pour conserver l'indépendance de MIMESIS V vis à vis de la géométrie. La structuration de l'espace n'est pas imposée, elle est proposée. La nuance entre l'import d'objets sphères d'une part, et une liste de distances et de points d'autre part, est cruciale pour renforcer cette distinction. C'est un point qui prend tout son sens dans la pédagogie de MIMESIS, et qui illustre bien toute la complexité du rapport forme/mouvement.

3.4. Contour optique et contour mécanique

Toujours dans cette problématique de la relation forme/mouvement, mais à un autre stade de la modélisation, la relation entre l'habillage du modèle et la sphérisation repose la question de la correspondance entre contour optique et contour mécanique. En effet, cette correspondance ne va pas de soi, il ne faut pas confondre le problème physique, c'est à dire la genèse de la cause du mouvement, avec le problème de la visualisation de ce mouvement. Confondre ces deux problèmes, chercher en permanence la correspondance entre les deux types de contour réduit le champ des possibles et ramène le modélisateur à des problèmes du type collision entre solides rigides. L'exemple de la modélisation de mouvements de foules dans un environnement urbain est à ce titre tout à fait pertinent. Le modélisateur peut en effet faire le choix de modéliser l'interaction entre la foule et les immeubles par des interactions de butées dessinant un évitement plus large que ne le suggère la géométrie (cf. figure 91). On a en effet rarement vu une foule se frotter au coin d'un immeuble ! En revanche, lors de la phase d'habillage, le modélisateur pourra préférer visualiser l'immeuble tel qu'il est défini géométriquement. Dans ce cas le contour physique et le contour optique de l'immeuble ne correspondent pas.

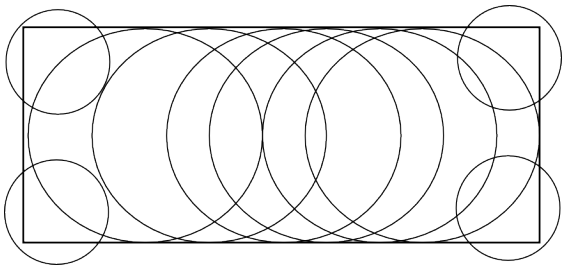


Figure 91 : Une sphérisation d'un rectangle représentant un immeuble

3.5. Chapelet et tapis de masses

La technique du chapelet (en 2D) ou du tapis (en 3D) de masses est une technique de pavage d'un contour par des cercles en 2D et des sphères en 3D. Le pavage est paramétré soit par le nombre total de masses soit par la densité (nombre de masses par

unité de longueur ou par unité de surface). Le deuxième paramètre est le rayon des cercles ou sphères. Le troisième dernier paramètre est la translation selon la normale qui permet de déplacer les masses vers l'intérieur ou l'extérieur du contour. La figure 92 montre un exemple de transformation d'un contour 2D en chapelet de masses.

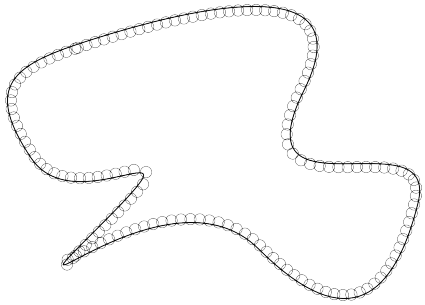


Figure 92 : Un contour 2D et un pavage associé

La combinaison des deux premiers paramètres permet de définir la rugosité résultante comme le rapport entre le rayon et la distance inter masses. Lorsque ce rapport est supérieur à 1, l'obstacle est poreux. Lorsqu'il est inférieur à 1, l'obstacle est rugueux, et plus le rapport est faible, plus l'obstacle est lisse. Cependant, ce rapport quantifie la rugosité géométrique. La rugosité physique dépend du rapport entre le seuil d'interaction et la distance inter masses, mais également des autres paramètres physiques de l'interaction (raideur et viscosité).

Nous avons testé l'algorithme de transformation d'Inkscape [Ink], qui traite des contours 2D. Nous avons constaté une certaine sensibilité sur les zones du contour où la courbure est importante (cf. figure 92). L'application dédiée à la transformation devra donc permettre de modifier manuellement l'ensemble des sphères pour pallier aux défauts de l'algorithme. Cette intervention reste très sommaire, elle est limitée aux actions de suppression, de création et de déplacement.

L'algorithme peut être étendu en intégrant des irrégularités dans la distribution et dans le rayon des masses. Des paramètres de bruit gaussien pour ces deux paramètres sont alors rendus disponibles. En conséquence, le paramètre de rugosité géométrique correspondra alors à une moyenne et non plus à une mesure identique sur tout le contour.

3.6. Sphère circonscrite

La méthode de la sphère circonscrite consiste à approximer un obstacle par la plus petite sphère le contenant. Cette méthode à l'avantage d'être simple à mettre en œuvre, autant au niveau du calcul que de l'importation dans le modèle physique. En revanche, elle est réservée à des interactions phénomène/obstacle très sommaires. Le modélisateur considère que la forme de l'obstacle n'est pas importante, seules sa position et son étendue nécessitent d'être prises en compte.

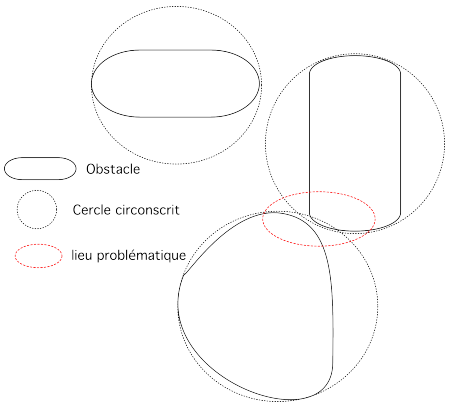


Figure 93 : Transformation d'une scène géométrique via la méthode des cercles circonscrits

La figure 93 montre un exemple de transformation de trois obstacles en cercle circonscrit. Cet exemple révèle un problème relié à la relation entre les différents obstacles. En effet, le manque de précision de cette méthode peut entrainer la fusion d'obstacles (comme indiqué sur le lieu problématique figure 92). Un goulot d'étranglement devient alors un obstacle, ce qui disqualifie cette méthode pour de pareils cas.

3.7. Sphérisation

La technique que nous avons appelé "sphérisation" s'appuie sur le Medial Axis Transform (MAT) ou squelette pondéré, décrit au §3.2.1. Elle consiste à approximer une forme en se basant sur le squelette pondéré. Concrètement, l'utilisateur demande une approximation du squelette pondéré ou bien on utilise le squelette pondéré comme canevas pour en déduire un sous-ensemble. Nous nous distinguons des travaux de Hubbard [Hub96], Bradshaw et O'Sullivan [BO02] sur deux points fondamentaux. D'abord nous ne cherchons pas à déduire une structure hiérarchique du squelette pondéré. En effet, nous ne recherchons pas une détection de collision efficace, le temps réel n'est pas une contrainte critique dans MIMESIS, qui se destine avant tout à l'animation en temps différé. En revanche, le modélisateur recherche une certaine qualité dans l'interaction. Or cette qualité ne peut être appréciée à partir de critères purement géométriques, comme une erreur mesurée avec la distance d'Hausdorff utilisée dans [Hub96]. La collection de sphères basée sur le squelette pondéré doit donc être construite de manière semi-automatique, de manière à s'insérer intelligemment dans le processus de modélisation décrit au §3.3.

Les algorithmes calculant le squelette pondéré sont nombreux et dépendent principalement du type de représentation de la forme : maillage polygonal, nuage de points, bitmap, surfaces implicites, surfaces paramétriques. Nous considérons que toutes les formes que nous traitons sont des maillages polygonaux. Il existe deux types d'algorithme pour les maillages polygonaux, les premiers travaillant directement sur le maillage ([CKM99]), les seconds transformant le maillage en nuage de points ([Hub96], [BO02], [AAH+07]). Ces derniers se distinguent notamment sur l'échantillonnage du maillage : répartition aléatoire uniforme pour [Hub96], un point par sommet pour

[AAH+07], échantillonnage adaptatif pour réduire l'erreur dans [BO02]. Etant donnés ces deux algorithmes, deux étapes de paramétrage sont possibles pour mettre en œuvre la sphérisation. La première étape consiste à paramétrer l'échantillonnage du maillage, non pas de manière automatique comme dans [BO02] mais semi-automatique. L'utilisateur régler ainsi la densité des points sur le maillage ainsi que la régularité de cette densité (plus de points sur les zones ou la courbure est faible. La deuxième étape consiste à extraire un sous-ensemble de sphères à partir du squelette pondéré calculé. Cette extraction est paramétrée sur deux axes. Le rayon minimum des sphères permet d'écarter les détails jugés inutiles de la forme. Un ensemble de paramètres plus complexe, basé sur le "Shock Scaffold" [LKG04], permet d'extraire les sphères selon leur relation au contour. En effet, Leymarie et al. Dotent le squelette pondéré d'une structure, le "Shock Scaffold", qui interprète dynamiquement le squelette pondéré comme le lieu des collisions entre les fronts d'onde issus du contour (pour reprendre l'image original de Blum pour construction de l'axe médian). Cette structure permet donc d'isoler les éléments critiques du squelette pondéré, de le décomposer en sous-ensembles homogènes, ce qui constitue un outil intéressant pour le comprendre et le traiter. Une troisième étape est encore possible pour affiner la sphérisation. Affiner ne veut pas dire ici se rapprocher du modèle géométrique mais retravailler la transformation en sphères. Cette étape consiste à relancer l'algorithme sur la différence entre la forme originale et la collection de sphères. Cela signifie un nouvel échantillonnage et une nouvelle sélection d'un sous ensemble du squelette pondéré. On retrouve ainsi une technique proche de celle utilisée par B. Chanclou & al [Cha96] pour modéliser des cailloux, technique qui s'est avérée très adaptée pour le rendu granuleux des cailloux.

3.8. Conclusion

L'étude de l'importation de géométries dans le cas particulier de la modélisation d'obstacles révèle la multiplicité des approches possibles. Nous avons exposé trois méthodes pour résoudre le problème géométrique. Ces trois méthodes sont complémentaires, elles ont chacune leurs propriétés propres et leurs conséquences sur le modèle physique. En effet, comme nous l'avons montré dans le §3.3, la transformation géométrique n'est qu'une étape de la modélisation d'obstacles. Les données spatiales qu'elles fournissent à MIMESIS sont une première proposition pour le placement des <SOL> et la définition des seuils des interactions concernées. Le processus de modélisation physique reprend alors le pas, le modélisateur ne se préoccupe plus seulement de l'obstacle mais du phénomène qu'il est en train de modéliser dans son entier. Les données importées demeurent une aide à cet objectif final, et non une fin en soi. C'est pourquoi leur exactitude géométrique n'a pas une grande importance. Ce qui est en revanche appréciable pour le modélisateur c'est de pouvoir définir sa transformation géométrique à partir de critères qui vont jouer dans la dynamique, comme par exemple la rugosité.

La méthode générale de modélisation d'obstacles que nous venons de présenter est cependant propre à l'utilisation d'interactions émettant des forces axiales et isotropes, d'où l'approximation de formes par des sphères. Cette limitation tient au respect du principe d'action-réaction, la partie suivante propose de la dépasser pour coder la forme au sein même de l'interaction.

4. Obstacles fixes : Interactions anisotropes pour le codage de la forme

4.1. Principe et champ d'application

Certains cas de modélisation présentent des géométries relativement simples mais qui sont synonymes de non-linéarités très fortes. L'exemple du choc d'une bille sur un angle droit ou même sur une surface plane est représentatif de ces cas de modélisation. On considère en effet une interaction dure avec une surface plane, très lisse, dont la normale présente une discontinuité très forte. Les techniques présentées dans la partie précédente ne peuvent pas modéliser d'interaction avec une surface plane et lisse, il y a toujours une rugosité résiduelle. On peut certes la diminuer en augmentant le nombre de sphères et/ou en augmentant leur rayon, mais l'objectif de modélisation s'en trouve alors modifié. La base de modules de MIMESIS se révèle donc insuffisante pour traiter ces cas de modélisation.

Les solutions proposées dans la partie précédente se basent toutes sur des interactions calculant une force axiale. La partie élastique de ces interactions se base sur un calcul de distance classique entre deux particules. Pour ces deux raisons, on peut qualifier ces interactions d'isotrope, car elles ne privilégient aucune direction de l'espace, elles ne se basent que sur la donnée des deux particules pour définir une distance et une direction de support pour les forces. Cette isotropie peut être interprétée comme une absence de géométrie au sein de l'interaction, ou tout du moins une influence réduite à son strict minimum. Nous proposons d'introduire plus de géométrie dans l'interaction en brisant l'isotropie. Les interactions physiques anisotropes proposent ainsi d'accroître l'influence de la géométrie dans le composant d'interaction. Ces interactions consistent dans le codage implicite des formes des obstacles fixes. Là où l'isotropie codait implicitement des sphères, l'anisotropie va permettre de coder d'autres géométries. Dans le composant d'interaction, l'anisotropie se manifeste à deux étapes : le calcul de la distance relative (et donc de la vitesse relative) et le choix d'une direction pour porter les forces. Ces deux étapes supposent une transformation géométrique commune, assurant la cohérence des forces calculées. Cette transformation porte sur le vecteur défini par les positions des deux masses en interaction. Contrairement aux composants d'interaction isotropes (c'est-à-dire tous les composants cités jusqu'à présent et détaillés en annexe 1), ces composants sont conçus exclusivement pour la modélisation d'obstacles. Les deux masses connectées par de telles interactions ne sont donc pas

équivalentes, l'une est fixe et représente l'inertie infinie de l'obstacle, alors que l'autre est mobile. Une telle dissymétrie permet de mieux appréhender ce que représente la transformation géométrique, et particulièrement le calcul de distance. Comme le suggère l'équation 9, le calcul de distance revient à définir une surface isopotentielle, la constante pouvant être remplacée par les différents seuils et longueurs au repos de l'interaction. Notons que si on prend la norme euclidienne pour f on retrouve une interaction isotrope, la surface isopotentielle est alors une simple sphère. Le gradient de la fonction f définit le support de l'unique force émise par une interaction anisotrope.

$$f(x - x_0, y - y_0, z - z_0) = \text{constante}$$

Équation 9 : Définition d'une surface isopotentielle

La fonction distance d'une interaction anisotrope est donc le paramètre complexe qui permet le codage implicite de la forme. Son introduction dans CORDIS-ANIMA est une opération délicate, car elle introduit de la géométrie à un haut niveau de complexité. Nous présentons dans la suite une famille de modules disponibles dans MIMESIS 4 qui correspond à un sous-ensemble élémentaire des interactions non isotropes. Nous présenterons ensuite différentes évolutions de cette famille et ce qu'elles impliquent comme paramétrage géométrique.

4.2. Description de l'existant dans MIMESIS 4

Toutes les interactions de MIMESIS 4 présentent trois versions anisotropes, chacune privilégiant une direction de l'espace (X, Y et Z). Les fonctions de définition de la distance sont élémentaires, comme on peut le voir dans l'équation 10 avec l'exemple des interactions privilégiant l'axe X.

$$f(x - x_0, y - y_0, z - z_0) = |x - x_0|$$

Équation 10 : Fonction distance privilégiant l'axe X

Les butées anisotropes (<BUTX>, <BUTY> et <BUTZ>) permettent de modéliser des contacts avec un plan "parfaitement lisse et de courbure nulle". Ces attributs géométriques se révèlent dynamiquement dans le phénomène élémentaire du rebond sur un sol élastique. En effet, on peut comparer trois modèles différents pour ce phénomène : le premier basé sur la technique "chapelet de masses", le deuxième utilisant une butée anisotrope et le troisième une butée simple. Dans ce dernier le seuil de la butée est très grand devant l'échelle du phénomène, simulant ainsi une courbure faible et une quasi horizontalité. Ces trois modèles produisent des versions très différentes du phénomène de rebond, et seul le modèle utilisant une butée anisotrope est réellement satisfaisant : le mouvement est limité par une ligne horizontale et la bille ne ralentit pas. Cet exemple prouve ainsi la validité des interactions anisotropes en tant qu'alternative aux techniques développées dans la partie précédente. Le §5.2 détaille des expérimentations élémentaires avec ces trois modèles.

Cependant, les variantes anisotropes des interactions dans MIMESIS 4 restent prédéterminées et très limitées : 3 directions de projection, pas de contrôle sur les

autres directions (les plans sont infinis). Faire sauter ces limitations, c'est introduire plus de généricité dans l'anisotropie, et donc plus de la modélisation géométrique dans le paramétrage de l'interaction, au cœur même du processus de modélisation physique. Cette généricité peut donc être dommageable au processus de modélisation physique car elle le dénature. Sans préjuger de ces dommages, nous proposons dans le paragraphe suivant une famille de modules permettant un réglage plus fin du codage de la forme dans l'interaction.

4.3. Evolutions

La modélisation de certains obstacles présentant une géométrie relativement simple, comme des escaliers, requiert une extension des modules anisotropes. Les modules présentés dans le paragraphe ne sont en effet pas suffisants, et les solutions proposées dans la partie précédente ne sont pas adaptées. Du point de vue du formalisme CORDIS-ANIMA, le cas des obstacles se prête bien à une intégration modulaire, car c'est un cas, rare, où une sous partie de réseau s'identifie à un objet monolithique. Une vision intégrée permet également d'unifier les paramètres tout en se connectant à un grand nombre de <MAT>. Nous proposons donc les modules <OBST> pour OBSTacle, qui présentent un point M et n points L. Les n points L connectent l'obstacle à tous les modules <MAT> qui sont en interaction avec lui. Les modules <OBST> dérivent des modules <LIA> déjà existants, il existe un sous-type de <OBST> pour chacune des interactions existantes. Comment introduire plus de généricité dans l'anisotropie ? Nous avons choisi de partir des modules présentés au paragraphe précédent. Etendre le paramétrage de l'anisotropie c'est justement déplacer le choix d'une direction de projection depuis le type (TYPE_X, TYPE_Y, TYPE_Z) vers les paramètres physiques, donc de PSQL vers PSQN. Les modules <OBST> présentent donc un premier paramètre supplémentaire : la direction de projection, donnée par un vecteur unitaire. Avec ce paramètre on définit à la fois le calcul de la distance et le support des forces émises par le module. On introduit donc pour la première fois dans un module CORDIS-ANIMA un paramètre vectoriel. Le gain est cependant peu satisfaisant, on peut modéliser des plans inclinés, mais des obstacles simples comme des parallélépipèdes ou des escaliers restent impossibles à modéliser. On cherche donc à limiter l'influence de l'obstacle dans le plan orthogonal à la direction de projection. La zone d'influence qui doit être délimitée dans l'espace définit une forme, on progresse donc vers plus de modélisation géométrique, car il faut définir cette forme. Les extensions peuvent ainsi s'accumuler : délimitation isotrope dans le plan orthogonal (pour définir des cylindres), délimitations non isotrope de forme rectangulaires (pour définir des parallélépipède), délimitations anisotropes de formes quelconque (pour définir des prismes droits quelconques). Plus on tend vers la généricité, plus la géométrie prend de l'importance, et cette marche ne peut s'arrêter que quand un modeleur géométrique complet aura été intégré dans MIMESIS. Or, l'intégration d'un logiciel complexe dans un logiciel également déjà complexe n'est bon ni pour l'utilisabilité ni pour le processus de modélisation physique, qui perd alors son intégrité. Nous en concluons que tout paramétrage géométrique d'une interaction doit se faire hors de MIMESIS, y compris le plus simple (choix d'une direction de projection). La spatialité n'apparaît dans l'interaction qu'à travers les seuils.

Le paramétrage géométrique des modules anisotropes doit être effectué dans une application tierce, à l'image des applications de transformation géométrique décrites dans la partie précédente. On conserve donc l'idée de modules <OBST>, mais on externalise le paramétrage géométrique. Ce glissement est d'ailleurs beaucoup plus cohérent avec la notion de module-obstacle en tant qu'objet, dans la mesure ou les propositions précédentes s'apparentaient davantage à des bouts d'objets, analogue aux facettes d'un modèle polygonal. La faisabilité d'une application permettant de transformer n'importe quelle forme en module <OBST> n'a pas été étudiée pendant nos travaux. Cependant les travaux sur la représentation de formes avec des champs de distance [FPR+00, FSG03], et leur application à la détection de collision [BMF03, FL01], permettent d'imaginer de possibles solutions. Notons que [FL01] s'intéresse à la détection de collision pour des objets faiblement déformables, et dont les champs de distance sont réévalués à chaque déformation. Par ailleurs, les champs de distance peuvent être définis à partir de plusieurs points, voire à partir de squelettes, et dans ces cas la transposition en modules <OBST> est compromise si on ne revoit pas leur structure.

4.4. Conclusion

Nous avons montré l'utilité des modules <OBST> et leur nécessaire intégration à MIMESIS, au moins sous une forme minimale permettant d'obtenir l'équivalent des modules type <BUTX>. Leur spécification, du point de vue du simulateur comme du modeleur, n'est pas arrêtée, les recherches doivent être poursuives sur la relation avec les champs de distance. D'autres questions restent encore en suspens, comme la manipulation des modules <OBST> dans MIMESIS (déplacement, rotation, dilatation). La complexité des formes que l'on souhaite introduire dans le modèle physique demeure également une question de modélisation importante pour le praticien de l'Art du mouvement. Ouvrir la porte vers l'importation de structures géométriques complexes repose ainsi la question de l'objectif du modélisateur. Si celui-ci choisit MIMESIS, c'est pour centrer son étude sur le mouvement et non sur la forme. C'est pourquoi nous ne considérons pas comme prioritaire la mise en place d'une application permettant de transformer n'importe quel modèle géométrique en module <OBST>, d'autant plus que les solutions proposées au chapitre précédent demeurent une solution intéressante. Elles s'intègrent en effet plus naturellement au processus de modélisation physique et permettent d'introduire des géométries quelconques. Cependant, les interactions anisotropes présentent deux avantages majeures : 1) la modélisation de surfaces parfaitement planes 2) un paramétrage physique plus facile à maîtriser. Nous allons montrer dans les expérimentations comment les solutions basées sur la sphérisation peuvent en effet poser problème sur ce deuxième point.

5. Expérimentations

5.1. Sphérisation

Des expérimentations 2D ont été réalisées sur des formes simples : rectangle, triangle, "boomerang". Le modèle physique est identique dans les trois cas, il s'agit du rebond d'une bille sur un obstacle fixe. Les transformations de ces formes en sphères, puis en <SOL> et <BUT> bien placés et paramétrés a été faite "à la main" :

1. Echantillonnage des formes
2. Calcul du graphe de Voronoï
3. Placement des sphères sur les sommets du graphe de Voronoï et réglage du rayon
4. Placement des <SOL> sur les sommets du graphe de Voronoï
5. Réglage des seuils des interactions de butée en fonction des rayons des sphères

La construction de la transformation a confirmé comment l'échantillonnage des formes était décisif. Un point très positif est que l'on anticipe assez facilement l'effet d'un changement dans l'échantillonnage. Par exemple, mettre de fortes densités de points dans les zones à courbure forte génère beaucoup de petites sphères. A l'inverse, ne pas mettre de points sur les singularités permet d'arrondir les angles en exagérant la forme.

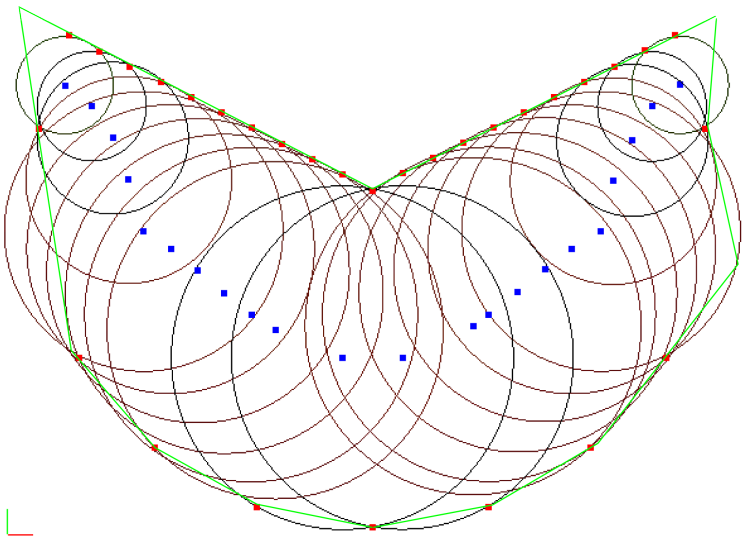


Figure 94 : Un contour (en vert), discrétisé (en rouge) et sphérisé (centres des sphères en bleu)

Au niveau des conséquences physiques d'un telle modélisation de l'obstacle, le modélisateur est confronté à une difficulté inaliénable de la méthode. En effet la superposition géométrique des sphères se manifeste dans le modèle physique par une combinaison complexe des élasticités et des viscosités des interactions entre l'obstacle et les masses qui sont en interaction avec lui. Dans un modèle basique de bille rebondissant sur un pavé, on pourra donc observer des rebonds plus ou moins importants suivant la zone du pavé et l'angle d'incidence de la bille. On remarque également le rendu granuleux des obstacles, qui présentent comme des petits creux à leur surface. La rugosité résultante n'est pas très facilement maitrisable lors de la transformation géométrique. En effet, on maitrise plus le nombre de sphères que leur rayon, le modélisateur ne dispose que d'un seul levier pour jouer sur ce paramètre.

5.2. Rebond sur un sol "lisse" : comparaison de trois méthodes de modélisation

Nous comparons dans cet exemple trois méthodes pour modéliser un sol horizontal rigide en interaction avec une bille. L'objectif est de montrer les différents effets propres à chaque méthode et d'observer l'effet d'un même changement de paramètre sur chacune des trois. Les modèles ont été réalisés avec MIMESIS 4 avec une fréquence de simulation de 1050 Hz.

La première méthode est réalisée à l'aide d'une interaction anisotrope <BUTY> modélisant ainsi un sol parfaitement plat et lisse. La deuxième méthode est basée sur le principe du chapelet de masses. Le rapport de rugosité que nous avons choisi est de 1/10, il peut être considéré comme assez faible. La troisième méthode est une sphérisation élémentaire, il consiste géométriquement à approximer le sol par un cercle de très grand diamètre. Le modèle est donc constitué d'un <SOL> en interaction de butée avec une <MAS>, le <SOL> étant placé très bas et le seuil de la butée très grand

(de l'ordre de 10^6). Nous avons réalisé ces trois méthodes au sein d'un même modèle, de manière à comparer les trois mouvements sur un même film. Les conditions initiales de la <MAS> par rapport au sol sont identiques dans les trois méthodes. Dans la première expérience les butées des trois méthodes sont réglées avec une raideur de 0.1 et une viscosité de 0.01. Dans la seconde expérience, les trois méthodes sont reprises à l'identique mais avec une viscosité de 0.001. Les résultats sont illustrés dans les figures 95 et 96 qui montrent chacune quatre instantanés de la simulation. La méthode utilisant une interaction anisotrope représente la bille comme un disque rouge et le sol comme une ligne horizontale infinie. La méthode du chapelet de masses représente la bille comme un triangle jaune et le sol comme un chapelet de cercles de rayon égal au seuil des butées. La méthode de sphérisation élémentaire représente la bille comme un carré vert et le sol comme un cercle ayant pour centre la position du <SOL> et pour rayon le seuil de la butée. Sur les figures 95 et 96, l'arc de cercle représenté apparaît comme une droite.

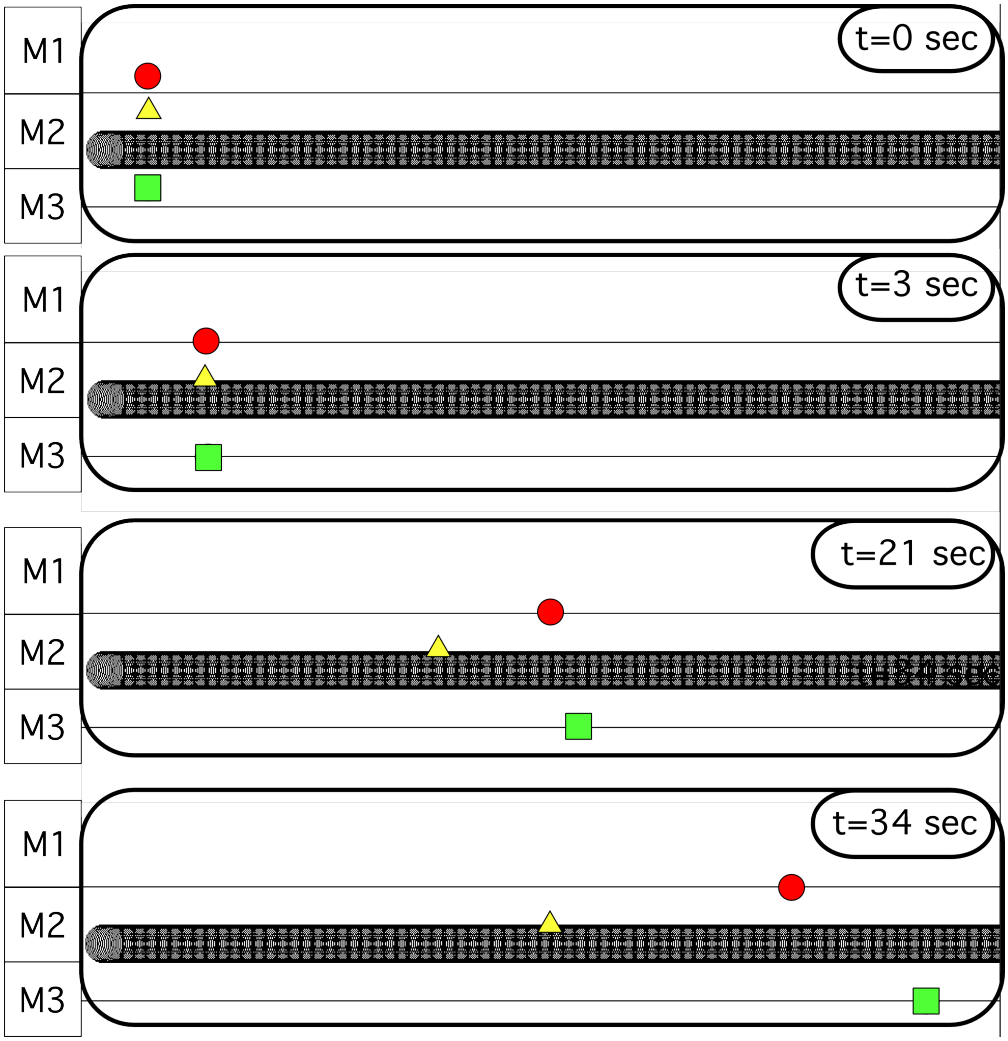


Figure 95 : Rebond sur un sol lisse : résultats avec trois méthodes de modélisation. Viscosité forte de l'interaction entre la bille et le sol (0.01)

Dans la première expérience (viscosité forte, figure 95), on constate que la rugosité induite par la méthode du chapelet de masses est quand même importante. En effet, on constate une forte décélération et la masse reste d'ailleurs à la même position après la dernière image. A l'inverse, on constate également que malgré une courbure très faible, la sphérisation induit une accélération due à la gravité. Visuellement le sol est plat, mais dynamiquement il est légèrement courbé. La modélisation de l'interaction avec le sol par une <BUTY> (interaction anisotrope) ne montre aucun effet de rugosité, la bille ne décélère pas et n'accélère pas non plus.

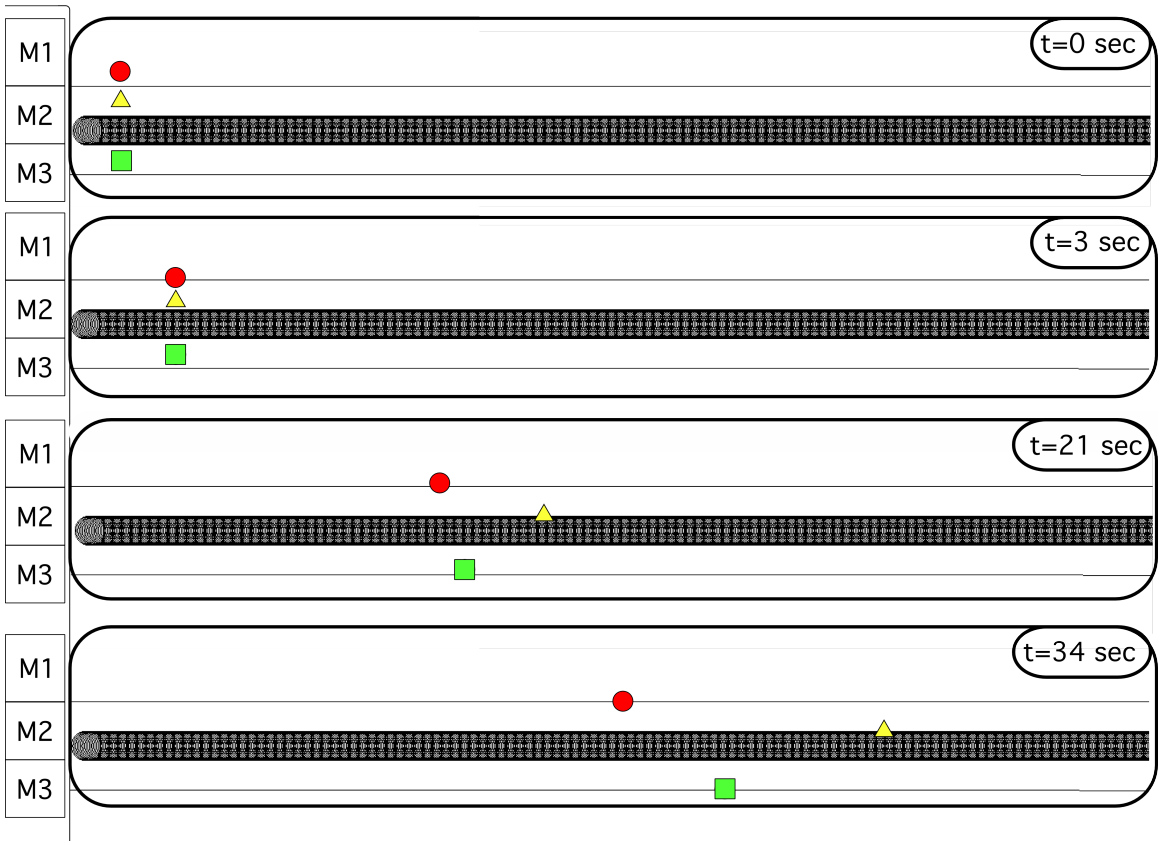


Figure 96 : Rebond sur un sol lisse : résultats avec trois méthodes de modélisation. Viscosité faible de l'interaction entre la bille et le sol (0.001)

La seconde expérience (viscosité faible, figure 96) montre plusieurs éléments par rapport à la première. Le sol paraît plus dur, puisqu'on observe encore des rebonds à la troisième seconde. Mais l'observation la plus intéressante concerne la méthode du chapelet de masses. En effet contrairement à la première expérience, l'interaction avec le sol semble accélérer la masse, qui évolue plus rapidement que dans les deux autres méthodes. On constate donc que la même rugosité géométrique peut avoir deux effets dynamiques opposés selon les paramètres physiques que l'on choisit.

Nous avons également expérimenté la méthode du chapelet de masses avec une rugosité géométrique plus faible. On constate immédiatement la relation entre rugosité géométrique et les paramètres physiques des butées, qui définit en quelque sorte l'élasticité et la viscosité résultante du sol. En effet, plus la rugosité géométrique est faible, plus grand sera le nombre de butées simultanément actives. Il n'y pas de relation exacte qui permettrait de définir avec précision l'élasticité et la viscosité du sol. Une fois

la rugosité géométrique fixée, il reste donc un travail fin de paramétrage physique à effectuer pour arriver au résultat souhaité.

Ces expérimentations n'invalide pas une méthode en particulier par rapport à l'objectif d'une interaction avec un sol lisse. Au contraire, elle montre la diversité des approches et la nécessité d'expérimenter longuement chacune d'elle sans négliger la phase PSQN. Il n'existe pas de "bonne" méthode, il existe différents objectifs de modélisation et différentes méthodes permettant d'atteindre son objectif.

5.3. Scénario de collaboration horizontale

Nous présentons un scénario très simple pour la définition des conditions initiales. L'objectif est de placer un grand nombre de masses sur un segment. Le modélisateur a besoin de plusieurs essais successifs pour trouver le bon segment, il doit donc simuler le modèle après chaque modification des conditions initiales. Nous proposons que l'utilisateur opère selon le protocole ci-après (toutes ces actions se situent dans la phase CI) :

1) Création graphique d'un objet segment.
2) Affectation d'un os-label à cet objet.
3) Création d'une fonction. Le script présente une seule procédure plaçant les masses sur le segment.
4) Exécution du script.

Une fois ce premier placement réalisé, l'utilisateur peut simuler son modèle pour en observer l'effet. Il revient ensuite à la phase CI pour modifier graphiquement le segment, applique ces modifications en exécutant la fonction, simule le modèle, et ainsi de suite jusqu'à obtenir l'effet souhaité.

Cet exemple élémentaire montre comment une fonction associée à un objet spatial constitue un instrument de placement performant. L'utilisation d'un script plutôt qu'une contrainte directe permet de complexifier le placement, par exemple en créant une ligne brisée en dupliquant et en déplaçant les duplicata. Par ailleurs, l'utilisation d'expressions de sélection de labels adaptées dans le script lui permet d'être directement opérationnel en cas de changements dans la topologie du réseau.

6. Conclusion

Nous avons abordé trois problématiques ayant trait aux conditions initiales et à la spatialité dans les modèles physiques masses-interactions. La première problématique nous a permis d'introduire les instruments adéquats pour le traitement de ces problèmes dans MIMESIS V. Les objets spatiaux sont certes basiques mais ils sont le support de manipulations plus complexes, soit directement dans MIMESIS en tirant parti de la puissance du script, soit en important des données depuis un logiciel tiers. Nous avons donc traité dans un deuxième temps des différentes voies possibles pour transformer des données géométriques en données exploitables dans MIMESIS, montrant ainsi comment cette transformation devient partie intégrante du processus de modélisation et ne peut être entièrement automatisée. Nous avons enfin examiné l'introduction de la spatialité au sein des composants d'interaction. Cette voie de modélisation introduit la géométrie au cœur du formalisme CORDIS-ANIMA. Les interactions anisotropes révèlent un type de complexité intéressant, liée aux fortes discontinuités spatiales, et qui ne peut être modélisé qu'avec leur concours. Ces différentes méthodes d'introduction de la géométrie sont autant de possibilités pour le modélisateur, elles doivent le servir dans son objectif de modélisation, et non occulter cet objectif en faisant passer la géométrie au premier plan.

Nous avons évoqué le sujet des vitesses initiales. Leur manipulation s'intègre à la phase CI de la manière semblable aux positions initiales, leur nature vectorielle devant être bien évidemment prise en compte. Cette prise en compte peut notamment se traduire par la manipulation séparée du module et de la direction d'un vecteur vitesse, plutôt que ses coordonnées. Le besoin de définir un grand nombre de vitesses initiales hétérogènes est certainement le sujet le plus épineux, mais peu de cas de modélisation le requièrent.

Les problèmes de spatialité nous ont amené à évoquer les <MAT1D> et <MAT2D>. Ces modules sont des <MAT> tridimensionnels, avec des degrés de liberté respectivement restreint à une et deux dimensions. Ils modélisent ainsi une contrainte géométrique, de manière analogue aux interactions anisotropes. Leur intégration pose donc des problèmes équivalents, notamment dans le cas d'une version générique de ces modules.

Une approche générique pour l'introduction de la géométrie dans MIMESIS est finalement antinomique au processus de modélisation physique. En effet toute démarche allant dans ce sens, quel qu'en soit l'objectif, équivaut à insérer la géométrie comme un bloc autonome. On aboutirait alors à disposer d'un modeleur géométrique dans MIMESIS V. Un tel emboitement serait non seulement très lourd à gérer pour l'utilisateur mais surtout détournerait MIMESIS de sa fonction première de modeleur physique particulaire.

Dans l'objectif de garder la modélisation physique au centre du processus de création, notre philosophie générale est donc d'introduire la géométrie localement pour répondre à des problèmes géométriques de manière fonctionnellement adaptée au sein d'une cartographie de fonctionnalités structurantes pour la modélisation physique : PSQN, PSQL, Conditions initiales, Habillage. Les instruments de placement et les processus d'importation rentrent dans cette logique d'intégration fonctionnelle.

Chapitre V. Conclusion et perspectives

MIMESIS propose de modéliser pour le mouvement visuel avec CORDIS-ANIMA. Comment donner à des personnes désireuses d'explorer la complexité qu'offre potentiellement CORDIS-ANIMA les moyens de le faire ? Nous avons voulu aborder ce problème sous trois angles différents : 1) Les non-linéarités dans CORDIS-ANIMA 2) un environnement interactif de modélisation pensé pour la complexité 3) les conditions initiales et le rapport à la géométrie. Notre travail pendant ces quatre années a donc consisté à explorer ces trois thématiques. Nous avons ainsi pu mieux appréhender ce qui fait la richesse et la complexité de la modélisation par réseaux masses-interactions. Cet effort s'est traduit par des outils qui tentent de mettre au centre la complexité sans que leur manipulation elle-même ne soit complexe.

Premier outil au centre de nos préoccupations, le contrôle paramétrique n'ouvre pas la voie vers plus de complexité mais vers des complexités différentes. Il donne à explorer des possibilités de modélisation inédites, particulièrement à travers la coopération de modèles 1D et 3D. Cette coopération n'est d'ailleurs qu'un cas particulier de couplage entre modèles de dimensionnalités différentes. Les autres cas, notamment le couplage entre modèle sonore et modèle 3D, posent de nombreux problèmes, tant au niveau de la modélisation qu'au niveau de l'interface. C'est donc un champ d'étude important que nous avons inauguré, dont la relation MIMESIS-GENESIS constitue l'un des piliers. Cette relation est explorée dans les travaux de thèse en cours de Maria Christou d'un point de vue artistique et cognitif. Le contrôle paramétrique en ligne dans CORDIS-ANIMA a également apporté une autre manière d'aborder le contrôle de modèle physique, notamment à travers l'exemple élémentaire du sauteur. Le contrôleur, construit entièrement par le modélisateur, illustre une démarche de conception radicalement différente des contrôleurs pilotés par l'objectif. Un champ d'expérimentation est ouvert, et la question de sa relation avec les outils classiques de contrôle moteur (algorithmes génétiques, réseaux de neurones, algorithmes comportementaux) est une question ouverte à explorer. Nous espérons que les briques simples que nous avons apportées dans CORDIS-ANIMA permettront d'ouvrir un nouveau champ de recherche alliant modèles physiques et modèles logiques et bio-logiques.

L'augmentation et l'enrichissement du formalisme CORDIS-ANIMA ne prennent réellement sens qu'au sein d'un environnement complet et interactif de modélisation. Ce type d'environnement, comme l'avait justement constaté Matthieu Evrard dans sa thèse [Evr09], n'existe pas. Les environnements de type RealFlow [Rea] peuvent certes apparaître comme dédiés à la modélisation physique particulaire, mais ils sont restreints à une classe de phénomènes bien déterminée et suivent une logique dans le processus de création radicalement différente de MIMESIS. Notre démarche est de proposer un environnement entièrement dédié à la modélisation de réseaux masses-interactions, un environnement pensé pour la complexité et la généricité que porte en lui ce système de modélisation. Cette démarche, ébauchée dans MIMESIS 4, nous l'affirmons dans MIMESIS V en y apportant des bases solides, communes à l'ensemble de la suite logicielle. Le principal défi auquel nous avons été confrontés est la collaboration entre ce que nous avons appelé les modalités script et graphique. Nous avons abordé ce problème en accordant une place centrale au script, plutôt que de le considérer comme une option accessible uniquement aux utilisateurs avancés. Ce rapport entre les deux modalités a fait émerger des manières nouvelles d'envisager leur collaboration, inédites à notre connaissance. Les problèmes soulevés sont nombreux, notamment les difficultés liées à la collaboration verticale et le rôle de liant que joue le système de labellisation. Ces

problèmes peuvent maintenant être étudiés sur une base solide, la réification du script. Mais avant même d'aborder ces problèmes, ces concepts d'interface doivent être mis en œuvre et être confrontés à l'utilisation. Le projet MIMESIS continue donc, ces travaux de thèse marquent le début d'une seconde étape, qui doit se poursuivre avec l'implantation de toutes les fonctionnalités que nous avons ici décrites. MIMESIS V constituera alors un environnement de modélisation pour l'art du mouvement, à destination des praticiens de l'art du mouvement, qu'ils soient d'origine scientifique ou artistique. Mais MIMESIS V est aussi un terrain d'expérimentation sur les outils de modélisation de réseaux masses-interactions pour le mouvement visuel. En effet, nous avons ouvert plusieurs champs de recherche, que ce soit sur la collaboration entre modalité graphique et script, mais aussi sur les différentes représentations graphiques du réseau. Le problème de la représentation conjointe de réseaux 1D et 3D a été évoqué, et il mérite d'être approfondi pour aller vers un outil de modélisation consacré à la multisensorialité.

Le problème des conditions initiales a été abordé avec toujours le même objectif de la gestion de la complexité. Ce problème très large a été circonscrit à l'étude des outils pour le prototypage d'obstacles fixes et indéformables. D'un autre point de vue, on peut le considérer comme le problème du placement des modules <SOL> et du paramétrage des interactions entre les <SOL> et les autres modules <MAT> du réseau. Nous avons alors constaté que les conditions initiales, qui ont directement trait à la spatialité, posent le problème de la relation entre modèles physiques et modèles géométriques. Cette relation est complexe et délicate à gérer pour le concepteur d'interface, car il doit arriver à un compromis entre des outils suffisamment puissants pour introduire la géométrie et une généricité trop importante qui déplacerait le centre d'intérêt du modélisateur du mouvement vers la forme.

La relation entre forme et mouvement a donc été abordée en amont du processus de modélisation. Mais il reste tout un pan du problème, situé en aval du processus de modélisation, que nous n'avons qu'évoqué à travers la phase d'habillage. Ce problème est actuellement abordé sous deux angles très différents. Le premier angle constitue le sujet de la thèse de Kevin Sillam, qui traite d'une méthode dynamique d'habillage d'un modèle physique particulaire, l'écran d'épingles [HL02, SEL07]. Le second angle fait l'objet d'un projet ANR intitulé Dynamé, dont l'un des trois axes est consacré à l'introduction de la topologie spatiale dans le processus d'habillage. Un modèle topologique, basé sur le formalisme G-cartes [Lie89], permet de modifier le modèle géométrique d'habillage en fonction de la simulation. Une telle introduction permettrait d'habiller de manière adaptée des phénomènes complexes comme des fractures, et de manière générale tous les phénomènes à topologie variable.

Le chemin vers un environnement complet de modélisation physique particulaire pour l'art du mouvement, mais aussi pour un art multisensoriel, est encore long. Nous avons posé des jalons, il faut maintenant continuer sur la voie tracée par le groupe ACROE-ICA depuis maintenant 30 ans, en diffusant ces outils, en accompagnant cette diffusion par une formation adaptée et en continuant la recherche sur ces outils.

Glossaire

CI	Conditions Initiales. Troisième phase du processus de modélisation d'un réseau CORDIS-ANIMA consistant à déterminer les positions et vitesses initiales des modules <MAT>
CORDIS-ANIMA	Formalisme de modélisation et de simulation d'objets physiques virtuels
ESL	Expression de Sélection de Label. Chaîne de caractères permettant de sélectionner une liste ordonnée de labels.
ESEP	Expression de Sélection de points d'Entrée de Paramètre.
ESP	Expression de Sélection de Points de communication
ESSP	Expression de Sélection de points de Sortie de Paramètre.
GENESIS	Environnement de modélisation de réseaux CORDIS-ANIMA pour la création musicale
Label	Nom, non nécessairement unique, donné par l'utilisateur à un module.
Label auto	Nom unique donné à un module lors de sa création
LCM	Liaison Conditionnelle à Mémoire. Type de <LIA> dont l'algorithme est défini par un automate d'états à mémoire.
LEP	Label de liste de points d'Entrée de Paramètre
<LIA>	Famille de modules CORDIS-ANIMA comportant deux points L et respectant le principe d'action réaction.
LIC	Liaison Conditionnelle. Type de <LIA> dont l'algorithme est défini par un automate d'état.
LLM	Liaison Linéaire par Morceaux.
LPC	Label de liste de Points de Communication
<MAT>	Famille de modules CORDIS-ANIMA comportant un point M.
<MCP>	Familles des Modules de Contrôle de Paramètres de CORDI-ANIMA
<MES>	Famille des modules de MESure de CORDIS-ANIMA
Modèle CORDIS-ANIMA	Objet formé par un réseau CORDIS-ANIMA et son habillage
Modèle CORDIS	
Module CORDIS-ANIMA Module CORDIS	Objet élémentaire de CORDIS-ANIMA composé d'un ensemble de points de communication, d'un ensemble de variables d'état, d'un algorithme, et des paramètres de cet algorithme.
Os-label	Nom, non nécessairement unique, donné à un objet spatial.
PNSL	Physical Network Scripting Language. Langage de script de

	Genesis[3] et MIMESIS V
Point de communication	Interface de communication faisant transiter une force dans un sens et une position dans l'autre.
Point EP	Point d'Entrée de Paramètre. Interface de communication des modules CORDIS-ANIMA permettant de contrôler la valeur de leurs paramètres.
Point L	Point de communication envoyant une force et recevant une position
Point M	Point de communication envoyant une position et recevant une force
Point SP	Point de Sortie de Paramètre. Interface de communication émettant un paramètre.
Projet	Ensemble des objets d'intérêt créés par l'utilisateur dans MIMESIS V : modèle CORDIS, objets auxiliaires (fonctions, objets spatiaux...)
PSQL	PréStructuration QuaLitative. Phase du processus de modélisation consistant à créer la topologie du réseau
PSQN	PréStructuration QuaNtitative. Phase du processus de modélisation consistant à définir les paramètres physiques des modules.
<REF>	Interaction visco-élastique élémentaire (définie au chapitre I §3.4 et dont l'algorithme est donnée par l'équation 4)
WIMP	Window, Icon, Menu, Pointer. Type d'interface basé sur la manipulation graphique, à base de fenêtres, d'icônes, de menus et d'activité de pointage.

Bibliographie

[AAH+07] O. Aichholzer, F. Aurenhammer, T. Hackl, B. Kornberger, M. Peternell, and H. Pottmann. Approximating boundary-triangulated objects with balls. In *roc. 23rd European Workshop on Computational Geometry, EWCG'07*, Graz, Austria, 2007.

[ACF+07] J. Allard, S. Cotin, F. Faure, P. J. Bensoussan, F. Poyer, C. Duriez, H. Delingette, and L. Grisoni B. Sofa – an open source framework for medical simulation. *Medicine meets virtual reality (MMVR'15)*, pages 13–18, 2007.

[ACK01] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust, unions of balls, and the medial axis transform. *Comput. Geom.*, 19(2-3):127–153, 2001.

[AG85] W. Armstrong and M. Green. The dynamics of articulated rigid bodies for purposes of animation. In *GRAPHICS' INTERFACE 1985*, pages 407–415, 1985.

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[BB88] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 179–188, New York, NY, USA, 1988. ACM.

[BC00] David Bourguignon and Marie-Paule Cani. Controlling anisotropy in mass-spring systems. In Nadia Magnenat-Thalmann, Daniel Thalmann, and Bruno Arnaldi, editors, *11th Eurographics Workshop on Computer Animation and Simulation, EGCAS 2000, August, 2000*, Springer Computer Science, pages 113–123, Interlaken, Suisse, 2000. Springer-Verlag.

[Ber94] Niels Ole Bernsen. Foundations of multimodal representations: A taxonomy of representational modalities. *Interacting with Computers*, 6(4):347–371, 1994.

[BL00] Michel Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-wimp user interfaces. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 446–453, New York, NY, USA, 2000. ACM.

[Bli82] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.

[Blu67] Harry Blum. A transformation for extraction new descriptors of shape. In Weiant Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, Massachusetts, 1967.

[BMF03] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 28–36, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[BO02] Gareth Bradshaw and Carol O'Sullivan. Sphere-tree construction using dynamic medial axis approximation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 33–40, New

York, NY, USA, 2002. ACM.

[BW76] N. Burtnyk and M. Wein. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Commun. ACM*, 19(10):564–569, 1976.

[Cad02] Claude Cadoz. The Physical Model as Metaphor for Musical Creation "pico..TERA", a piece entirely generated by physical model. In *International computer music conference proceedings 2002 ICMC 2002*, pages 305–312, Goteborg Suède, 2002.

[Cas10] Julien Castet. Conception et Réalisation de plates-formes de simulation synchrone multisensorielle interactive - L'interaction instrumentale et les réalités virtuelles -. PhD thesis, Grenoble Institut National Polytechnique, 2010.

[CC02] Nicolas Castagné and Claude Cadoz. GENESIS : a friendly musician-oriented environment for mass-interaction physical modeling. In *International Computer Music Conference Proceedings ICMC 2002*, pages 330–337, Goteborg Suède, 2002.

[CCA+09] Nicolas Castagné, Claude Cadoz, Ali Allaoui, and Olivier Tache. G3 : GENESIS software environment update. In *Proceedings of the international computer music conference 2009 International Computer Music Conference (ICMC)*, pages 407–410, Montréal Canada, 07 2009.

[CCP82] T. W. Calvert, J. Chapman, and A. Patla. Aspects of the kinematic simulation of human movement. *IEEE Comput. Graph. Appl.*, 2(9):41–50, 1982.

[Cha04] Joan Chaumont. *Mimesis-forme : Etude logicielle pour un mapping de formes sur un modèle physique particulaire*. Master's thesis, Institut National Polytechnique de Grenoble, 2004.

[Cha96] Benoit Chanclou. *Modélisation physique de véhicules tout-terrain : application à la simulation dynamique de robots mobiles d'intervention en milieu naturel*. PhD thesis, Université Claude Bernard (Lyon 1), Lyon, 1996.

[CKM99] Tim Culver, John Keyser, and Dinesh Manocha. Accurate computation of the medial axis of a polyhedron. In *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*, pages 179–190, New York, NY, USA, 1999. ACM.

[CLF84] Claude Cadoz, Annie Luciani, and Jean-Loup Florens. Responsive input devices and sound synthesis by simulation of instrumental mechanisms : The cordis system. *Computer Music Journal*, 8:60–73, 1984.

[CLF93] Claude Cadoz, Annie Luciani, and Jean-Loup Florens. Cordis-anima : a modeling and simulation system for sound and image synthesis. *Computer Music Journal*, 17:10–29, 1993.

[CLF94] Claude Cadoz, Annie Luciani, and Jean-Loup Florens. Physical models for music and animated image. the use of CORDIS-ANIMA in ESQUISSES a music film by ACROE. In *Proceedings of the International Computer Music Conference 1994*, pages 11–18, Aarhus Danemark, septembre 1994.

[CLH96] Benoit Chanclou, Annie Luciani, and Arash Habibi. Physical models of loose soils dynamically marked by a moving object. In *CA '96: Proceedings of the Computer Animation*, page 27, Washington, DC, USA, 1996. IEEE Computer Society.

[CMN80] Stuart K. Card, Thomas P. Moran, and Allen Newell. The keystroke-level model for user perfor- mance time with interactive systems. *Commun. ACM*, 23(7):396–410, 1980.

[CN94] Joëlle Coutaz and Laurence Nigay. Les propriétés care dans les interfaces multimodales. In *Actes de la conférence IHM'94, Lille*, pages 7–14, 1994.

[CNS+95] Joëlle Coutaz, Laurence Nigay, Daniel Salber, A. Blandford, J. May, and R. Young. Four easy pieces for assessing the usability of multimodal interaction: The CARE properties. In S. A. Arnesen and D. Gilmore, editors, *Proceedings of the INTERACT'95 conference*, pages 115–120, Lillehammer, Norway, 1995. Hall Publ.

[CP04] Matthieu Chabanas and Emmanuel Promayon. Physical model language: Towards a unified representation for continuous and discrete models. In S. Cotin and D. Metaxas, editors, *proceedings of Medical Simulation: International Symposium, ISMS 2004*, Lecture Notes in Computer Science, pages 256–266, Cambridge, MA, USA, 2004.

[DC96] Mathieu Desbrun and Marie-Paule Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. In R. Boulic and G. Hegron, editors, *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pages 61–76. Springer-Verlag, August 1996.

[ELC06] Matthieu Evrard, Annie Luciani, and Nicolas Castagné. Mimesis : Interactive interface for mass-interaction modeling. In Nadia Magnenat-Thalmann & al., editor, *Proceedings of CASA 2006*, pages 177–186, Geneva, July 2006.

[Evr09] Matthieu Evrard. *MIMESIS, un environnement de conception et de simulation de modèles physiques particulaires masses – interactions CORDIS-ANIMA pour l'animation : du mouvement généré à l'image du mouvement*. PhD thesis, Grenoble Institut National Polytechnique, 2009.

[FC90] Jean-Loup Florens and Claude Cadoz. Modèles et simulation en temps réel de corde frottée. *Le Journal de Physique Colloques*, 51(C2):4, 1990.

[FL01] Susan Fisher and Ming C. Lin. Deformed distance fields for simulation of non-penetrating flexible bodies. In *Proceedings of the Eurographic workshop on Computer animation and simulation*, pages 99–111, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[FM95] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical models and image processing*, 58:23–30, 1995.

[FPR+00] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 249–254, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[FRL+86] Jean-Loup Florens, Aimé Razafindrakoto, Annie Luciani, and Claude Cadoz. Optimized real time simulation of objects for musical synthesis and animated image synthesis. In *Proceedings of the International Computer Music Conference 1986*, pages 65–70, 1986.

[Fro96] David Frohlich. Direct manipulation and other lessons. In M.G. Hellander,

T.K. Landauer, and P.V. Prabhu, editors, *Handbook of HCI : Second completely revised edition*, chapter 22, pages 463–488. Elsevier Science, Amsterdam, 1996.

[FSG03] Arnulph Fuhrmann, Gerrit Sobottka, and Clemens Groß. Distance fields for rapid collision detection in physically based modeling. In *Proceedings of Graphicon 2003*, Moscow, Russia, 2003.

[GM77] R.A. Gingold and J.J. Monaghan. Smoothed particle hydrodynamics - theory and application to non-spherical stars. *Royal Astronomical Society, Monthly Notices*, 181:375–389, November 1977.

[GM85] Michael Girard and A. A. Maciejewski. Computational modeling for the computer animation of legged figures. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 263–270, New York, NY, USA, 1985. ACM.

[Gou71] H. Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, 20(6):623–629, 1971.

[Gre73] Donald Greenspan. *Discrete Models*. Addison-Wesley, 1973.

[Gre97] Donald Greenspan. *Particle Modeling*. Birkhauser Editors, 1997.

[Gro] Groboto. http://www.braid.com/groboto-site/

[GT95] Radek Grzeszczuk and Demetri Terzopoulos. Automated learning of muscle-actuated locomotion through control abstraction. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 63–70, New York, NY, USA, 1995. ACM.

[GTH98] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: fast neural network emulation and control of physics-based models. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 9–20, New York, NY, USA, 1998. ACM.

[Hal84] Daniel Conrad Halbert. *Programming by example*. PhD thesis, University of California, Berkeley, 1984.

[HHN85] Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. Direct manipulation interfaces. *Human Computer Interaction*, 1(4):311–338, 1985.

[HL02] Arash Habibi and Annie Luciani. Dynamic particle coating. *IEEE Transactions on Visualization and Computer Graphics*, 8(4):383–394, 2002.

[HL05] Chi-Min Hsieh and Annie Luciani. Generating dance verbs and assisting computer choreography. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 774–782, New York, NY, USA, 2005. ACM.

[HLT+03] Laure Heigeas, Annie Luciani, Joelle Thollot, and Nicolas Castagné. A physically-based particle model of emergent crowd behaviors. In *International Conference Graphicon 2003*, pages 1–9, september 2003.

[HLV96] Arash Habibi, Annie Luciani, and Alexis Vapillon. A physically-based model for the simulation of reactive turbulent objects. In *Proceeding of WSCG'96*, volume 1, pages 113–122. Plzen (Csech Republic), 1996.

[Hsi07] Chi Min Hsieh. *Grammaire de Mouvements Dansés par Modélisation Physique: Utilisation pour la Composition Chorégraphique*. PhD thesis, Institut National Polytechnique de Grenoble, 2007.

[Hub96] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. In *ACM Transaction on Graphics*, volume 15, pages 179–210, July 1996.

[Hut88] Edwin Hutchins. Metaphors for interface design. In M.M. Taylor, F. Neel, and D.G. Bouwhuis, editors, *The structure of multimodal dialogue*, pages 11–28. Amsterdam: North Holland, 1988.

[Ink] Inkscape. http://www.inkscape.org/?lang=fr

[Jim93] Stéphane Jimenez. *Modélisation et simulation physique d'objets volumiques déformables complexes*. PhD thesis, Institut National Polytechnique de Grenoble, 1993.

[JK96] Bonnie E. John and David E. Kieras. The goms family of user interface analysis techniques: comparison and contrast. *ACM Trans. Comput.-Hum. Interact.*, 3(4):320–351, 1996.

[JL93] Stéphane Jimenez and Annie Luciani. Animation of interacting objects with collisions and prolonged contacts. In B. Falcidien and T.L. Kunii, editors, *Modeling in computer graphics—methods and applications*, Proceedings of the IFIP WG 5.10 Working Conference, pages 129–141. Springer Verlag, 1993.

[JLL91] Stéphane Jimenez, Annie Luciani, and Christian Laugier. Physical modeling as an help for planning the motions of a land vehicle. In *IEEE/RSJ International Workshop on Intelligent Robot and Systems*, Osaka, 1991.

[Kay90] Alan Kay. *The Art of Computer Interface*, chapter User Interface : a personal view. Addison-Wesley, New York, NY, USA, 1990.

[KPo] KPovModeler. http://www.kpovmodeler.org/.

[Lau90] Brenda Laurel. *The Art of Computer Interface*, chapter Interface Agents : metaphor with character. Addison-Wesley, New York, NY, USA, 1990.

[LC84] Annie Luciani and Claude Cadoz. Modélisation et animation gestuelle d'objets - le système anima. In *CESTA - 1er colloque Image*, pages 183–189, Biarritz, 1984.

[LCJ+89] Annie Luciani, Claude Cadoz, Stéphane Jimenez, Jean-Loup Florens, and Olivier Raoult. Modèles comportementaux - vers une approche instrumentale de la synthèse d'images. In *Bigre+Globule Journées AFCET-GroPlan " Informatique Géométrique et graphique"*, Strasbourg, décembre 1989.

[LEC+06] Annie Luciani, Matthieu Evrard, Damien Couroussé, Nicolas Castagné, Claude Cadoz, and Jean-Loup Florens. A basic gesture and motion format for virtual reality multisensory applications. In José Braz, Joaquim A. Jorge, Miguel Dias, and Adérito Marcos, editors, *GRAPP 2006: Proceedings of the First International Conference on Computer Graphics Theory and Applications GRAPP'06*, pages 349–356, Setubal Portugal, 2006. INSTICC - Institute for Systems and Technologies of Information, Control and Communication.

[LG97] Annie Luciani and A. Godard. Simulation of Physical Object Construction Featuring Irreversible State Changes. In Vaclav Magnenat Thalmann Nadia Skala, editor, *WSCG' 97 The fifth international conference in central europe on Computer Graphic and Visualizatoin' 97 (WSCG' 97)*, pages 321–330, Plzen Tchèque, République, 1997. University of west bohemia department of computer science.

[Lie89] P. Lienhardt. Subdivisions of n-dimensional spaces and n-dimensional generalized maps. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 228–236, New York, NY, USA, 1989. ACM.

[LJC+91] Annie Luciani, Stéphane Jimenez, Claude Cadoz, Jean-Loup Florens, and Olivier Raoult. A unified view of multitude behavior, flexibility, plasticity and fractures: balls, bubbles and agglomerates. In *Proceedings of the IFIP WG*, pages 55–74, Tokyo, Japan, 1991. Springer Verlag.

[LJF+91] Annie Luciani, Stéphane Jimenez, Jean-Loup Florens, Claude Cadoz, and Olivier Raoult. Computational physics: a modeler simulator for animated physical objects. In Elsevier, editor, *Proceedings of the European Computarer Graphics Conference and Exhibition*, pages 425–436, Vienna, Austria, September 1991.

[LKG04] Frederic F. Leymarie, Benjamin B. Kimia, and Peter J. Giblin. Towards surface regularization via medial axis transitions. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3*, pages 123–126, Washington, DC, USA, 2004. IEEE Computer Society.

[LLD08] Nicolas Lassabe, Hervé Luga, and Yves Duthen. New skills for evolving artificial creatures. *International Journal of Information Technology and Intelligent Computing*, 1:120–146, Janvier 2008.

[Luc00] Annie Luciani. From granular avalanches to fluid turbulences through oozing pastes. A mesoscopic physically-based particle model. In *Graphicon 2000 Conference Proceedings Graphicon 2000*, pages 282–289, Moscou Russie, Fédération De, 2000.

[Luc04] Annie Luciani. Why and how physically-based model are able to model emergent crowd behaviors? In Dimitri Plemenos, editor, *Proceeding of 3IA conference*, pages 41–56, Limoges, 2004.

[Lug09] Hervé Luga. Expérimentations virtuelles: vie artificielle pour la génération de formes et de comportement. In ACROE-INPG, editor, *Actes de JIM 2009 Journées d'informatique musicales - JIM 2009*, pages pp93–103, Grenoble France, 04 2009.

[Man79] Benoit Mandelbrot. *Fractals: form, chance and dimension*. W.H.Freeman & Co, San Francisco (CA, USA), 1979.

[Man93] Emmanuel Manzotti. Modèle physique pour l'animation d'image. modèles d'objets à microstructure complexe (sables, amas, ...). Master's thesis, ENSIMAG, 1993.

[Mar98] J.C. Martin. Tycoon: theoretical and software tools for multimodal interfaces. In John Lee (Ed.), editor, *Intelligence and Multimodality in Multimedia interfaces. AAAI Press.*, 1998.

[Max] Max msp jitter. http://cycling74.com/products/maxmspjitter/

[May] Maya. http://www.autodesk.fr/adsk/servlet/pc/index?siteID=458335&id=14657767

[MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Mil88] Gavin S. P. Miller. The motion dynamics of snakes and worms. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 169–173, New York, NY, USA, 1988. ACM.

[MP89] Gavin Miller and Andrew Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics*, 13(3):305–309, 1989.

[MTP+04] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 449–456, New York, NY, USA, 2004. ACM.

[NC93] Laurence Nigay and Joëlle Coutaz. A design space for multimodal interfaces: concurrent processing and data fusion. In *INTERCHI'93 Proceedings, Amsterdam, S. Ashlung, K. Mullet, A. Henderson, E. Hollnagel, T. White Eds., ACM New York Publ*, pages 172–178, 1993.

[NC96] Laurence Nigay and Joëlle Coutaz. Espaces conceptuels pour l'interaction multimédia et multimodale. *TSI, spécial Multimédia et Collecticiel, AFCET*, Vol 15(9):1195–1225, 1996.

[Phy] Physx. http://www.nvidia.com/object/physx_new.html

[PL99] Gilles Pommereuil and Annie Luciani. Un langage de modélisation des comportements complexes de la matière pour la construction d'objets physiques simulables. In *Proceedings de MICAD'99*, Paris (Palais des expositions), 1999. Hermès.

[Pro95] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In Wayne A. Davis and Przemyslaw Prusinkiewicz, editors, *Graphics Interface '95*, pages 147–154. Canadian Human-Computer Communications Society, 1995.

[Raz86] *Le système ANIMA : éditeur d'objets producteurs d'images; implantation d'algorithmes de simulation temps réel*. PhD thesis, Université scientifique, technologique et médicale de Grenoble et institut national polytechnique de Grenoble, 1986.

[Rea] RealFlow. http://realflow.com

[Ree83] William T. Reeves. Particle systems—a technique for modeling a class of fuzzy objects. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 359–375, New York, NY, USA, 1983. ACM.

[Rey82] Craig W. Reynolds. Computer animation with scripts and actors. In *SIGGRAPH '82: Proceedings of the 9th annual conference on Computer graphics and*

interactive techniques, pages 289–296, New York, NY, USA, 1982. ACM.

[Rey87] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, 1987.

[SDE05] Joshua Schpok, William Dwyer, and David S. Ebert. Modeling and animating gases with simulation features. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 97–105, New York, NY, USA, 2005. ACM.

[SEL07] Kevin Sillam, Matthieu Evrard, and Annie Luciani. A real-time implementation of the dynamic particle coating method on a GPU architecture. In *Proceedings of VRIPHYS'07*, Dublin, 2007.

[Shn84] Ben Shneiderman. The future of interactive systems and the emergence of direct manipulation. In *Proc. of the NYU symposium on user interfaces on Human factors and interactive computer systems*, pages 1–28, Norwood, NJ, USA, 1984. Ablex Publishing Corp.

[Shn87] Ben Shneiderman. *Designing the user interface: professional development courses from the University of Maryland*. University of Maryland at College Park, College Park, MD, USA, 1987.

[Sim] Simulink. http://www.mathworks.com/products/simulink/

[Sim94] Karl Sims. Evolving virtual creatures. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22, New York, NY, USA, 1994. ACM.

[Smi84] Alvy Ray Smith. Plants, fractals, and formal languages. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 1–10, New York, NY, USA, 1984. ACM.

[Sod] Sodaplay. http://sodaplay.com

[Sta99] Jos Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[Sut63] Ivan E. Sutherland. Sketchpad: a man-machine graphical communication system. In *AFIPS '63 (Spring): Proceedings of the May 21-23, 1963, spring joint computer conference*, pages 329–346, New York, NY, USA, 1963. ACM.

[TF88] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation: viscolelasticity, plasticity, fracture. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 269–278, New York, NY, USA, 1988. ACM.

[TKH+04] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M. p. Cani, F. Faure, N. Magnenat-thalmann, W. Strasser, and P." Volino. Collision detection for deformable objects. *COMPUTER GRAPHICS FORUM*, 24:61–81, 2004.

[Ton91] David Tonnesen. Modeling liquids and solids using thermal particles. In *Graphics Interface '91*, pages 255–262, June 1991.

[TPF89] Demetri Terzopoulos, John Platt, and Kurt Fleischer. Heating and melting deformable models (From goop to glop). In *Graphics Interface '89*, pages 219–226, June 1989.

[TT94] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: physics, locomotion, perception, behavior. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 43–50, New York, NY, USA, 1994. ACM.

[vdPF93] Michiel van de Panne and Eugene Fiume. Sensor-actuator networks. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 335–342, New York, NY, USA, 1993. ACM.

[vdPKF94] Michiel van de Panne, Ryan Kim, and Eugene Flume. Virtual wind-up toys for animation. *Proceedings of Graphics Interface '94*, pages 208–215, 1994.

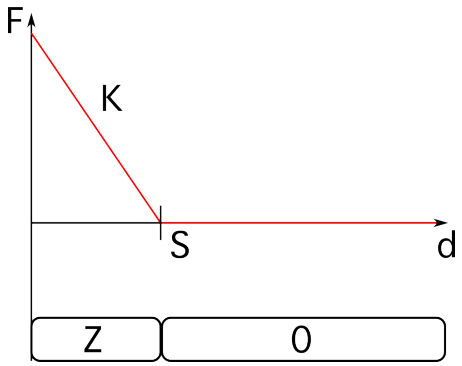
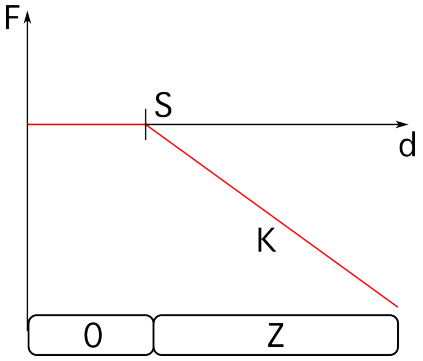
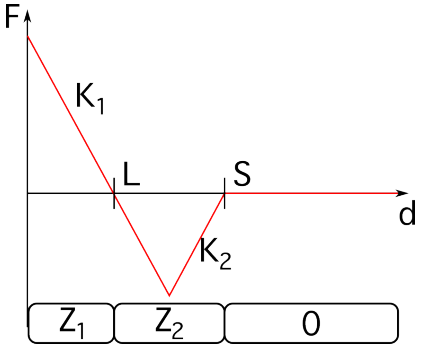
[VN00] Frédéric Vernier and Laurence Nigay. A framework for the combination and characterization of output modalities. In *DSV-IS2000, Limerick (IR), Lecture Notes in Computer Sciences, Springer-Verlag publ*, pages 32–48, 2000.

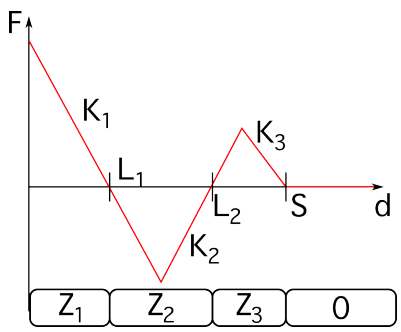
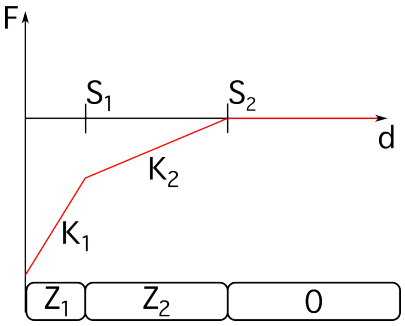
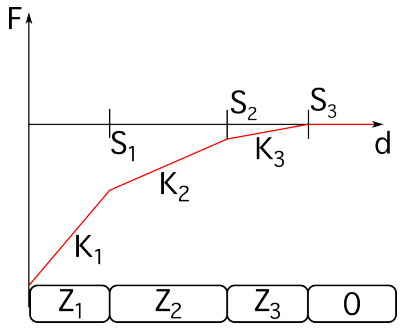
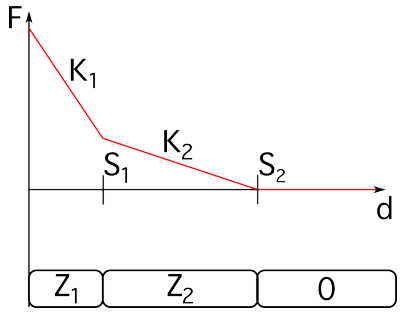
[WK88] Andrew Witkin and Michael Kass. Spacetime constraints. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 159–168, New York, NY, USA, 1988. ACM.

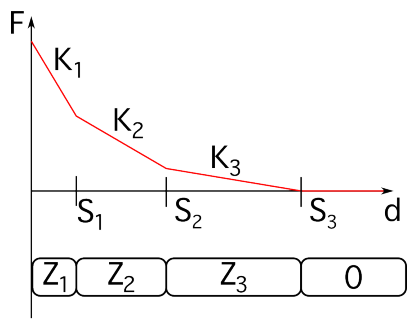
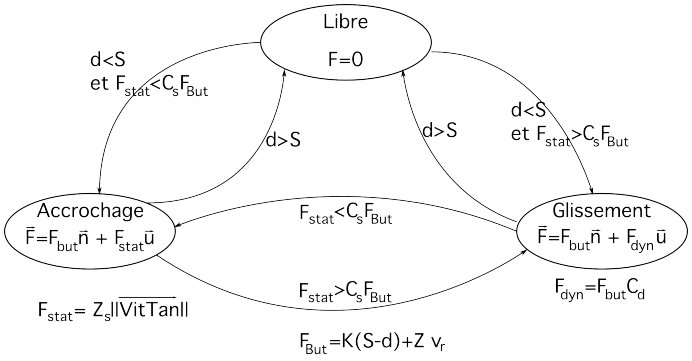
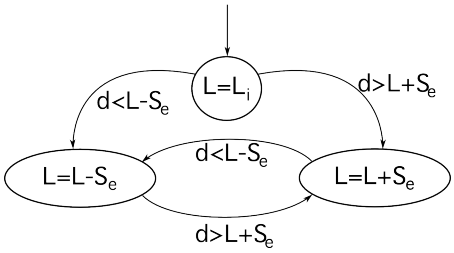
[Zac02] Gabriel Zachmann. Minimal hierarchical collision detection. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 121–128, New York, NY, USA, 2002. ACM.

[Zia07] Mouna Ziat. *Enaction and Enactive Interfaces, a Handbook of Terms*, chapter Zoomable Interfaces, pages 164–167. Enactive Systems Books, 2007. A. Luciani and C. Cadoz editors.

Annexe 1 : Les modules de MIMESIS 4

Modules <LIA>		
Nom et description des interactions	Liste des paramètres	Caractéristique force-distance. Toutes les interactions sont également visqueuses avec des seuils en distance. La représentation graphique montre donc les différents coefficients de viscosité en fonction de la distances
<BUT> interaction de butée visco-élastique	K, Z, S	
<BUL> interaction de bulle visco-élastique	K, Z, S	
<COH3> interaction de cohésion	K_1, K_2, Z_1, Z_2, L, S	

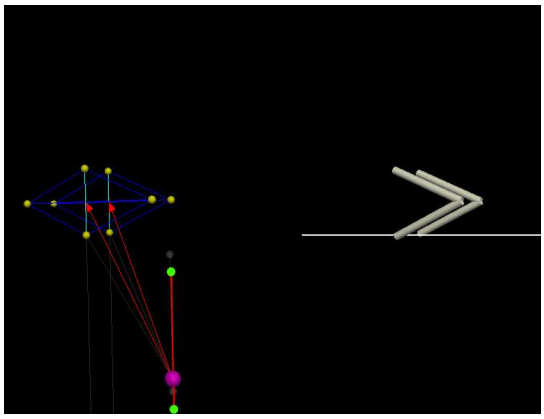
<COH4> interaction de cohésion	K_1, K_2, K_3, Z_1, Z_2, Z_3, L, S	
<ATR3> interaction d'attraction à trois paliers	K_1, K_2, Z_1, Z_2, S_1, S_2	
<ATR4> interaction d'attraction à quatre paliers	K_1, K_2, K_3, Z_1, Z_2, Z_3, S_1, S_2, S_3	
<REP3> interaction de répulsion à trois paliers	K_1, K_2, Z_1, Z_2, S_1, S_2	

<REP4> interaction de répulsion à quatre paliers	K_1, K_2, K_3, Z_1, Z_2, Z_3, S_1, S_2, S_3	
<FROS> interaction de frottement sec	K, Z, S, C_s, C_d, Z_s	 $\overrightarrow{VitTan}$ est le vecteur vitesse tangentiel, c'est à dire la projection de la vitesse relative dans le plan orthogonal à $\overrightarrow{M_1M_2}$ $\overrightarrow{M_1M_2}$=d $\vec{n}$ $\overrightarrow{VitTan}$=||$\overrightarrow{VitTan}$||$\vec{u}$
<PLAST> interaction de plasticité. Interaction visco-élastique dont la longueur à vide est déterminé par l'automate d'état à mémoire ci-contre	K, Z, Se (Seuil d'élasticité), Li (Longueur au repos initiale)	

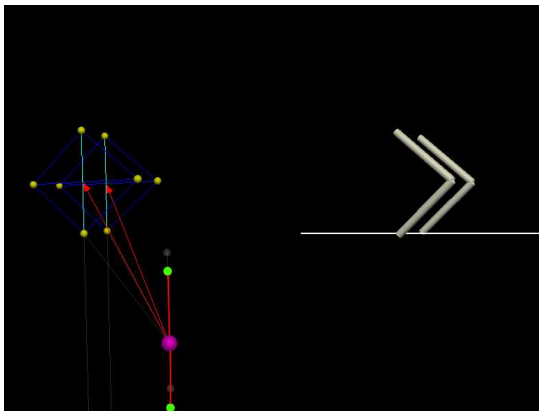
Modules <MAT>	
Nom et description des éléments matériels	Liste des paramètres
<MAS> Masse mobile intégrant une force de gravité réglable	M, G_x, G_y, G_z
<MASV> Masse mobile intégrant une force de gravité réglable et une viscosité de milieu	M, G_x, G_y, G_z, Z
<SOL> Masse fixe représentant une masse d'inertie infinie	Néant
<GP> Module générateur de position. Sa position est entièrement définie par un fichier geste au format GMS.	Néant

Annexe 2 : Le saut et ses différentes étapes

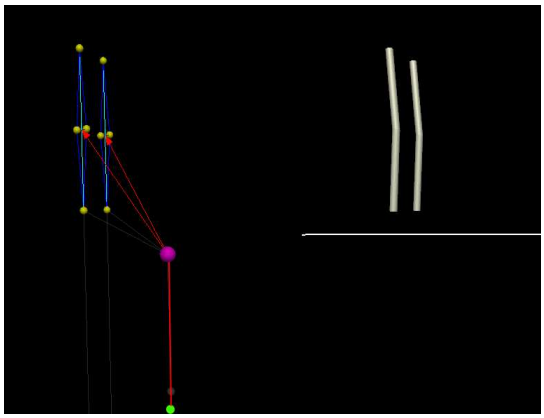
Le saut est ici illustré depuis la première extension jusqu'au début de la deuxième. A gauche, on peut observer le modèle et son contrôleur, à droite, un habillage élémentaire du modèle.



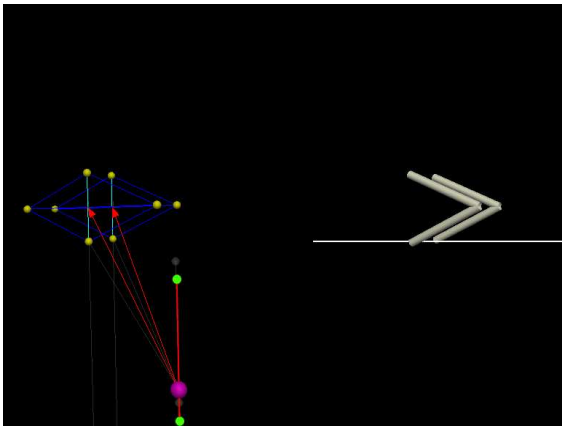
1



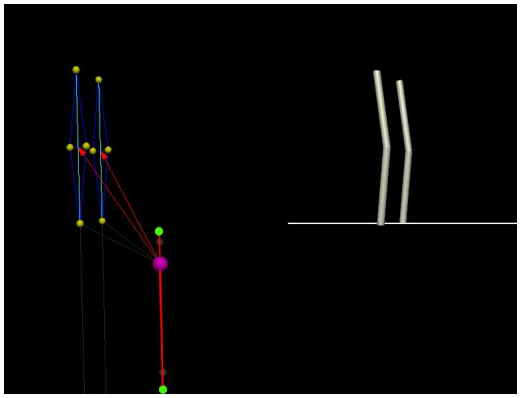
4



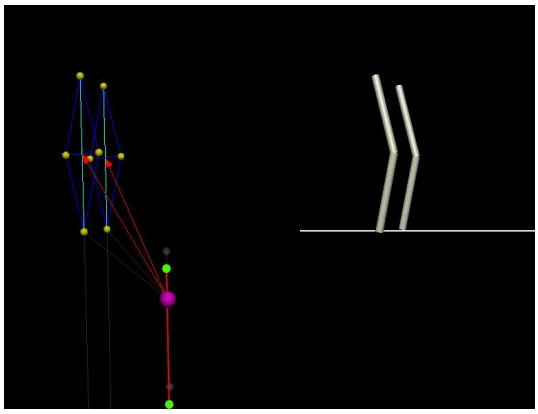
2



5



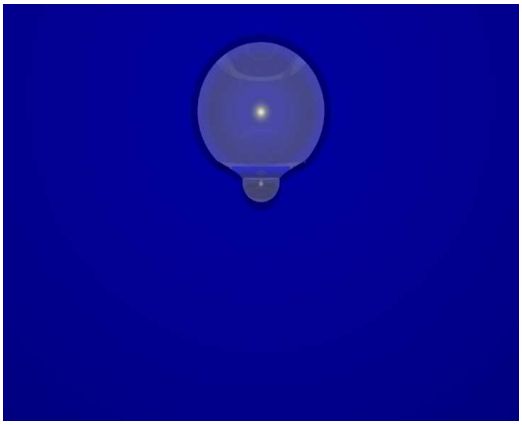
3



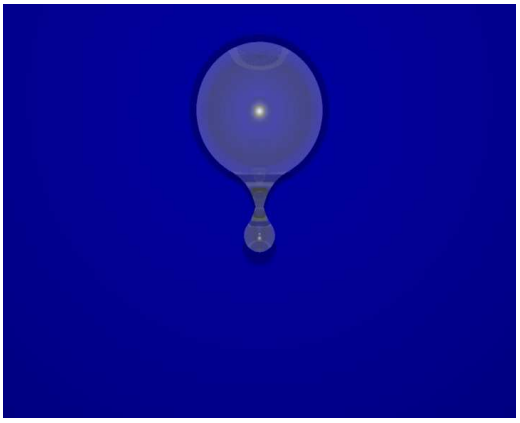
6

Annexe 3 : La goutte

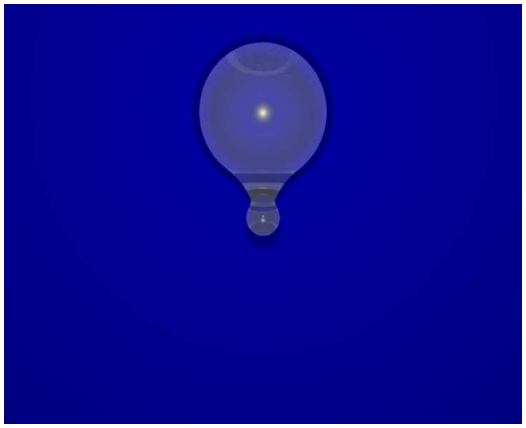
Le phénomène de striction, dont une modélisation est détaillée au chapitre II §4.2, habillé à l'aide de surfaces implicites dans le logiciel POV-Ray.



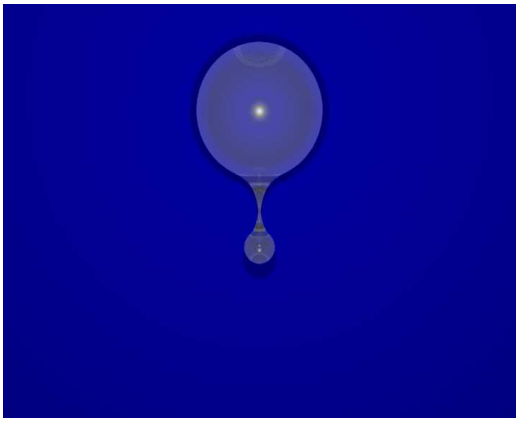
1



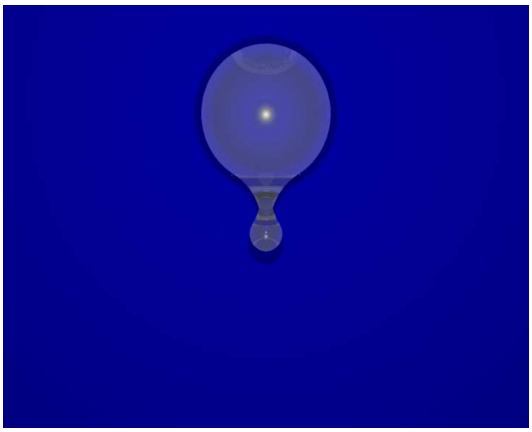
4



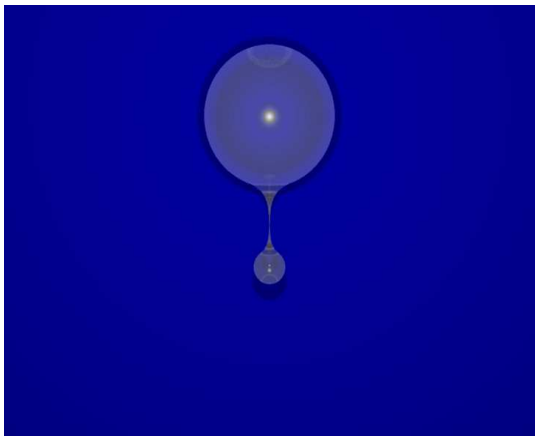
2



5



3

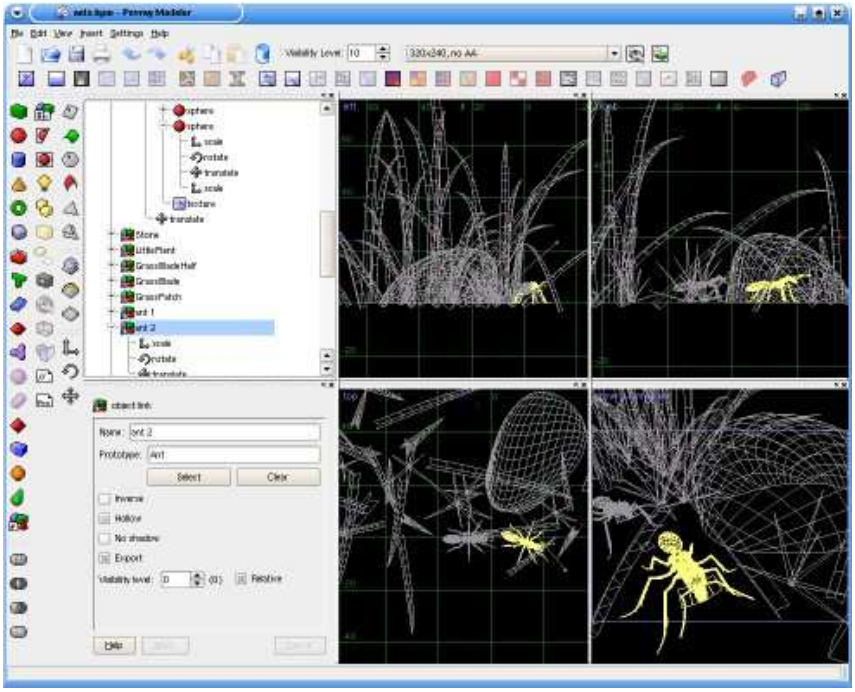


6

Annexe 4 : KPovModeler, un exemple de réification du script

Présentation

KPoModeler est un modeleur de scènes pour le logiciel de lancer de rayon POV-Ray. Une scène POV-Ray est décrite par un fichier texte contenant une description des objets de la scène et de l'illumination. Cette description se fait avec le POV-Ray *scene description language* (SDL). KPovModeler permet de générer un tel fichier en s'appuyant sur une palette graphique d'objets. On place ces objets dans un arbre de scène. L'arbre de scène devient alors la représentation graphique du fichier texte écrit en SDL et qui représente la scène 3D. Les paramètres des objets peuvent être réglés dans une boîte de dialogue ou par manipulation directe dans une fenêtre 3D lorsque c'est possible. Le choix de la structure en arbre se justifie par l'aspect hiérarchique des objets : un objet va avoir ses propriétés primaires qui lui sont propres (par exemple pour une sphère son centre et son rayon) et des propriétés secondaires décrites par des objets fils (par exemple la texture, une transformation spatiale comme la translation ou le redimensionnement, etc...). La structure d'arbre est aussi intéressante pour les objets composés (CSG, Blobs).



KPovModeler offre de plus un type d'objet qui nous intéresse particulièrement : l'objet "code povray brut" qui permet d'insérer dans l'arbre une portion de *scene description language*. Ainsi ce qui n'est pas accessible par la palette graphique, notamment tout ce qui est structure de contrôle (condition, boucle, etc...) peut être inséré dans la scène. Cet objet est un parfait exemple de réification d'un script.

Deux autres objets retiennent notre attention :

1. Déclaration : c'est un objet conteneur. Il contient tous ses fils et est référencé

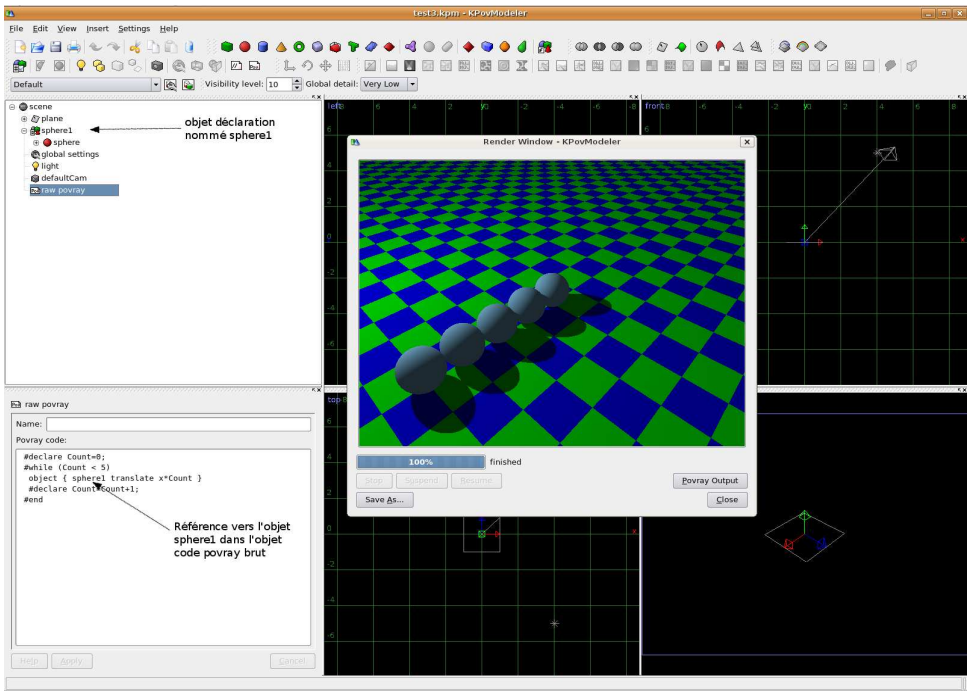
par son seul paramètre : son nom. Si on ne fait aucune référence à un objet déclaration dans l'arbre de scène, ses fils ne sont pas dans la scène.

2. Lien objet : lorsqu'un objet déclaration contient des objets géométriques, on utilise le lien objet pour y faire référence.

En résumé, l'objet d'intérêt est la scène 3D, c'est-à-dire l'objet informatique qui va être traité par l'algorithme de lancer de rayons. Cet objet a deux représentations : le fichier de *scene description language* et l'arbre de scène. L'arbre de scène est une représentation de la scène parce qu'il est une représentation graphique d'un fichier de *scene description language.* En effet, chacun de ses éléments représente une portion de *scene description language*, explicitement ("code povray brut") ou non (éléments de la palette graphique).

Un exemple de coopération entre modalités graphique et textuelle

L'exemple exploite la possibilité de faire référence à un objet créé avec la palette graphique dans un objet "code povray brut". L'utilisateur a créé un objet géométrique qu'il veut dupliquer 100 fois et placer selon une ligne. Il insère cet objet comme fils d'un objet Déclaration. Il crée un objet "code povray brut" dans lequel une boucle appelle à chaque itération l'objet Déclaration et une translation.



Cet exemple montre que l'association des deux modalités permet d'arriver à son but de manière compacte et efficace. Mais c'est surtout le rôle de la réification dans l'association qui paraît intéressant. Elle permet de travailler le script pour lui-même et l'objet dupliqué peut être travaillé en manipulation directe. Nous observons donc un cas

intéressant de collaboration horizontale, mise en œuvre grâce à l'objet déclaration, dont on note la ressemblance avec la notion de label.

Les limites

1ère limite :

On a vu dans l'exemple comment le "code povray brut" pouvait faire référence à un objet créé graphiquement. Inversement, faire référence à un objet code povray dans un objet créé graphiquement est impossible. Concrètement, un objet code povray brut ne peut être le fils d'un objet Déclaration, mais il se manipule librement comme tout autre objet dans l'arbre de scène.

2ème limite :

On ne peut pas passer d'une représentation à l'autre. C'est-à-dire qu'un objet "code povray brut" ne peut pas être transformé en un sous-arbre de scène et inversement. Pour reprendre l'exemple, on aurait pu souhaiter transformer l'objet "code povray brut" représentant les 100 sphères en 100 objets sphère, pour par exemple changer le rayon d'une des sphères ou en supprimer une.